Intel® Stratix® 10 L- and H-Tile Transceiver PHY User Guide

Subscribe

Send Feedback

UG-20055 | 2020.03.03

Latest document on the web: PDF | HTML

Contents

1. Overview

Intel® Stratix® 10 devices offer up to 144 transceivers with integrated advanced high-speed analog signal conditioning and clock data recovery circuits for chip-to-chip, chip-to-module, and backplane applications.

The Intel Stratix 10 devices contain a combination of GX, GXT, or GXE channels, in addition to the hardened IP blocks for PCI Express* and Ethernet applications.

The Intel Stratix 10 device introduces several transceiver tile variants to support a wide variety of protocol implementations. These transceiver tile variants are L-tiles, H-tiles, and E-tiles. This user guide describes both the L- and H-tile transceivers. For Intel Stratix 10 devices that only contain E-tiles, refer to the *E-Tile Transceiver PHY User Guide*.

Table 1. Transceiver Tile Variants—Comparison of Transceiver Capabilities

Feature	L-Tile (GX, SX)	H-Tile (GX, SX, TX, MX)	E-Tile (TX, MX)
Maximum Transceiver Data Rate (Chip-to-chip)	GX [1]—17.4 Gbps GXT [1]—26.6 Gbps	GX—17.4 Gbps GXT—28.3 Gbps	GXE [2]—57.8 Gbps Pulse Amplitude Modulation 4 (PAM4)/28.9 Gbps Non-return to zero (NRZ)
Maximum Transceiver Data Rate (Backplane)	GX—12.5 Gbps GXT—12.5 Gbps		
Number of Transceiver Channels (per tile)	GX—16 per tile GXT—8 per tile Total—24 per tile (4 banks, 6 channels per bank)	GX—8 per tile GXT—16 per tile Total—24 per tile (4 banks, 6 channels per bank)	GXE—24 individual channels per tile
Hard IP (per tile)	PCIe*—Gen3 x16	PCIe—Gen3 x16, SR-IOV (4 PF, 2K VF) Ethernet—50/100GbE MAC	Ethernet—100GbE MAC and RS (528, 514)-FEC, 4 per tile Ethernet—KP-FEC, 4 per tile Ethernet—10/25GbE MAC and RS (528, 514)-FEC, 24 per tile

In all Intel Stratix 10 devices, the various transceiver tiles connect to the FPGA fabric using Intel EMIB (Embedded Multi-Die Interconnect Bridge) technology.

Related Information

- L-Tile/H-Tile Building Blocks on page 15
- See *AN 778: Intel Stratix 10 Transceiver Usage* for transceiver channel placement guidelines for L-tiles and H-tiles.

(1) Refer to the *L-Tile/H-Tile Building Blocks* section for further descriptions of GX and GXT channels.

(2) Refer to the *E-Tile Transceiver PHY User Guide* for a full description of GXE channels.

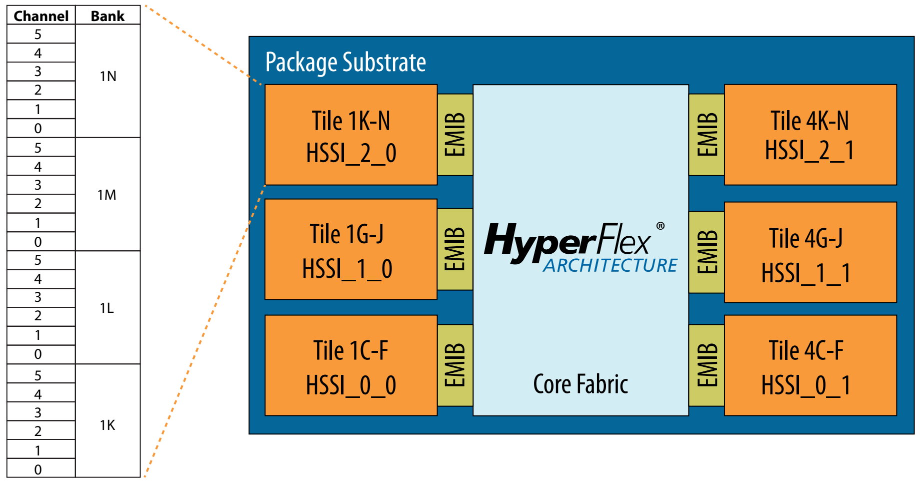
- Intel Stratix 10 GX/SX Device Overview
- Intel Stratix 10 MX (DRAM System-in-Package) Device Overview
- Intel Stratix 10 TX Device Overview
- E-Tile Transceiver PHY User Guide
- Intel FPGA IP for Transceiver PHY—Support Center

1.1. L-Tile/H-Tile Layout in Intel Stratix 10 Device Variants

Intel Stratix 10 GX/SX device variants support both L- and H-Tiles. Intel Stratix 10 TX and MX device variants support both H- and E-Tiles.

Intel Stratix 10 devices are offered in a number of different configurations based on layout. There is a maximum of six possible locations for a tile. The following figure maps these layouts to the corresponding transceiver tiles and banks.

Figure 1. Intel Stratix 10 Tile Layout



1.1.1. Intel Stratix 10 GX/SX H-Tile Configurations

The Intel Stratix 10 GX FPGAs meet the high-performance demands of high-throughput systems with up to 10 teraflops (TFLOPs) of floating-point performance. Intel Stratix 10 GX FPGAs also provide transceiver support up to 28.3 Gbps for chip-module, chip-to-chip, and backplane applications.

The Intel Stratix 10 SX SoCs features a hard processor system with 64 bit quad-core ARM* Cortex*-A53 processor available in all densities, in addition to all the features of Intel Stratix 10 GX devices.

Figure 2. Intel Stratix 10 GX/SX Device with 1 H-Tile (24 Transceiver Channels)

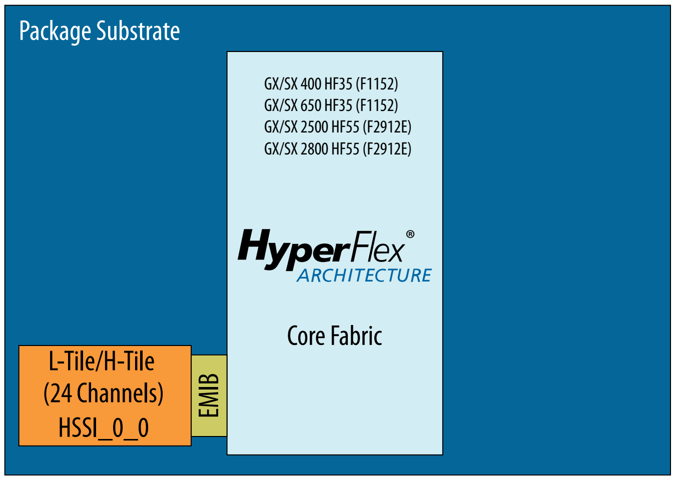


Figure 3. Intel Stratix 10 GX/SX Device with 2 H-Tiles (48 Transceiver Channels)

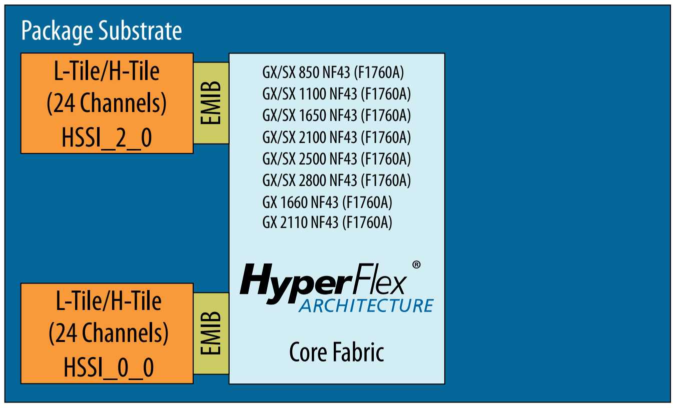
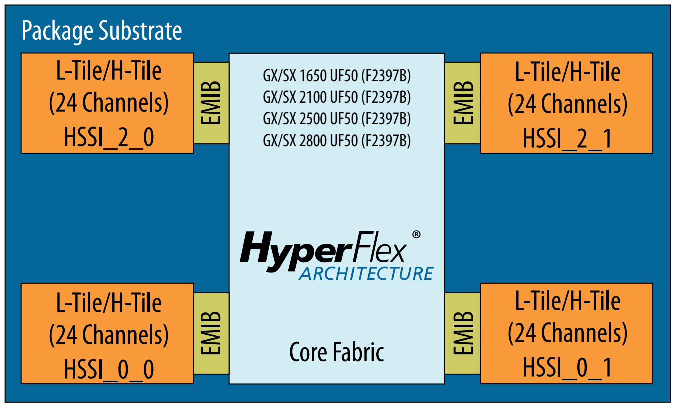


Figure 4. Intel Stratix 10 GX/SX Device with 4 H-Tiles (96 Transceiver Channels)



1.1.2. Intel Stratix 10 TX H-Tile and E-Tile Configurations

The Intel Stratix 10 TX FPGAs deliver the most advanced transceiver capabilities in the industry by combining H-Tile and E-Tile transceivers.

Figure 5. Intel Stratix 10 TX Device with 1 E-Tile and 1 H-Tile (48 Transceiver Channels)

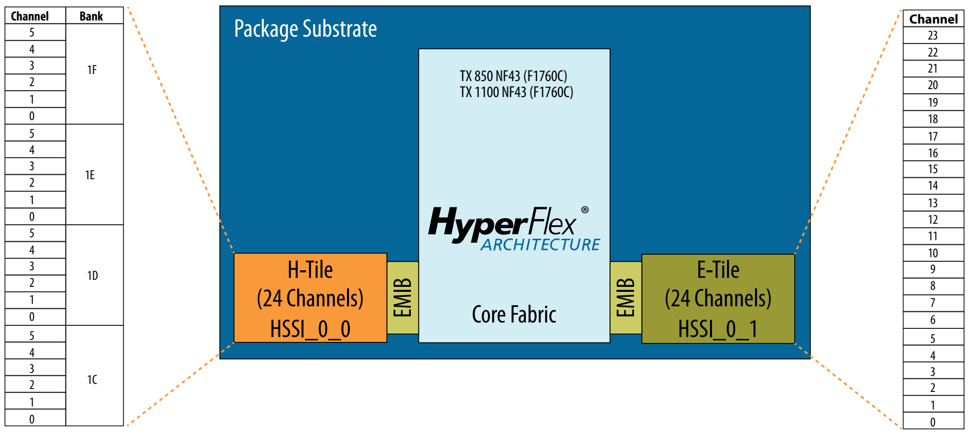


Figure 6. Intel Stratix 10 TX Device with 2 E-Tiles and 1 H-Tile (72 Transceiver Channels)

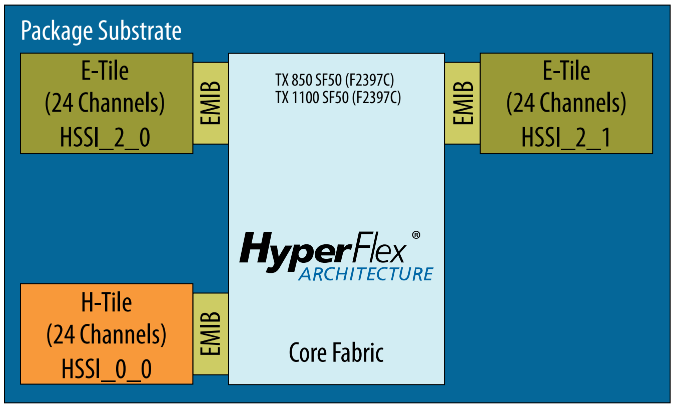


Figure 7. Intel Stratix 10 TX Device with 3 E-Tiles and 1 H-Tile (96 Transceiver Channels)

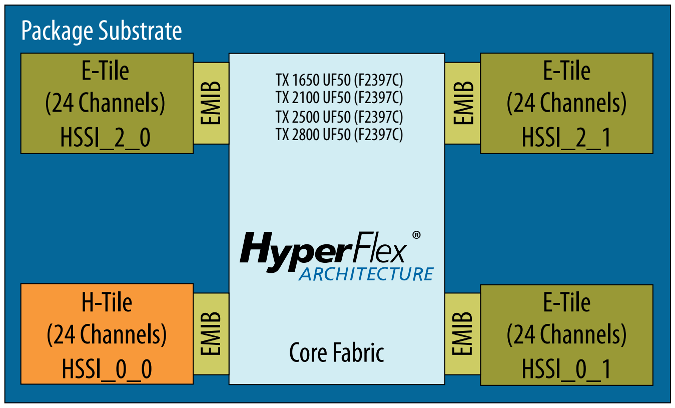
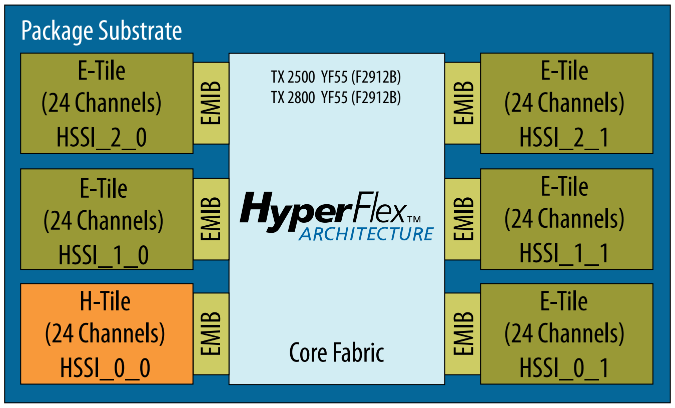


Figure 8. Intel Stratix 10 TX Device with 5 E-Tiles and 1 H-Tile (144 Transceiver Channels)



Note:
1. No package migration available between GX/SX and TX device families (H-Tile and E-Tile)
2. Migration available within GX/SX from L-Tile to H-Tile variants

1.1.3. Intel Stratix 10 MX H-Tile and E-Tile Configurations

The Intel Stratix 10 MX devices combine the programmability and flexibility of Intel Stratix 10 FPGAs and SoCs with 3D stacked high-bandwidth memory 2 (HBM2). The DRAM memory tile physically connects to the FPGA using Intel Embedded Multi-Die Interconnect Bridge (EMIB) technology.

Figure 9. Intel Stratix 10 MX Device with 2 H-Tiles (48 Transceiver Channels) and 2 HBM2

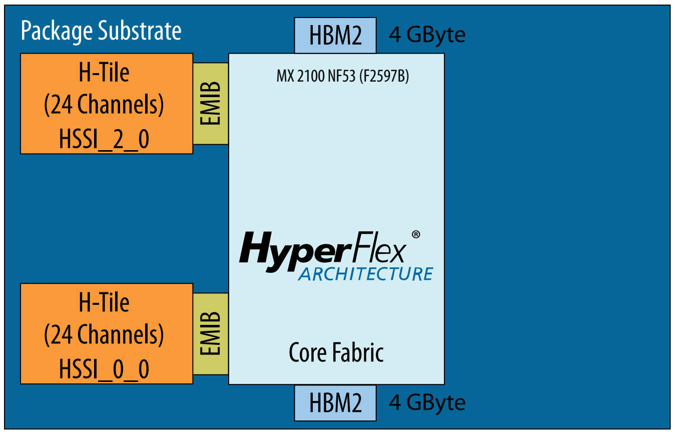


Figure 10. Intel Stratix 10 MX Device with 4 H-Tiles (96 Transceiver Channels) and Two 4 GB HBM2

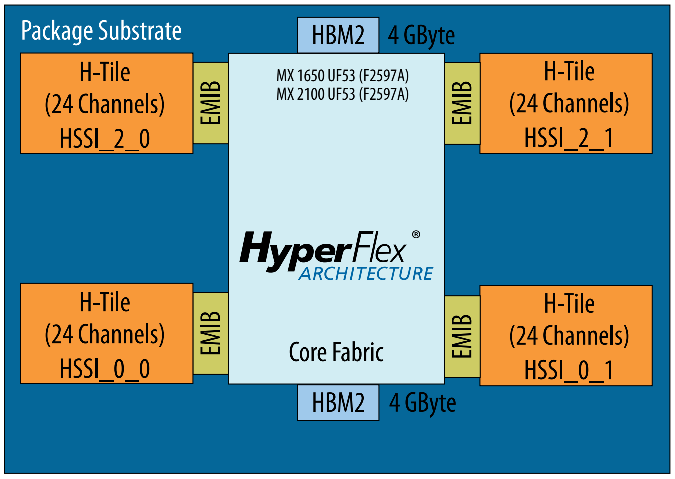


Figure 11. Intel Stratix 10 MX Device with 4 H-Tiles (96 Transceiver Channels) and Two 8 GB HBM2

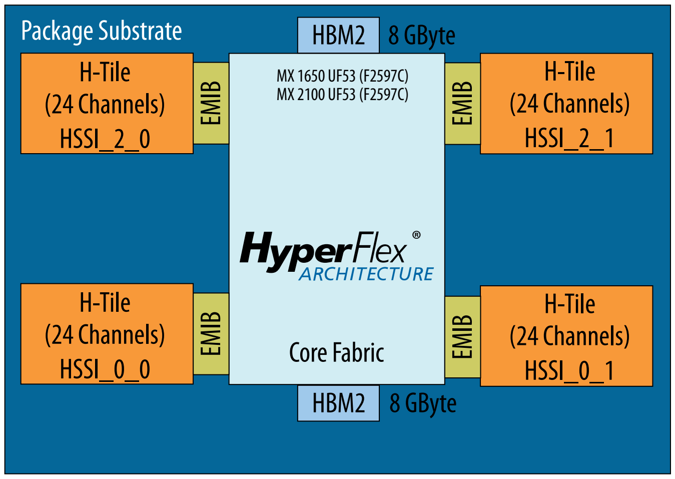
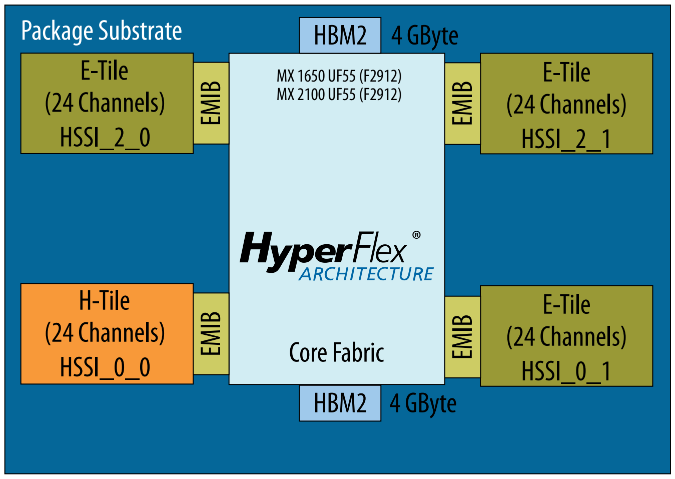


Figure 12. Intel Stratix 10 MX Device with 3 E-Tiles, 1 H-Tile (96 Transceiver Channels) and 2 HBM2



1.2. L-Tile/H-Tile Counts in Intel Stratix 10 Devices and Package Variants

Table 2. L-Tile/H-Tile Counts in Intel Stratix 10 GX/SX Devices (HF35, NF43, UF50, HF55)

The number in the Intel Stratix 10 GX/SX Device Name column indicates the device's Logic Element (LE) count (in thousands LEs).

Intel Stratix 10 GX/SX Device Name	F1152 HF35 (35x35 mm^2)	F1760A NF43 (42.5x42.5 mm^2)	F2397B UF50 (50x50 mm^2)	F2912E HF55 (55x55 mm^2)
GX 400/ SX 400	1			
GX 650/ SX 650	1			

continued...

Intel Stratix 10 GX/SX Device Name	F1152 HF35 (35x35 mm^2)	F1760A NF43 (42.5x42.5 mm^2)	F2397B UF50 (50x50 mm^2)	F2912E HF55 (55x55 mm^2)
GX 850/ SX 850		2		
GX 1100/ SX 1100		2		
GX 1650/ SX 1650		2	4	
GX 2100/ SX 2100		2	4	
GX 2500/ SX 2500		2	4	1
GX 2800/ SX 2800		2	4	1
GX 1660		2		
GX 2110		2		

Table 3. H- and E-Tile Counts in Intel Stratix 10 TX Devices (HF35, NF43, SF50, UF50, YF55)

The number in the Intel Stratix 10 TX Device Name column indicates the device's Logic Element (LE) count (in thousands LEs).

Cell legend: H-Tile count, E-Tile count

Intel Stratix 10 TX Device Name	F1152 HF35 (35x42.5 mm^2)	F1760C NF43 (42.5x42.5 mm^2)	F2397C SF50, UF50 (50x50 mm^2)	F2912B YF55 (55x55 mm^2)
TX 850	—	1, 1	1, 2	—
TX 1100	—	1, 1	1, 2	—
TX 1650	—	—	1, 3	—
TX 2100	—	—	1, 3	—
TX 2500	—	—	1, 3	1, 5
TX 2800	—	—	1, 3	1, 5

Table 4. H- and E-Tile Counts in Intel Stratix 10 MX Devices (NF53, UF53, UF55)

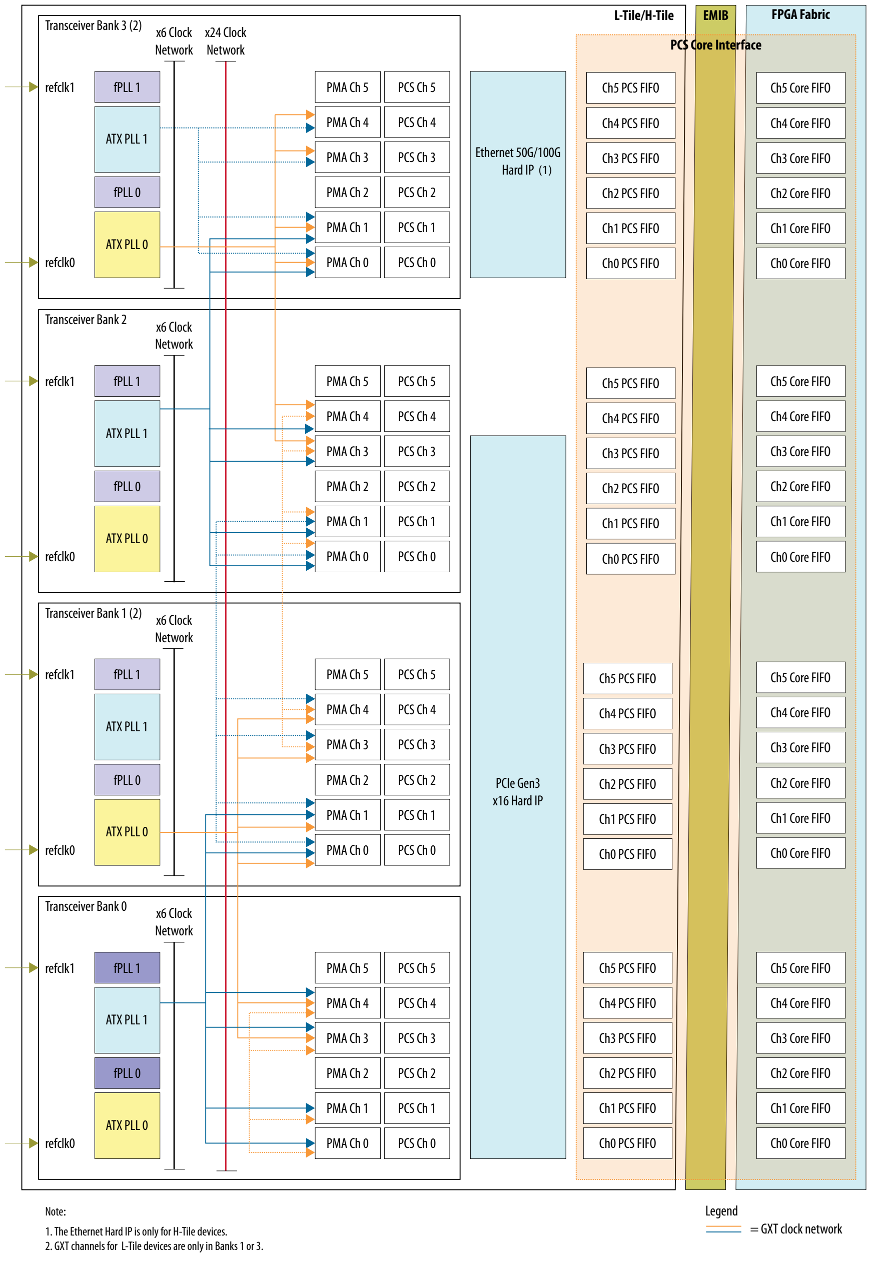
The number in the Intel Stratix 10 MX Device Name column indicates the device's Logic Element (LE) count (in thousands LEs).

Cell legend: H-Tile count, E-Tile count

Intel Stratix 10 MX Device Name	F2597A UF53 (52.5x52.5 mm^2)	F2597B NF53 (52.5x52.5 mm^2)	F2597C UF53 (52.5x52.5 mm^2)	F2912 UF55 (55x55 mm^2)
MX 1650	4, 0	—	4, 0	1, 3
MX 2100	4, 0	2, 0	4, 0	1, 3

1.3. L-Tile/H-Tile Building Blocks

Figure 13. High Level Block Diagram of L-Tile/H-Tile in Intel Stratix 10 Devices



Note:
1. The Ethernet Hard IP is only for H-Tile devices.
2. GXT channels for L-Tile devices are only in Banks 1 or 3.

1.3.1. Transceiver Bank Architecture

Each L-Tile/H-tile transceiver tile contains four transceiver banks. The transceiver channels are grouped into transceiver banks, where each bank has six channels. These six channels are a combination of GX and GXT channels which you can configure in the following ways:

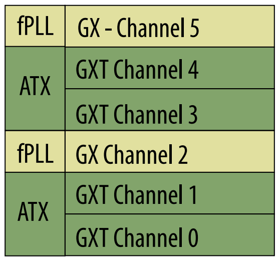
- All six channels as GX channels
- Channels 0, 1, 3, and 4 as GXT channels. L-Tile supports GXT channels in banks 1 and 3. H-Tile supports GXT channels in banks 0, 1, 2, and 3.
- All six channels as a mix of GX and GXT channels; for example, two GX channels and four GXT channels on H-Tile Devices. On L-Tile devices, you can use a maximum of four channels in a bank when any channel is configured as a GXT channel.

Each channel can also run in any of the following operational modes:

- **Duplex** (default)—Specifies a single channel that supports both transmission and reception
- **Transmitter (TX) Simplex**—Specifies a single channel that supports only transmission
- **Receiver (RX) Simplex**—Specifies a single channel that supports only reception

Each transceiver bank contains two Advanced Transmit (ATX) PLLs, two fractional PLLs (fPLL), and two Clock Multiplier Unit (CMU) PLLs.

Figure 14. Transceiver Banks in the L-Tile/H-Tile



Related Information

PLLs and Clock Networks on page 249

1.3.2. Transceiver Channel Types

Each transceiver has a Physical Coding Sublayer (PCS) and a Physical Medium Attachment (PMA). Additionally, each transceiver has loopback modes and internal pattern generator and verifier blocks for debugging.

1.3.2.1. GX Channel

Each GX transceiver channel has four types of PCS blocks that together support continuous datarates up to 17.4 Gbps. The various PCS blocks contain data processing functions such as encoding or decoding, scrambling or descrambling, word alignment, frame synchronization, FEC, and so on.

Figure 15. GX Transceiver Channel in TX/RX Duplex Mode

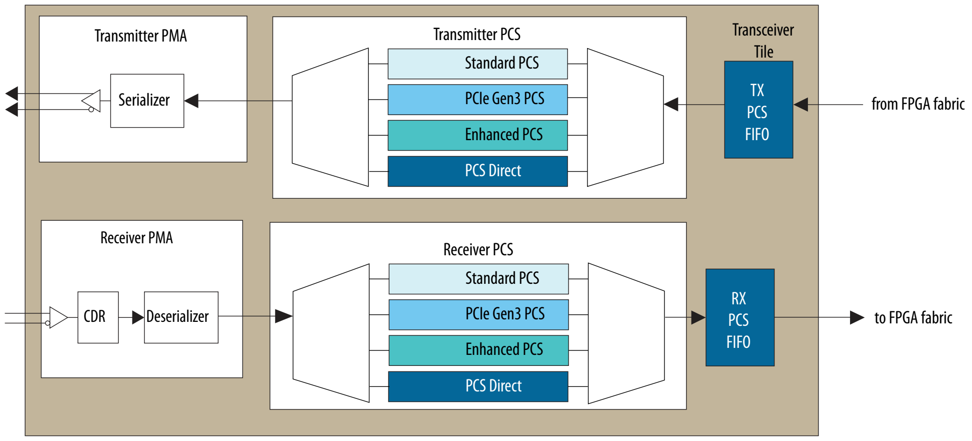


Table 5. PCS Types Supported by GX Transceiver Channels

PCS Type	L-Tile Production		H-Tile Production		
	-2 Speed Grade	-3 Speed Grade	-1 Speed Grade	-2 Speed Grades	-3 Speed Grade
Standard PCS	12 Gbps[3] or 10.81344 Gbps[4]	9.8304 Gbps[4]	12 Gbps[3] or 10.81344 Gbps[4]	12 Gbps[3] or 10.81344 Gbps[4]	9.8304 Gbps[4]
Enhanced PCS	17.4 Gbps				
PCIe Gen3 PCS	8 Gbps				
PCS Direct	17.4 Gbps				

Note: Use the L-Tile/H-Tile Transceiver Native PHY Intel Stratix 10FPGA IP Parameter Editor to determine the datarate limitations of your selected PCS configuration.

Refer to Table 12 on page 38 for a definition of the PCS Direct mode.

1.3.2.2. GXT Channel

Each GXT transceiver channel has two types of PCS blocks that together support continuous datarates up to 28.3 Gbps for H-Tile and 26.6 Gbps for L-Tile. Use PCS Direct or Enhanced PCS to implement a GXT channel.

Refer to the *Intel Stratix 10 Device Datasheet* for more details on transceiver specifications.

(3) The 12 Gbps data rate at the receiver is only supported when the **RX word aligner mode** parameter is set to **Manual**.

(4) This data rate is only supported when **Byte Serializer and Deserializer** mode is enabled.

Figure 16. GXT Transceiver Channel in TX/RX Duplex Mode

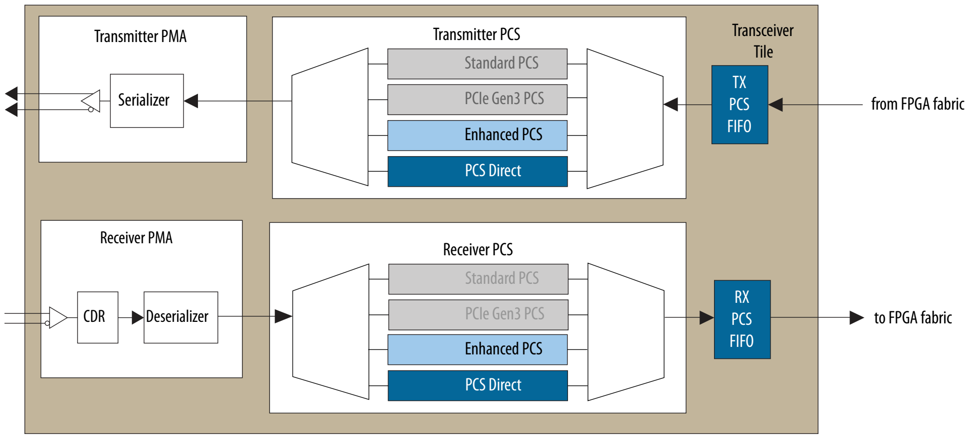


Table 6. PCS Types Supported by GXT Transceiver Channels

PCS Type	L-Tile Production		H-Tile Production		
	-2 Speed Grade	-3 Speed Grade	-1 Speed Grade	-2 Speed Grades	-3 Speed Grade
Enhanced PCS	26.6 Gbps	No GXT	28.3 Gbps	26.6 Gbps	No GXT
PCS Direct	26.6 Gbps	No GXT	28.3 Gbps	26.6 Gbps	No GXT

Note: Use the Native PHY IP Parameter Editor to determine the datarate limitations of your selected PCS configuration.

Related Information

Intel Stratix 10 Device Datasheet

1.3.3. GX and GXT Channel Placement Guidelines

Refer to *AN 778: Intel Stratix 10 Transceiver Usage* for detailed information on this section.

Related Information

AN 778: Intel Stratix 10 Transceiver Usage

1.3.4. GXT Channel Usage

Intel Stratix 10 L-Tile/H-Tile transceivers support GXT channels.

Table 7. Channel Types

There are a total of 24 channels available per tile. You can configure them as either GX channels or as a combination of GX and (up to 16) GXT channels provided that the total does not exceed 24. You can use GXT channels as a GX channel, but they are subject to all of the GX channel placement constraints.

Tile	Channel Type	Number of Channels per Tile	Channel Capability	
			Chip-to-Chip	Backplane
L-Tile	GX	Up to 24	17.4 Gbps	12.5 Gbps
	GXT [5]	Up to 8	26.6 Gbps	12.5 Gbps
H-Tile	GX	Up to 24	17.4 Gbps	
	GXT [5]	Up to 16	28.3 Gbps	28.3 Gbps

An ATX PLL can serve as the transmit PLL for up to six GXT channels.

Refer to *AN 778: Intel Stratix 10 Transceiver Usage* for detailed information about this section.

Related Information

- Intel Stratix 10 Device Datasheet
- AN 778: Intel Stratix 10 Transceiver Usage

1.3.5. PLL and Clock Networks

There are two different types of clock networks to distribute the high speed serial clock to the channels:

- Transceiver clock network that supports GX channels and allows a single TX PLL to drive up to 24 bonded channels in a tile.
- High Performance clock network that allows a single ATX PLL to drive up to 6 GXT channels in unbonded configurations.

Table 8. Channel Type Supported by Different Clock Networks

Clock Network	Clock Lines	Channel Type Support
Standard	x1, x6, x24	GX
High Performance	PLL Direct Connect	GXT

1.3.5.1. PLLs

1.3.5.1.1. Transceiver Phase-Locked Loops

Each transceiver channel in Intel Stratix 10 devices has direct access to three types of high performance PLLs:

- Advanced Transmit (ATX) PLL
- Fractional PLL (fPLL)
- Channel PLL / Clock Multiplier Unit (CMU) PLL.

[5] If you use GXT channel data rates, the V_{CCR_GXB} and V_{CCT_GXB} voltages must be set to 1.12 V.

These transceiver PLLs along with the Master or Local Clock Generation Blocks (CGB) drive the transceiver channels.

Related Information

PLLs on page 251
For more information about transceiver PLLs in Stratix 10 devices.

Advanced Transmit (ATX) PLL

The ATX PLL is the transceiver channel’s primary transmit PLL. It can operate over the full range of supported datarates required for high datarate applications. An ATX PLL supports both integer frequency synthesis and coarse resolution fractional frequency synthesis (when configured as a cascade source).

Fractional PLL (fPLL)

A fractional PLL (fPLL) is an alternate transmit PLL used for generating lower clock frequencies for lower datarate applications. fPLLs support both integer frequency synthesis and fine resolution fractional frequency synthesis. Unlike the ATX PLL, you can also use the fPLL to synthesize frequencies that can drive the core through the FPGA fabric clock networks.

Channel PLL (CMU/CDR PLL)

A channel PLL is located within each transceiver channel. The channel's primary function is clock and data recovery in the transceiver channel when you use the PLL in clock data recovery (CDR) mode. You can use the channel PLLs of channel 1 and 4 as transmit PLLs when configured in clock multiplier unit (CMU) mode. You cannot configure the channel PLLs of channel 0, 2, 3, and 5 in CMU mode; therefore, you cannot use them as transmit PLLs. You cannot use the receiver channel when you use it as a Channel PLL/CMU.

1.3.5.1.2. Clock Generation Block (CGB)

Intel Stratix 10 devices include the following types of clock generation blocks (CGBs):

- Master CGB
- Local CGB

Transceiver banks have two master CGBs. The master CGB divides and distributes bonded clocks to a bonded channel group. The master CGB also distributes non-bonded clocks to non-bonded channels across the x6/x24 clock network.

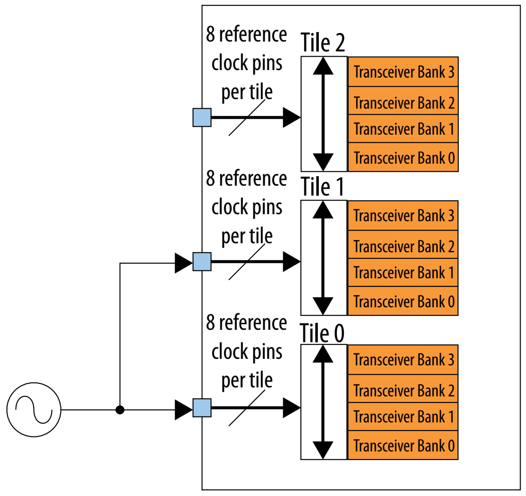
Each transceiver channel has a local CGB. The local CGB divides and distributes non-bonded clocks to the corresponding PCS and PMA blocks.

1.3.5.2. Input Reference Clock Sources

- Eight dedicated reference clocks available per transceiver tile
  - Two reference clocks per transceiver bank
  - You must route multiple copies of reference clocks on the PCB to span beyond a transceiver tile
- Reference clock network
  - Reference clock network does not span beyond the transceiver tile
  - There are two regulated reference clock networks for better performance per tile that any reference clock pin can access
- You can use unused receiver pins as additional reference clocks

Note: Unused receiver pins used as reference clocks can only be used within the same tile.

Figure 17. Reference Clock Network



For the best jitter performance, place the reference clock as close as possible to the transmit PLL. Use the reference clock in the same triplet of the bank as the transmit PLL.

1.3.5.3. Transceiver Clock Network

1.3.5.3.1. x1 Clock Lines

The ATX PLL, fPLL, or CMU PLL can access the x1 clock lines. The x1 clock lines allow the TX PLL to drive multiple transmit channels in the same bank in non-bonded mode.

For more information, refer to the *x1 Clock Lines* section.

Related Information

x1 Clock Lines on page 283

1.3.5.3.2. x6 Clock Lines

The ATX PLL or fPLL can access the x6 clock lines through the master CGB. The x6 clock lines allow the TX PLL to drive multiple bonded or non-bonded transmit channels in the same bank.

For more information, refer to the *x6 Clock Lines* section.

Related Information

x6 Clock Lines on page 284

1.3.5.3.3. x24 Clock Lines

Route the x6 clock lines onto x24 clock lines to allow a single ATX PLL or fPLL to drive multiple bonded or non-bonded transmit channels in multiple banks in an L-/H-Tile.

1.3.5.3.4. GXT Clock Network

The GXT Clock Network allows the ATX PLL to drive up to six GXT channels in non-bonded mode.

The top ATX PLL in a bank can drive:

- Channels 0, 1, 3, 4 in the bank
- Channels 0, 1 in the bank above in the same H-Tile

The bottom ATX PLL in a bank can drive:

- Channels 0, 1, 3, 4 in the bank
- Channels 3, 4 in the bank below in the same H-Tile

Related Information

GXT Clock Network on page 289

1.3.6. Ethernet Hard IP

1.3.6.1. 100G/50G Ethernet MAC Hard IP

The 100G/50G Ethernet MAC Hard IP block implements an Ethernet stack with MAC and PCS layers, as defined in the www.ieee802.org/3/.

Note: This Hard IP only apples to Intel Stratix 10 H-Tile devices.

- Supported Protocols
  - 100G MAC + PCS Ethernet x4 lanes
  - 50G MAC + PCS Ethernet x2 lanes
- Modes
  - MAC + PCS
  - PCS only
  - PCS66 (encoder/scrambler bypass)
  - Loopbacks
  - AN/LT with soft logic: dynamic switching
- Requires a soft Auto Negotiation / Link Training (AN/LT) logic implemented in the core fabric. Implement the AN/LT logic, or use a MAC IP.

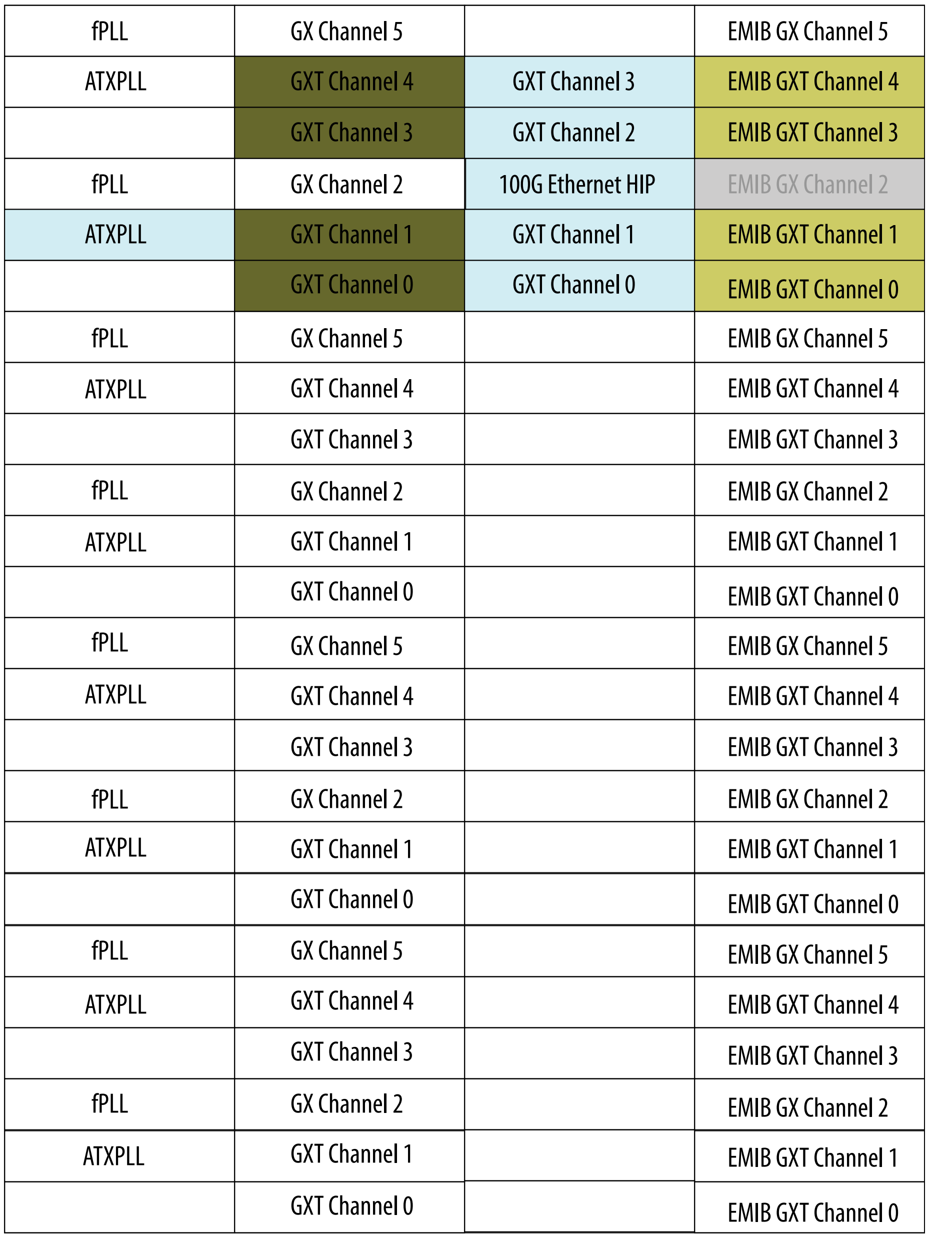
Note: Auto negotiation (AN) is an exchange in which link partners to determine the highest performance datarate that they both support. Link training (LT) is the process that defines how a receiver (RX) and a transmitter (TX) on a high-speed serial link communicate with each other to tune their PMA settings.

The protocol specifies how to request the link partner TX driver to adjust TX deemphasis, but the standard does not state how or when to adjust receiver equalization. The manufacturer determines how they adjust their receiver equalization. The algorithm for RX settings is different between tiles.

1.3.6.2. 100G Configuration

The Ethernet Hard IP uses 5 channels in the top transceiver bank of the tile. Channels 0, 1, 3 and 4 send or receive data at 25 Gbps. Channel 2 bonds the 4 transceiver channels and it cannot be used for other purposes.

Figure 18. 100G Configuration



fPLL	GX Channel 5		EMIB GX Channel 5
ATXPLL	GXT Channel 4	GXT Channel 3	EMIB GXT Channel 4
	GXT Channel 3	GXT Channel 2	EMIB GXT Channel 3
fPLL	GX Channel 2	100G Ethernet HIP	EMIB GX Channel 2
ATXPLL	GXT Channel 1	GXT Channel 1	EMIB GXT Channel 1
	GXT Channel 0	GXT Channel 0	EMIB GXT Channel 0
fPLL	GX Channel 5		EMIB GX Channel 5
ATXPLL	GXT Channel 4		EMIB GXT Channel 4
	GXT Channel 3		EMIB GXT Channel 3
fPLL	GX Channel 2		EMIB GX Channel 2
ATXPLL	GXT Channel 1		EMIB GXT Channel 1
	GXT Channel 0		EMIB GXT Channel 0
fPLL	GX Channel 5		EMIB GX Channel 5
ATXPLL	GXT Channel 4		EMIB GXT Channel 4
	GXT Channel 3		EMIB GXT Channel 3
fPLL	GX Channel 2		EMIB GX Channel 2
ATXPLL	GXT Channel 1		EMIB GXT Channel 1
	GXT Channel 0		EMIB GXT Channel 0
fPLL	GX Channel 5		EMIB GX Channel 5
ATXPLL	GXT Channel 4		EMIB GXT Channel 4
	GXT Channel 3		EMIB GXT Channel 3
fPLL	GX Channel 2		EMIB GX Channel 2
ATXPLL	GXT Channel 1		EMIB GXT Channel 1
	GXT Channel 0		EMIB GXT Channel 0

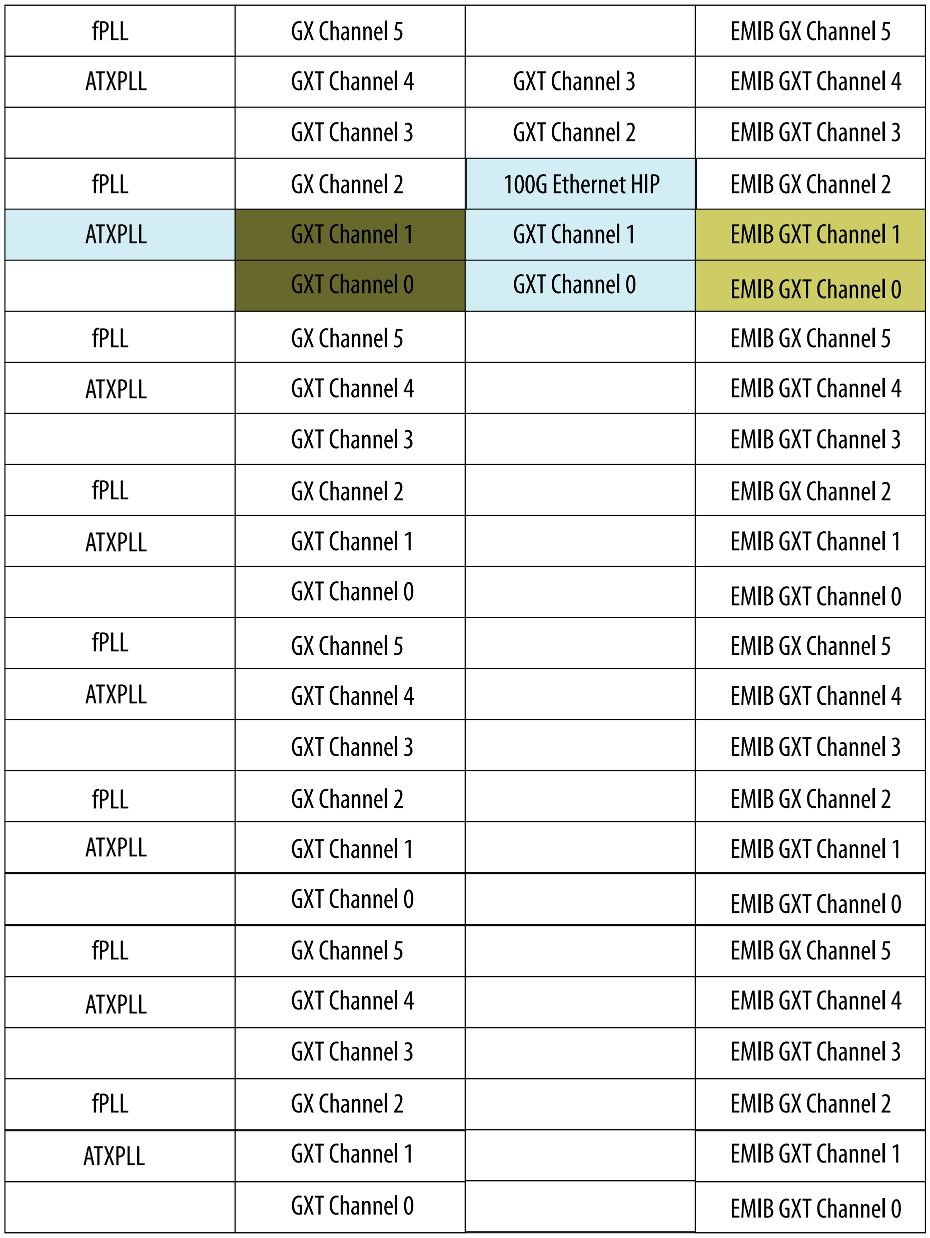
1.3.6.3. 50G Configuration

Channel 0 and 1 of the top transceiver bank implement the 50G configuration.

Note: Auto negotiation (AN) is an exchange in which link partners to determine the highest performance datarate that they both support. Link training (LT) is the exchange to arrive at PMA settings.

The protocol specifies how to request the link partner TX driver to adjust TX deemphasis, but the standard does not state how or when to adjust receiver equalization. The manufacturer determines how they adjust their receiver equalization. The algorithm for RX settings is different between tiles.

Figure 19. 50G Configuration



fPLL	GX Channel 5		EMIB GX Channel 5
ATXPLL	GXT Channel 4	GXT Channel 3	EMIB GXT Channel 4
	GXT Channel 3	GXT Channel 2	EMIB GXT Channel 3
fPLL	GX Channel 2	100G Ethernet HIP	EMIB GX Channel 2
ATXPLL	GXT Channel 1	GXT Channel 1	EMIB GXT Channel 1
	GXT Channel 0	GXT Channel 0	EMIB GXT Channel 0
fPLL	GX Channel 5		EMIB GX Channel 5
ATXPLL	GXT Channel 4		EMIB GXT Channel 4
	GXT Channel 3		EMIB GXT Channel 3
fPLL	GX Channel 2		EMIB GX Channel 2
ATXPLL	GXT Channel 1		EMIB GXT Channel 1
	GXT Channel 0		EMIB GXT Channel 0
fPLL	GX Channel 5		EMIB GX Channel 5
ATXPLL	GXT Channel 4		EMIB GXT Channel 4
	GXT Channel 3		EMIB GXT Channel 3
fPLL	GX Channel 2		EMIB GX Channel 2
ATXPLL	GXT Channel 1		EMIB GXT Channel 1
	GXT Channel 0		EMIB GXT Channel 0
fPLL	GX Channel 5		EMIB GX Channel 5
ATXPLL	GXT Channel 4		EMIB GXT Channel 4
	GXT Channel 3		EMIB GXT Channel 3
fPLL	GX Channel 2		EMIB GX Channel 2
ATXPLL	GXT Channel 1		EMIB GXT Channel 1
	GXT Channel 0		EMIB GXT Channel 0

You can use channels 2-5 in the top bank of the tile when the Ethernet hard IP is configured in 50G mode.

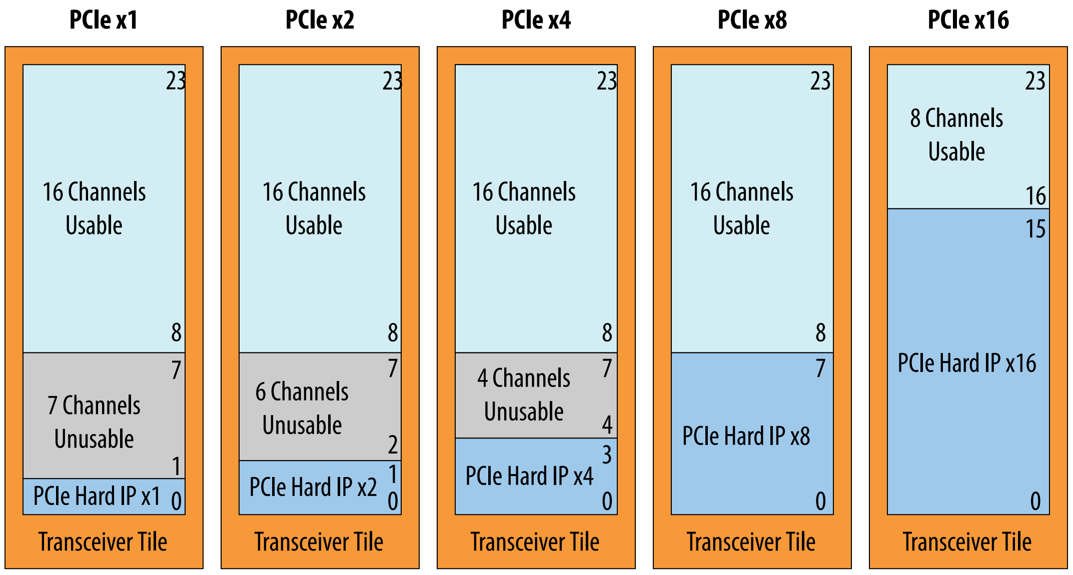
1.3.7. PCIe Gen1/Gen2/Gen3 Hard IP Block

The PCIe Hard IP is an IP block that provides multiple layers of the protocol stack for PCI Express. The Intel Stratix 10 Hard IP for PCIe is a complete PCIe solution that includes the Transaction, Data Link, and PHY/MAC layers. The Hard IP solution contains dedicated hard logic that connects to the transceiver PHY interface. Each transceiver tile contains a PCIe Hard IP block supporting PCIe Gen1, Gen2, or Gen3 protocols with x1, x2, x4, x8, and x16 configurations. x1, x2, and x4 configurations result in unusable channels. The Hard IP resides at the bottom of the tile, and is 16 channels high. Additionally, the block includes extensible VF (Virtual Functions) interface to enable implementation of up to 2K VFs via the SRIOV-w (Single-Root I/O Virtualization) bridge. The following table and figure show the possible PCIe Hard IP channel configurations, the number of unusable channels, and the number of channels available for other protocols.

Table 9. PCIe Hard IP Channel Configurations Per Transceiver Tile

PCIe Hard IP Configuration	Number of Unusable Channels	Number of Channels Available for Other Protocols
PCIe x1	7	16
PCIe x2	6	16
PCIe x4	4	16
PCIe x8	0	16
PCIe x16	0	8

Figure 20. PCIe Hard IP Channel Configurations Per Transceiver Tile



The table below maps all transceiver channels to PCIe Hard IP channels in available tiles.

Table 10. PCIe Hard IP Channel Mapping Across all Tiles

Tile Channel Sequence	PCIe Hard IP Channel	Index within I/O Bank	Bottom Left Tile Bank Number	Top Left Tile Bank Number	Bottom Right Tile Bank Number	Top Right Tile Bank Number
23	—	5	1F	1N	4F	4N
22	—	4	1F	1N	4F	4N
21	—	3	1F	1N	4F	4N
20	—	2	1F	1N	4F	4N
19	—	1	1F	1N	4F	4N
18	—	0	1F	1N	4F	4N
17	—	5	1E	1M	4E	4M
16	—	4	1E	1M	4E	4M
15	15	3	1E	1M	4E	4M
14	14	2	1E	1M	4E	4M
13	13	1	1E	1M	4E	4M
12	12	0	1E	1M	4E	4M
11	11	5	1D	1L	4D	4L
10	10	4	1D	1L	4D	4L
9	9	3	1D	1L	4D	4L
8	8	2	1D	1L	4D	4L
7	7	1	1D	1L	4D	4L
6	6	0	1D	1L	4D	4L
5	5	5	1C	1K	4C	4K
4	4	4	1C	1K	4C	4K
3	3	3	1C	1K	4C	4K
2	2	2	1C	1K	4C	4K
1	1	1	1C	1K	4C	4K
0	0	0	1C	1K	4C	4K

The PCIe Hard IP block includes extensible VF (Virtual Functions) interface to enable the implementation of up to 2K VFs via the SRIOV-2 (Single-Root I/O Virtualization) bridge.

In network virtualization, single root input/output virtualization or SR-IOV is a network interface that allows the isolation of the PCI Express resources for manageability and performance reasons. A single physical PCI Express is shared on a virtual environment using the SR-IOV specification. The SR-IOV specification offers different virtual functions to different virtual components, such as a network adapter, on a physical server machine.

Related Information

http://www.design-reuse.com/articles/32998/single-root-i-o-virtualization.html

1.4. Overview Revision History

Document Version	Changes
2020.03.03	Made the following changes: • Updated the Intel Stratix 10 TX devices in the "Intel Stratix 10 TX Device with 1 E-Tile and 1 H-Tile (48 Transceiver Channels)" figure and the "H- and E-Tile Counts in Intel Stratix 10 TX Devices (HF35, NF43, SF50, UF50, YF55)" table. • For GX Standard PCS data rates in *GX Channel*, added 12 Gbps and the note, "The 12 Gbps data rate at the receiver is only supported when the **RX word aligner mode** parameter is set to **Manual**.
2019.03.22	Made the following change: • Changed the data rate for E-tile Non-Return to Zero (NRZ) to 28.9 Gbps. • Changed 60 GXE channels/device for PAM-4 to 57.8 Gbps. • Updated plan of record devices. • Updated device configuration drawings.
2018.07.06	Made the following changes: • Changed the GXT data rate limit for L-Tile to 26.6 Gbps in the "Channel Types" table. • Changed the data rate limit for -2 speed grades on both L-Tile and H-Tile to 26.6 Gbps in the "PCS Types Supported by GXT Type Transceiver Channels" table. • Clarified the number of reference clocks pins in the "Reference Clock Network" figure. • Changed the standard PCS data rates for L-Tile and H-Tile devices in the "PCS Types Supported by GX Transceiver Channels" table. • Changed the backplane data rate for L-Tile GX channels in the "Channel Types" table.
2018.03.16	Made the following changes: • Added the operational modes description for channels in the "Transceiver Bank Architecture" section. • Added PCS Direct to the "GX Transceiver Channel in TX/RX Duplex Mode" figure. • Added a cross-reference to the "General and Datapath Parameters" table in the "GX Channel" section. • Added PCS Direct to the "PCS Types Supported by GX Type Transceiver Channels" table. • Changed the description in the "GXT Channel" section. • Added PCS Direct to the "GXT Transceiver Channel in TX/RX Duplex Mode" figure. • Updated ATX PLL description stating "An ATX PLL supports both integer frequency synthesis and coarse resolution fractional frequency synthesis (when configured as a cascade source)". • Removed the NF48 package from the "L-Tile/H-Tile Counts in Intel Stratix 10 GX/SX Devices (HF35, NF43, UF50, HF55)" table.
2017.08.11	Made the following changes: • Added the "Transceiver Tile Variants—Comparison of Transceiver Capabilities" table. • Removed the "H-Tile Transceivers" section. • Added description to the "L-Tile/H-Tile Layout in Stratix 10 Device Variants" section. • Added the "Stratix 10 Tile Layout" figure. • Changed the package and tile counts in the "H- and E-Tile Counts in Intel Stratix 10 MX Devices (NF43, UF53, UF55)" table. • Added separate datarate support for L-Tile and H-Tile in the "PCS Types Supported by GX Type Transceiver Channels" table.
2017.06.06	Made the following changes: • Removed CEI 56G support from the "Stratix 10 Transceiver Protocols, Features, and IP Core Support" table. • Added tile names based on the thermal models to the figures in the "Stratix 10 GX/SX H-Tile Configurations" section. • Added tile names based on the thermal models to the figures in the "Stratix 10 TX H-Tile and E-Tile Configurations" section. • Added tile names based on the thermal models to the figures in the "Stratix 10 MX H-Tile and E-Tile Configurations" section.

continued...

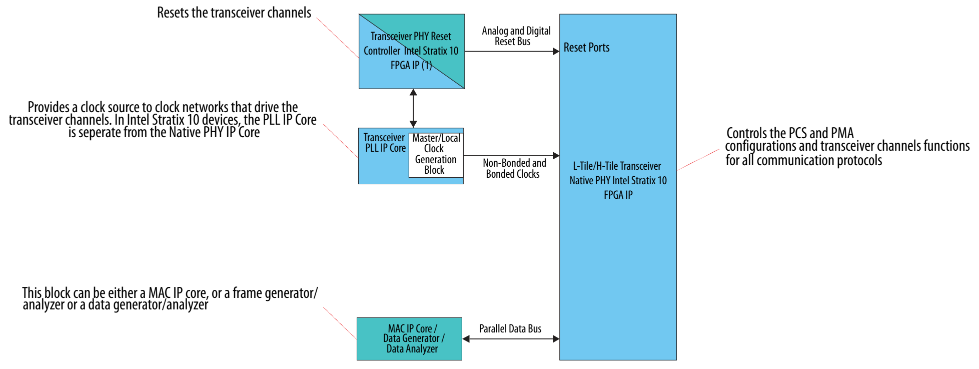
Document Version	Changes
	• Changed the number of GXT channels that the ATX PLL can support as a transmit PLL in the "GXT Channel Usage" section. • Changed the number of GXT channels an ATX PLL can support in the "GXT Channel Usage" section. • Removed a note in the "Input Reference Clock Sources" section.
2017.03.08	Made the following changes: • Changed all the notes in the "GXT Channel Usage" section. • Changed all the notes in the "PLL Direct Connect Clock Network" section.
2017.02.17	Made the following changes: • Completely updated the "GXT Channel Usage" section.
2016.12.21	Initial release.

2. Implementing the Transceiver PHY Layer in L-Tile/H-Tile

2.1. Transceiver Design IP Blocks

The following figure shows all the design blocks involved in designing and using Intel Stratix 10 transceivers.

Figure 21. Intel Stratix 10 Transceiver Design Fundamental Building Blocks


Resets the transceiver channels
Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP (1)
Analog and Digital Reset Bus
Reset Ports
Provides a clock source to clock networks that drive the transceiver channels. In Intel Stratix 10 devices, the PLL IP Core is seperate from the Native PHY IP Core
Transceiver PLL IP Core
Master/Local Clock Generation Block
Non-Bonded and Bonded Clocks
L-Tile/H-Tile Transceiver Native PHY Intel Stratix 10 FPGA IP
Controls the PCS and PMA configurations and transceiver channels functions for all communication protocols
This block can be either a MAC IP core, or a frame generator/ analyzer or a data generator/analyzer
MAC IP Core / Data Generator / Data Analyzer
Parallel Data Bus

Note:
(1) You can either design your own reset controller or use the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP core.

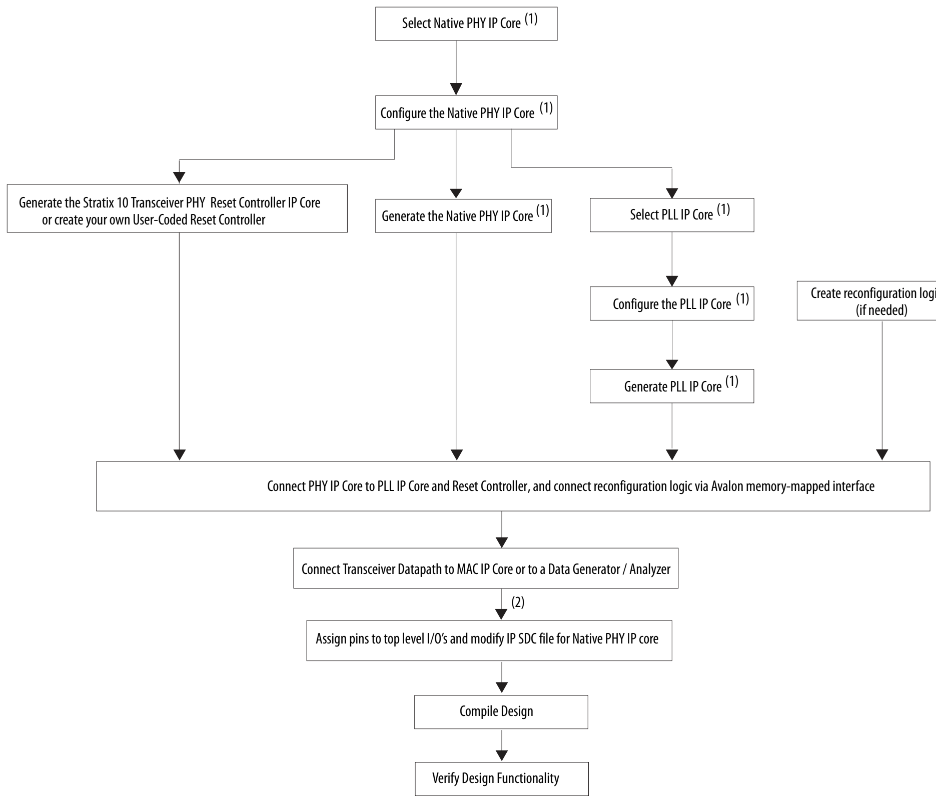
Legend:
🟦 Intel generated IP block
🟩 User created IP block

Related Information

Resetting Transceiver Channels on page 318

2.2. Transceiver Design Flow

Figure 22. Transceiver Design Flow


Select Native PHY IP Core (1)
Configure the Native PHY IP Core (1)
Generate the Stratix 10 Transceiver PHY Reset Controller IP Core or create your own User-Coded Reset Controller
Generate the Native PHY IP Core (1)
Select PLL IP Core (1)
Configure the PLL IP Core (1)
Create reconfiguration logic (if needed)
Generate PLL IP Core (1)
Connect PHY IP Core to PLL IP Core and Reset Controller, and connect reconfiguration logic via Avalon memory-mapped interface
Connect Transceiver Datapath to MAC IP Core or to a Data Generator / Analyzer
(2)
Assign pins to top level I/O's and modify IP SDC file for Native PHY IP core
Compile Design
Verify Design Functionality

Note:

(1) For more information refer to the "Introduction to Intel FPGA IP Cores" chapter in the "Quartus Prime Standard Edition Handbook Volume 1: Design and Synthesis"

(2) Select analog parameter settings. Implementation information will be available in the future release of this user guide.

Related Information

Introduction to Intel FPGA IP Cores

2.2.1. Select the PLL IP Core

Intel Stratix 10 transceivers have the following three types of PLL IP cores:

- Advanced Transmit (ATX) PLL IP core.
- Fractional PLL (fPLL) IP core.
- Channel PLL / Clock Multiplier Unit (CMU) PLL IP core.

Select the appropriate PLL IP for your design. For additional details, refer to the *PLLs and Clock Networks* chapter.

Refer to *Introduction to Intel FPGA IP Cores* in the Intel Quartus® Prime handbook for details on instantiating, generating and modifying IP cores.

Related Information

- PLLs and Clock Networks on page 249

- Introduction to Intel FPGA IP Cores

2.2.2. Reset Controller

There are two methods to reset the transceivers in Intel Stratix 10 devices:

- Use the Intel Stratix 10 Transceiver PHY Reset Controller IP Core.
- Create your own reset controller that follows the recommended reset sequence.

Related Information

Resetting Transceiver Channels on page 318

2.2.3. Create Reconfiguration Logic

Dynamic reconfiguration is the ability to dynamically modify the transceiver channels and PLL settings during device operation. To support dynamic reconfiguration, your design must include an Avalon® memory-mapped interface master that can access the dynamic reconfiguration registers using the Avalon memory-mapped interface.

The Avalon memory-mapped interface enables PLL and channel reconfiguration. You can dynamically adjust the PMA parameters, such as differential output voltage swing, and pre-emphasis settings. This adjustment can be done by writing to the Avalon memory-mapped interface reconfiguration registers through the user-generated Avalon memory-mapped interface master.

For detailed information on dynamic reconfiguration, refer to *Reconfiguration Interface and Dynamic Reconfiguration* chapter.

Related Information

Reconfiguration Interface and Dynamic Reconfiguration on page 394

2.2.4. Connect the Native PHY IP Core to the PLL IP Core and Reset Controller

Connect the PHY IP, PLL IP core, and the reset controller. Write the top level module to connect all the IP blocks.

All of the I/O ports for each IP, can be seen in the <*phy instance name*>.v file or <*phy instance name*>.vhd, and in the <*phy_instance_name*>_bb.v file.

For more information about description of the ports, refer to the ports tables in the *PLLs*, *Using the Transceiver Native PHY IP Core*, and *Resetting Transceiver Channels* chapters.

Related Information

Resetting Transceiver Channels on page 318

2.2.5. Connect Datapath

Connect the transceiver PHY layer design to the Media Access Controller (MAC) IP core or to a data generator/analyzer or a frame generator/analyzer. Assign pins to all I/O's using the **Assignment Editor** or **Pin Planner**, or updating the Intel Quartus Prime Settings File (**.qsf**).

1. Assign FPGA pins to all the transceiver and reference clock I/O pins. For more details, refer to the *Intel Stratix 10 Device Family Pin Connection Guidelines*.
2. All of the pin assignments set using the **Pin Planner** and the **Assignment Editor** are saved in the <top_level_project_name>.qsf file. You can also directly modify the Intel Quartus Prime Settings File (.qsf).

Related Information

- Intel Quartus Prime Pro Edition User Guide: Getting Started
  For more information about the Assignment Editor and Pin Planner
- Intel Stratix 10 Device Family Pin Connection Guidelines

2.2.6. Modify Native PHY IP Core SDC

IP SDC is a new feature of the Native PHY IP core.

IP SDC is produced for any clock that reaches the FPGA fabric. In transceiver applications where the `tx_clkouts` and `rx_clkouts` (plus some more) are routed to the FPGA fabric, these clocks have SDC constraints on them in the Native PHY IP core.

2.2.7. Compile the Design

To compile the transceiver design, add the <*phy_instancename*>.ip files for all the IP blocks generated using the IP Catalog to the Intel Quartus Prime project library.

Related Information

Intel Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design
For more information about compilation details.

2.2.8. Verify Design Functionality

Simulate your design to verify the functionality of your design. For more details, refer to the *Intel Quartus Prime Pro Edition User Guide: Debug Tools*.

Related Information

- Simulating the Native PHY IP Core on page 236
- System Debugging Tools Overview section of the *Intel Quartus Prime Pro Edition User Guide: Debug Tools*
- Debugging Transceiver Links section of the *Intel Quartus Prime Pro Edition User Guide: Debug Tools*

2.3. Configuring the Native PHY IP Core

This section describes the use of the Intel-provided Transceiver Native PHY IP core. This Native PHY IP core is the primary design entry tool and provides direct access to Intel Stratix 10 transceiver PHY features.

Use the Native PHY IP core to configure the transceiver PHY for your protocol implementation. To instantiate the IP, select the Intel Stratix 10 device family, click **Tools ➤ IP Catalog** to select your IP core variation. Use the **Parameter Editor** to specify the IP parameters and configure the PHY IP for your protocol implementation. To quickly configure the PHY IP, select a preset that matches your protocol configuration as a starting point. Presets are PHY IP configuration settings for various protocols that are stored in the IP **Parameter Editor**. Presets are explained in detail in the *Presets* section below.

You can also configure the PHY IP by selecting an appropriate **Transceiver Configuration Rule**. The transceiver configuration rules check the valid combinations of the PCS and PMA blocks in the transceiver PHY layer, and report errors or warnings for any invalid settings.

Use the Native PHY IP core to instantiate one of the following PCS options:

- Standard PCS
- Enhanced PCS
- PCIe Gen3 PCS
- PCS Direct

Based on the Transceiver Configuration Rule that you select, the PHY IP core selects the appropriate PCS. Refer to the *How to Place Channels for PIPE Configuration* section or the PCIe solutions guides on restrictions on placement of transceiver channels next to active banks with PCI Express interfaces that are Gen3 capable.

After you configure the PHY IP core in the **Parameter Editor**, click **Generate HDL** to generate the IP instance. The top level file generated with the IP instance includes all the available ports for your configuration. Use these ports to connect the PHY IP core to the PLL IP core, the reset controller IP core, and to other IP cores in your design.

Figure 23. Native PHY IP Core Ports and Functional Blocks

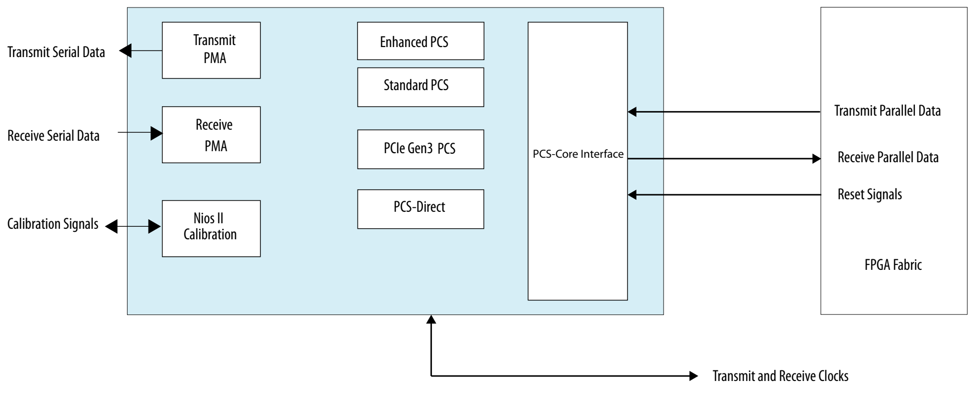
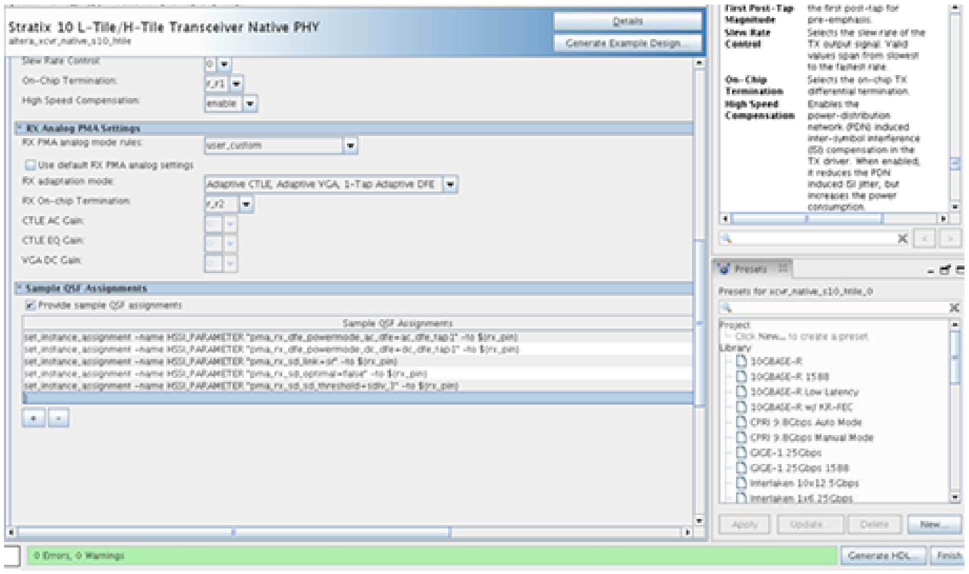


Figure 24. Native PHY IP Core Parameter Editor



Stratix 10 L-Tile/H-Tile Transceiver Native PHY
altera_xcvr_native_s10_htile

Slew Rate Control: 0
On-Chip Termination: r_r1
High Speed Compensation: enable

RX Analog PMA Settings
RX PMA analog mode rules: user_custom
☐ Use default RX PMA analog settings
RX adaptation mode: Adaptive CTLE, Adaptive VGA, 1-Tap Adaptive DFE
RX On-chip Termination: r_r2
CTLE AC Gain: 0
CTLE EQ Gain: 0
VGA DC Gain: 0

Sample QSF Assignments
☑ Provide sample QSF assignments

Sample QSF Assignments
set_instance_assignment -name HSSI_PARAMETER "pma_rx_dfe_powermode_ac_dfe=ac_dfe_tap1" -to $(rx_pin)
set_instance_assignment -name HSSI_PARAMETER "pma_rx_dfe_powermode_dc_dfe=dc_dfe_tap1" -to $(rx_pin)
set_instance_assignment -name HSSI_PARAMETER "pma_rx_sd_link=sr" -to $(rx_pin)
set_instance_assignment -name HSSI_PARAMETER "pma_rx_sd_optimal=false" -to $(rx_pin)
set_instance_assignment -name HSSI_PARAMETER "pma_rx_sd_sd_threshold=sdlv_3" -to $(rx_pin)

0 Errors, 0 Warnings

First Post-Tap Magnitude	the first post-tap for pre-emphasis.
Slew Rate Control	Selects the slew rate of the TX output signal. Valid values span from slowest to the fastest rate.
On-Chip Termination	Selects the on-chip TX differential termination.
High Speed Compensation	Enables the power-distribution network (PDN) induced inter-symbol interference (ISI) compensation in the TX driver. When enabled, it reduces the PDN induced SI jitter, but increases the power consumption.

Presets for xcvr_native_s10_htile_0

Project
Click New... to create a preset.
Library
10GBASE-R
10GBASE-R 1588
10GBASE-R Low Latency
10GBASE-R w/ KR-FEC
CPRI 9.8Gbps Auto Mode
CPRI 9.8Gbps Manual Mode
GIGE-1.25Gbps
GIGE-1.25Gbps 1588
Interlaken 10x12.5Gbps
Interlaken 1x6.25Gbps

Note: Although the Intel Quartus Prime Pro Edition software provides legality checks, the supported FPGA fabric to PCS interface widths and the supported datarates are pending characterization.

Related Information

- How to Place Channels for PIPE Configurations on page 207
- Intel Stratix 10 Avalon Memory-Mapped Interface Hard IP for PCIe Design Example User Guide
- Intel Stratix 10 Avalon Memory-Mapped Interface Interface for PCI Express Solutions User Guide
- Intel Stratix 10 Avalon Streaming Interface Hard IP for PCIe Design Example User Guide
- Intel Stratix 10 Avalon Streaming Interface and Single Root I/O Virtualization (SR-IOV) Interface for PCI Express Solutions User Guide

2.3.1. Protocol Presets

You can select preset settings for the Native PHY IP core defined for each protocol. Use presets as a starting point to specify parameters for your specific protocol or application.

To apply a preset to the Native PHY IP core, double-click the preset name. When you apply a preset, all relevant options and parameters are set in the current instance of the Native PHY IP core. For example, selecting the **Interlaken** preset enables all parameters and ports that the Interlaken protocol requires.

Selecting a preset does not prevent you from changing any parameter to meet the requirements of your design. Any changes that you make are validated by the design rules for the transceiver configuration rules you specified, not the selected preset.

Note: Selecting a preset clears any prior selections you have made so far.

2.3.2. GXT Channels

You can instantiate up to 16 GXT channels per H-tile and up to eight GXT channels per L-Tile using a Intel Stratix 10 L-/H-Tile Native PHY IP instance.

Set the following parameters:

- Set the **VCCR_GXB and VCCT_GXB supply voltage** for the transceiver parameter to **1_1V**.
- Set the **TX channel bonding mode** parameter to **Not Bonded**.
- Set the **datarate** parameter between `17400` and `25800` (L-Tile, and `28300` (H-Tile).
- Set the **number of channels** between `1` and `16`.

Because each ATX PLL's tx_serial_clk_gt can connect up to 2 GXT channels, you must instantiate one to eight ATX PLLs. Be aware of the GXT channel location and connect the appropriate ATX PLL's `tx_serial_clk_gt` port to the Native PHY IP Core's `tx_serial_clk` port.

Refer to *Using the ATX PLL for GXT Channels* section for more details.

Refer to *AN 778: Intel Stratix 10 Transceiver Usage* for more information about transceiver channel placement guidelines for both L- and H-Tiles.

Related Information

- Using the ATX PLL for GXT Channels on page 255
- *AN 778: Intel Stratix 10 Transceiver Usage*

2.3.3. General and Datapath Parameters

You can customize your instance of the Native PHY IP core by specifying parameter values. In the **Parameter Editor**, the parameters are organized in the following sections for each functional block and feature:

- General, Common PMA Options, and Datapath Options
- TX PMA
- RX PMA
- Standard PCS
- Enhanced PCS
- PCS Direct Datapath
- PCS-Core Interface
- Analog PMA Settings (Optional)
- Dynamic Reconfiguration
- Generation Options

Table 11. General, Common PMA Options, and Datapath Options

Parameter	Value	Description
Message level for rule violations	**error** **warning**	Specifies the messaging level to use for parameter rule violations. Selecting **error** causes all rule violations to prevent IP generation. Selecting **warning** displays all rule violations as warnings in the message window and allows IP generation despite the violations. (6)
Use fast reset for simulation	**On/Off**	When enabled, the IP disables reset staggering in simulation. The reset behavior in simulation is different from the reset behavior in the hardware.
VCCR_GXB and VCCT_GXB supply voltage for the Transceiver	**1_0V, 1_1V** (7)	Selects the V_{CCR_GXB} and V_{CCT_GXB} supply voltage for the transceiver.
Transceiver Link Type	**sr, lr**	Selects the type of transceiver link. SR-Short Reach (Chip-to-chip communication), LR-Long Reach (Backplane communication).
Transceiver channel type	**GX, GXT**	Specifies the transceiver channel variant.
Transceiver configuration rules	User Selection	Specifies the valid configuration rules for the transceiver. This parameter specifies the configuration rule against which the **Parameter Editor** checks your PMA and PCS parameter settings for specific protocols. Depending on the transceiver configuration rule selected, the **Parameter Editor** validates the parameters and options selected by you and generates error messages or warnings for all invalid settings. To determine the transceiver configuration rule to be selected for your protocol, refer to *Transceiver Protocols using the Intel Stratix 10 H-Tile Transceiver Native PHY IP Core* table for more details about each transceiver configuration rule. This parameter is used for rule checking and is not a preset. You need to set all parameters for your protocol implementation. *Note:* For a full description of the Transceiver Configuration Rule Parameter Settings, refer to Table 12 on page 38 in this section.
PMA configuration rules	**Basic** **SATA/SAS** **GPON**	Specifies the configuration rule for the PMA. Select Basic for all other protocol modes except for SATA, and GPON. SATA (Serial ATA) can be used only if the **Transceiver configuration rule** is set to **Basic/Custom (Standard PCS)**. Select **GPON** only if the **Transceiver configuration rule** is set to **Basic (Enhanced PCS)**.
Transceiver mode	**TX/RX Duplex** **TX Simplex** **RX Simplex**	Specifies the operational mode of the transceiver. • **TX/RX Duplex** : Specifies a single channel that supports both transmission and reception. • **TX Simplex** : Specifies a single channel that supports only transmission. • **RX Simplex** : Specifies a single channel that supports only reception. The default is **TX/RX Duplex**.

continued...

(6) Although you can generate the PHY with warnings, you may not be able to compile the PHY in Intel Quartus Prime Pro Edition.

(7) Refer to the *Intel Stratix 10Device Datasheet* for details about the minimum, typical, and maximum supply voltage specifications.

Parameter	Value	Description
Number of data channels	**1 – 24**	Specifies the number of transceiver channels to be implemented. The default value is **1**.
Data rate	*< valid transceiver datarate >*	Specifies the datarate in megabits per second (Mbps).
Enable datapath and interface reconfiguration	**On/Off**	When you turn this option on, you can preconfigure and dynamically switch between the Standard PCS, Enhanced PCS, and PCS direct datapaths. You cannot enable the simplified data interface option if you intend on using this feature to support channel reconfiguration. The default value is **Off**.
Enable simplified data interface	**On/Off**	By default, all 80-bits are ports for the `tx_parallel_data` and `rx_parallel_data` buses are exposed. You must understand the mapping of data and control signals within the interface. Refer to the *Enhanced PCS TX and RX Control Ports* section for details about mapping of data and control signals. When you turn on this option, the Native PHY IP core presents a simplified data and control interface between the FPGA fabric and transceiver. Only the sub-set of the 80-bits that are active for a particular FPGA fabric width are ports. You cannot enable simplified data interface when double rate transfer mode is enabled. The default value is **Off**.
Enable double rate transfer mode	**On/Off**	When selected, the Native PHY IP core splits the PCS parallel data into two words and each word is transferred to and from the transceiver interface at twice the parallel clock frequency and half the normal width of the fabric core interface. You cannot enable simplified data interface when double rate transfer mode is enabled.
Enable PIPE EIOS RX Protection	**On/Off**	This feature is available for Gen 2 and Gen 3 PCIe PIPE interface selectable in **Transceiver configuration rules**. When selected, the Native PHY IP core improves the fault-tolerance and compatibility. You need to enable **Enable dynamic reconfiguration** and connect clock and reset. When selected, Intel recommends using these commands to enable physical simulation models: • In ModelSim: vlog –sv **+define +USE_PMA_ORORA_MODELS** • In VCS: vcs –lcs **+define+USE_PMA_ORORA_MODELS**

Table 12. Transceiver Configuration Rule Parameters

Transceiver Configuration Setting	Description
Basic/Custom (Standard PCS)	Enforces a standard set of rules within the Standard PCS. Select these rules to implement custom protocols requiring blocks within the Standard PCS or protocols not covered by the other configuration rules.
Basic/Custom w /Rate Match (Standard PCS)	Enforces a standard set of rules including rules for the Rate Match FIFO within the Standard PCS. Select these rules to implement custom protocols requiring blocks within the Standard PCS or protocols not covered by the other configuration rules.
CPRI (Auto)	Enforces rules required by the CPRI protocol. The receiver word aligner mode is set to **Auto**. In **Auto** mode, the word aligner is set to deterministic latency.

continued...

Transceiver Configuration Setting	Description
CPRI (Manual)	Enforces rules required by the CPRI protocol. The receiver word aligner mode is set to **Manual**. In **Manual** mode, logic in the FPGA fabric controls the word aligner.
GbE	Enforces rules that the 1 Gbps Ethernet (1 GbE) protocol requires.
GbE 1588	Enforces rules for the 1 GbE protocol with support for Precision time protocol (PTP) as defined in the *IEEE 1588 Standard*.
Gen1 PIPE	Enforces rules for a Gen1 PCIe PIPE interface that you can connect to a soft MAC and Data Link Layer.
Gen2 PIPE	Enforces rules for a Gen2 PCIe PIPE interface that you can connect to a soft MAC and Data Link Layer.
Gen3 PIPE	Enforces rules for a Gen3 PCIe PIPE interface that you can connect to a soft MAC and Data Link Layer.
Basic (Enhanced PCS)	Enforces a standard set of rules within the Enhanced PCS. Select these rules to implement protocols requiring blocks within the Enhanced PCS or protocols not covered by the other configuration rules.
Interlaken	Enforces rules required by the Interlaken protocol.
10GBASE-R	Enforces rules required by the 10GBASE-R protocol.
10GBASE-R 1588	Enforces rules required by the 10GBASE-R protocol with 1588 enabled. This setting can also be used to implement CPRI protocol version 6.0 and later.
10GBASE-R w/KR FEC	Enforces rules required by the 10GBASE-R protocol with KR FEC block enabled.
40GBASE-R w/KR FEC	Enforces rules required by the 40GBASE-R protocol with the KR FEC block enabled.
Basic w/KR FEC	Enforces a standard set of rules required by the Enhanced PCS when you enable the KR FEC block. Select this rule to implement custom protocols requiring blocks within the Enhanced PCS or protocols not covered by the other configuration rules.
PCS Direct	Enforces rules required by the PCS Direct mode. In this configuration the data flows through the PCS channel, but all the internal PCS blocks are bypassed. If required, the PCS functionality can be implemented in the FPGA fabric.

Related Information

- Enhanced PCS TX and RX Control Ports on page 76
- Intel Stratix 10 Device Datasheet

2.3.4. PMA Parameters

You can specify values for the following types of PMA parameters:

TX PMA:

- TX Bonding Options
- TX PLL Options
- TX PMA Optional Ports

RX PMA:

- RX CDR Options
- RX PMA Optional Ports

Table 13. TX Bonding Options

Parameter	Value	Description
TX channel bonding mode	**Not bonded** **PMA only bonding** **PMA and PCS bonding**	Selects the bonding mode to be used for the channels specified. Bonded channels use a single TX PLL to generate a clock that drives multiple channels, reducing channel-to-channel skew. The following options are available: **Not bonded**: In a non-bonded configuration, only the high speed serial clock is expected to be connected from the TX PLL to the Native PHY IP core. The low speed parallel clock is generated by the local clock generation block (CGB) present in the transceiver channel. For non-bonded configurations, because the channels are not related to each other and the feedback path is local to the PLL, the skew between channels cannot be calculated. **PMA only bonding**: In PMA bonding, the high speed serial clock is routed from the transmitter PLL to the master CGB. The master CGB generates the high speed and low parallel clocks and the local CGB for each channel is bypassed. Refer to the *Channel Bonding* section for more details. **PMA and PCS bonding** : In a PMA and PCS bonded configuration, the local CGB in each channel is bypassed and the parallel clocks generated by the master CGB are used to clock the network. The master CGB generates both the high and low speed clocks. The master channel generates the PCS control signals and distributes to other channels through a control plane block. The default value is **Not bonded**. Refer to *Channel Bonding* section in *PLLs and Clock Networks* chapter for more details.
PCS TX channel bonding master	**Auto, 0 to <number of channels> -1**	This feature is only available if PMA and PCS bonding mode has been enabled. Specifies the master PCS channel for PCS bonded configurations. Each Native PHY IP core instance configured with bonding must specify a bonding master. If you select **Auto**, the Native PHY IP core automatically selects a recommended channel. The default value is **Auto**. Refer to the *PLLs and Clock Networks* chapter for more information about the TX channel bonding master.
Actual PCS TX channel bonding master	**0 to <number of channels> -1**	This parameter is automatically populated based on your selection for the **PCS TX channel bonding master** parameter. Indicates the selected master PCS channel for PCS bonded configurations.
PCS reset sequence	**Independent** **Simultaneous**	Selects whether PCS `tx/rx_digitalreset` is asserted and deasserted independently or simultaneously. Selecting independent staggers the assertion and deassertion of the PCS reset of each transceiver channel one after the other. The independent setting is recommended for PCS non-bonded configurations. Selecting simultaneous, simultaneously asserts and deasserts all the PCS resets of each transceiver channel. Simultaneous setting is required for the following operations: • PCS bonding configuration • When multiple channels need to be released from reset at the same time. Example: Interlaken is non-bonded but requires the channels to be out of reset at the same time.

Table 14. TX PLL Options

TX PLL Options are only available if you have selected non bonded for TX channel bonding mode. The note on the bottom of the **TX PLL Options** tab in the GUI indicates the required output clock frequency of the external TX PLL IP instance.

Parameter	Value	Description
TX local clock division factor	**1, 2, 4, 8**	Specifies the value of the divider available in the transceiver channels to divide the TX PLL output clock to generate the correct frequencies for the parallel and serial clocks.
Number of TX PLL clock inputs per channel	**1, 2, 3 , 4**	Specifies the number of TX PLL clock inputs per channel. Use this parameter when you plan to dynamically switch between TX PLL clock sources. Up to four input sources are possible.
Initial TX PLL clock input selection	**0 to <number of TX PLL clock inputs> -1**	Specifies the initially selected TX PLL clock input. This parameter is necessary when you plan to switch between multiple TX PLL clock inputs.

Table 15. TX PMA Optional Ports

Parameter	Value	Description
Enable tx_pma_iqtxrx_clkout port	**On/Off**	Enables the optional `tx_pma_iqtxrx_clkout` output clock. This clock can be used to cascade the TX PMA output clock to the input of a PLL.
Enable tx_pma_elecidle port	**On/Off**	Enables the `tx_pma_elecidle` port. When you assert this port, the transmitter is forced into an electrical idle condition. This port has no effect when the transceiver is configured for PCI Express.

Table 16. RX CDR Options

Parameter	Value	Description
Number of CDR reference clocks	**1 - 5**	Specifies the number of CDR reference clocks. Up to 5 sources are possible. The default value is 1.
Selected CDR reference clock	**0 to <number of CDR reference clocks> -1**	Specifies the initial CDR reference clock. This parameter determines the available CDR references used. The default value is 0.
Selected CDR reference clock frequency	*< datarate dependent >*	Specifies the CDR reference clock frequency. This value depends on the datarate specified. You should choose a lane datarate that results in a standard board oscillator reference clock frequency to drive the CDR reference clock and meet jitter requirements. Choosing a lane datarate that deviates from standard reference clock frequencies may result in custom board oscillator clock frequencies, which may be prohibitively expensive or unavailable.
PPM detector threshold	**100** **300** **500** **1000**	Specifies the PPM threshold for the CDR. If the PPM between the incoming serial data and the CDR reference clock exceeds this threshold value, the CDR declares lose of lock. The default value is 1000.

Table 17. RX PMA Optional Ports

Parameters	Value	Description
Enable rx_pma_iqtxrx_clkout port	**On/Off**	Enables the optional `rx_pma_iqtxrx_clkout` output clock. This clock can be used to cascade the RX PMA output clock to the input of a PLL.
Enable rx_pma_clkslip port	**On/Off**	Enables the optional `rx_pma_clkslip` control input port. When asserted, causes the deserializer to either skip one serial bit or pauses the serial clock for one cycle to achieve word alignment.
Enable rx_is_lockedtodata port	**On/Off**	Enables the optional `rx_is_lockedtodata` status output port. This signal indicates that the RX CDR is currently in lock to data mode or is attempting to lock to the incoming data stream. This is an asynchronous output signal.
Enable rx_is_lockedtoref port	**On/Off**	Enables the optional `rx_is_lockedtoref` status output port. This signal indicates that the RX CDR is currently locked to the CDR reference clock. This is an asynchronous output signal.
Enable rx_set_lockedtodata port and rx_set_lockedtoref ports	**On/Off**	Enables the optional `rx_set_lockedtodata` and `rx_set_lockedtoref` control input ports. You can use these control ports to manually control the lock mode of the RX CDR. These are asynchronous input signals.
Enable PRBS (Pseudo Random Bit Sequence) verifier control and status ports	**On/Off**	Enables the optional `rx_prbs_err`, `rx_prbs_clr`, and `rx_prbs_done` control ports. These ports control and collect status from the internal PRBS verifier.
Enable rx_seriallpbken port	**On/Off**	Enables the optional `rx_seriallpbken` control input port. The assertion of this signal enables the TX to RX serial loopback path within the transceiver. This is an asynchronous input signal.

Related Information

- PLLs and Clock Networks on page 249
- Channel Bonding on page 299
- Transceiver PHY PCS-to-Core Interface Reference Port Mapping on page 86

2.3.5. PCS-Core Interface Parameters

This section defines parameters available in the Native PHY IP core GUI to customize the PCS to core interface. The following table describes the available parameters. Based on the selection of the Transceiver Configuration Rule , if the specified settings violate the protocol standard, the **Native PHY IP core Parameter Editor** prints error or warning messages.

Table 18. PCS-Core Interface Parameters

Parameter	Range	Description
General Interface Options		
Enable PCS reset status ports	**On / Off**	Enables the optional TX digital reset and RX digital reset release status output ports including: • `tx_transfer_ready` • `rx_transfer_ready` • `tx_fifo_ready`

continued...

Parameter	Range	Description
		• `rx_fifo_ready` • `tx_digitalreset_timeout` • `rx_digitalreset_timeout` The PCS reset status ports help you to debug on why the transceiver native phy does not come out of reset. You can use these ports to debug common connectivity issues, such as the `tx/rx_coreclkin` being undriven, incorrect frequency, or FIFOs not being set properly. Please refer to the "Debugging with the PCS reset status ports" section for more detail.
	TX PCS-Core Interface FIFO	
TX Core Interface FIFO Mode	**Phase-Compensation** **Register** **Interlaken** **Basic**	The TX PCS FIFO is always operating in Phase Compensation mode. The selection range specifies one of the following modes for the TX Core FIFO: • **Phase Compensation**: The TX Core FIFO compensates for the clock phase difference between the read clock `tx_clkout` and the write clocks `tx_coreclkin` or `tx_clkout`. • **Register**: This mode is limited to PCS Direct with interface widths of 40 bits or less. The TX Core FIFO is bypassed. You must connect the write clock `tx_coreclkin` to the read clock `tx_clkout`. The `tx_parallel_data`, `tx_control` and `tx_enh_data_valid` are registered at the FIFO output. Assert `tx_enh_data_valid` port 1'b1 at all times. • **Interlaken**: The TX Core FIFO acts as an elastic buffer. In this mode, there are additional signals to control the data flow into the FIFO. Therefore, the FIFO write clock frequency does not have to be the same as the read clock frequency. You can control writes to the FIFO with `tx_fifo_wr_en`. By monitoring the FIFO flags, you can avoid the FIFO full and empty conditions. The Interlaken frame generator controls reading of the data from the TX FIFO. • **Basic**: The TX Core FIFO acts as an elastic buffer. This mode allows driving write and read side of FIFO with different clock frequencies. Monitor FIFO flag to control write and read operations. For additional details refer to *Enhanced PCS FIFO Operation* section. Refer to the *Special TX PCS Reset Release Sequence* section to see if you need to implement a special reset release sequence in your top-level code.
TX FIFO partially full threshold	**0-31**	Specifies the partially full threshold for the PCS TX Core FIFO. Enter the value at which you want the TX Core FIFO to flag a partially full status.
TX FIFO partially empty threshold	**0-31**	Specifies the partially empty threshold for the PCS TX Core FIFO. Enter the value at which you want the TX Core FIFO to flag a partially empty status.
Enable tx_fifo_full port	**On** / **Off**	Enables the **tx_fifo_full port**. This signal indicates when the TX Core FIFO is full. This signal is synchronous to `tx_coreclkin`.
Enable tx_fifo_empty port	**On** / **Off**	Enables the **tx_fifo_empty port**. This signal indicates when the TX Core FIFO is empty. This is an asynchronous signal.
Enable tx_fifo_pfull port	**On** / **Off**	Enables the **tx_fifo_pfull port**. This signal indicates when the TX Core FIFO reaches the specified partially full threshold. This signal is synchronous to `tx_coreclkin`.
Enable tx_fifo_pempty port	**On** / **Off**	Enables the **tx_fifo_pempty port**. This signal indicates when the Core TX FIFO reaches the specified partially empty threshold. This is an asynchronous signal.

continued...

Parameter	Range	Description
Enable tx_dll_lock port	**On/Off**	Enables the transmit delay locked-loop port. This signal is synchronous to `tx_clkout`.
	RX PCS-Core Interface FIFO	
RX PCS-Core Interface FIFO Mode	**Phase-Compensation** **Phase-Compensation - Register** **Phase Compensation - Basic** **Register** **Register - Phase Compensation** **Register - Basic** **Interlaken** **10GBASE-R**	Specifies one of the following modes for PCS RX FIFO: • **Phase Compensation**: This mode places both the RX PCS FIFO and RX Core FIFO in Phase Compensation mode. It compensates for the clock phase difference between the read clocks `rx_coreclkin` or `tx_clkout` and the write clock `rx_clkout`. • **Phase Compensation-Register**: This mode places the RX PCS FIFO in Phase Compensation mode and the RX Core FIFO in Register Mode. The RX Core FIFO's read clock `rx_coreclkin` and write clock `rx_clkout` are tied together. With double rate transfer mode disabled, this mode is limited to Standard PCS PMA widths combinations of 8, 10, 16, or 20 with byte serializer/deserializer disabled and Enhanced PCS with Gearbox Ratios of 32:32 or 40:40 and PCS Direct with interface widths of 40-bits or less. Additional configurations can be supported with double rate transfer mode enabled. • **Phase Compensation-Basic**: This mode places the RX PCS FIFO in Phase Compensation mode and the RX Core FIFO in Basic Mode. This mode can only be used with Enhanced PCS and PCS Direct. The RX Core FIFO in Basic mode acts as an elastic buffer or clock crossing FIFO similar to Interlaken mode where the `rx_coreclkin` and `rx_clkout` can be asynchronous and of different frequencies. You must implement a FSM that monitors the FIFO status flags and manage the FIFO read and write enable in preventing the FIFO overflow and underflow conditions. • **Register** : This mode is limited to PCS Direct with interface widths of 40 bits or less. The RX PCS FIFO and RX Core FIFO is bypassed. The FIFO's read clock `rx_coreclkin` and write clock `rx_clkout` are tied together. The `rx_parallel_data`, `rx_control`, and `rx_enh_data_valid` are registered at the FIFO output. • **Register-Phase Compensation**: This mode places the RX PCS FIFO in Register mode and the RX Core FIFO in Phase Compensation mode. This mode is limited to Standard PCS PMA widths combinations of 8, 10, 16, or 20 with byte serializer/deserializer disabled and Enhanced PCS with Gearbox Ratios of 32:32 or 40:40 and PCS Direct with interface widths of 40-bits or less. • **Register-Basic**: This mode places the RX PCS FIFO in Register mode and the RX Core FIFO in Basic mode. This mode can only be used with Enhanced PCS with Gearbox Ratios of 32:32 or 40:40 and PCS Direct with interface widths of 40-bits or less. The RX Core FIFO in Basic mode acts as an elastic buffer or clock crossing FIFO similar to Interlaken mode where the `rx_coreclkin` and `rx_clkout` can be asynchronous and of different frequencies. You must implement a FSM that monitors the FIFO status flags and manage the FIFO read and write enable in preventing the FIFO overflow and underflow conditions. • **Interlaken**: Select this mode for the Interlaken protocol. To implement the deskew process, you must implement an FSM that controls the FIFO operation based on FIFO flags. In this mode the FIFO acts as an elastic buffer. • **10GBASE-R**: In this mode, data passes through the FIFO after block lock is achieved. OS (Ordered Sets) are deleted and Idles are inserted to compensate for the clock difference between the RX PMA clock and the fabric clock of +/- 100 ppm for a maximum packet length of 64000 bytes.

continued...

Parameter	Range	Description
		Note: The fifo status flags are for Interlaken and Basic mode only. They should be ignored in all other cases.
RX FIFO partially full threshold	**0-63**	Specifies the partially full threshold for the PCS RX Core FIFO. The default value is 5.
RX FIFO partially empty threshold	**0-63**	Specifies the partially empty threshold for the PCS RX Core FIFO. The default value is 2.
Enable RX FIFO alignment word deletion (Interlaken)	**On** / **Off**	When you turn on this option, all alignment words (sync words), including the first sync word, are removed after frame synchronization is achieved. If you enable this option, you must also enable control word deletion.
Enable RX FIFO control word deletion (Interlaken)	**On** / **Off**	When you turn on this option, Interlaken control word removal is enabled. When the Enhanced PCS RX Core FIFO is configured in **Interlaken** mode, enabling this option, removes all control words after frame synchronization is achieved. Enabling this option requires that you also enable alignment word deletion.
Enable rx_data_valid port	**On** / **Off**	Enables the **rx_data_valid port**. When asserted, this signal indicates when there is valid data on the RX parallel databus.
Enable rx_fifo_full port	**On** / **Off**	Enables the **rx_fifo_full port**. This signal is required when the RX Core FIFO is operating in Interlaken or Basic mode and indicates when the RX Core FIFO is full. This is an asynchronous signal.
Enable rx_fifo_empty port	**On** / **Off**	Enables the **rx_fifo_empty port**. This signal indicates when the RX Core FIFO is empty. This signal is synchronous to `rx_coreclkin`.
Enable rx_fifo_pfull port	**On** / **Off**	Enables the **rx_fifo_pfull port**. This signal indicates when the RX Core FIFO has reached the specified partially full threshold that is set through the Native PHY IP core **PCS-Core Interface** tab. This is an asynchronous signal.
Enable rx_fifo_pempty port	**On** / **Off**	Enables the **rx_fifo_pempty port**. This signal indicates when the RX Core FIFO has reached the specified partially empty threshold that is set through the Native PHY IP core **PCS-Core Interface** tab. This signal is synchronous to `rx_coreclkin`.
Enable rx_fifo_del port (10GBASE-R)	**On** / **Off**	Enables the optional **rx_fifo_del status output port**. This signal indicates when a word has been deleted from the RX Core FIFO. This signal is only used for 10GBASE-R transceiver configuration rule. This is an asynchronous signal.
Enable rx_fifo_insert port (10GBASE-R)	**On** / **Off**	Enables the **rx_fifo_insert port**. This signal indicates when a word has been inserted into the Core FIFO. This signal is only used for 10GBASE-R transceiver configuration rule. This signal is synchronous to `rx_coreclkin`.
Enable rx_fifo_rd_en port	**On** / **Off**	Enables the **rx_fifo_rd_en input port**. This signal is enabled to read a word from the RX Core FIFO. This signal is synchronous to `rx_coreclkin` and is required when the RX Core FIFO is operating in Interlaken or Basic mode.
Enable rx_fifo_align_clr port (Interlaken)	**On** / **Off**	Enables the **rx_fifo_align_clr input port**. Only used for Interlaken. This signal is synchronous to `rx_clkout`.

Table 19. TX Clock Options

Parameter	Range	Description
Selected tx_clkout clock source	**PCS clkout** **PCS clkout x2**	Specifies the `tx_clkout` output port source.

continued...

Parameter	Range	Description
	pma_div_clkout	
Enable tx_clkout2 port	**On/ Off**	Enables the tx_clkout2 port.
Selected tx_clkout2 clock source	**PCS clkout** **PCS clkout x2** **pma_div_clkout**	You must enable tx_clkout2 port in order to make a selection for this parameter. Specifies the tx_clkout2 output port source.
TX pma_div_clkout division factor	**Disabled** **1, 2, 33, 40, 66**	You must select the pma_div_clkout under selected tx_clkout clock source or tx_clkcout2 clock source option in order to enable a selection for this parameter. Selects the divider that generates the appropriate pma_div_clkout frequency that the tx_clkout or tx_clkout2 ports use. Example: For 10.3125 Gbps datarate, if the divider value 33 is selected, the pma_div_clkout resulting frequency is 156.25MHz.
Selected tx_coreclkin clock network	**Dedicated Clock** **Global Clock**	Specifies the clock network used to drive the tx_coreclkin input. Select "Dedicated Clock" if the tx_coreclkin input port is being driven by either tx/rx_clkout or tx/rx_clkout2 from the transceiver channel. Select "Global Clock" if the tx_coreclkin input port is being driven by the Fabric clock network. You can also select "Global Clock" if tx_coreclkin is being driven by tx/rx_clkout or tx/rx_clkout2 via the Fabric clock network.
Enable tx_coreclkin2 port	**On/ Off**	Enable this clock port to provide a fifo read clock when you have double rate transfer enabled with a PMA width of 20 without byte serialization.

Table 20. RX Clock Options

Parameter	Range	Description
Selected rx_clkout clock source	**PCS clkout** **PCS clkout x2** **pma_div_clkout**	Specifies the rx_clkout output port source.
Enable rx_clkout2 port	**On/ Off**	Enables the rx_clkout2 port.
Selected rx_clkout2 clock source	**PCS clkout** **PCS clkout x2** **pma_div_clkout**	You must enable rx_clkout2 port in order to make a selection for this parameter. Specifies the rx_clkout2 output port source.
RX pma_div_clkout division factor	**Disabled** **1, 2, 33, 40, 66**	You must select the pma_div_clkout under selected rx_clkout clock source or selected rx_clkcout2 clock source option in order to enable a selection for this parameter. Selects the divider that generates the appropriate pma_div_clkout frequency that the rx_clkout port uses. Example: For 10.3125Gbps datarate, if the divider value 33 is selected, the pma_div_clkout resulting frequency is 156.25MHz.
Selected rx_coreclkin clock network	**Dedicated Clock** **Global Clock**	Specifies the clock network used to drive the rx_coreclkin input. Select "Dedicated Clock" if the rx_coreclkin input port is being driven by either tx/rx_clkout or tx/rx_clkout2 from the transceiver channel. Select "Global Clock" if the rx_coreclkin input port is being driven by the Fabric clock network. You can also select "Global Clock" if rx_coreclkin is being driven by tx/rx_clkout or tx/rx_clkout2 via the Fabric clock network.

Table 21. Latency Measurements Options

Parameter	Range	Description
Enable latency measurement ports	**On/ Off**	Enables latency measurement ports: `tx_fifo_latency_pulse`, `rx_fifo_latency_pulse` `tx_pcs_fifo_latency_pulse`, `rx_pcs_fifo_latency_pulse`, `latency_sclk`

Related Information

- How to Enable Low Latency in Basic (Enhanced PCS) on page 146
- Enhanced PCS FIFO Operation on page 129
- Using PCS Reset Status Port on page 331
- Special TX PCS Reset Release Sequence on page 327

2.3.6. Analog PMA Settings Parameters

In older device families, such as Intel Arria® 10 and Stratix V, you can only set the analog PMA settings through the Assignment Editor or the Quartus Settings File (QSF). However, for Intel Stratix 10 transceivers, you can also set them through the Native PHY IP Parameter Editor. There is also an option to provide sample QSF assignments for the settings chosen through the Native PHY IP Parameter Editor. Use this method when you need to modify one or two individual settings, or want to modify the settings without regenerating the IP.

You can specify values for the following types of analog PMA settings parameters in the Native PHY IP Parameter Editor:

- TX analog PMA settings:
  - TX PMA analog mode rules
  - Output swing level (VOD)
  - Use default TX PMA analog settings
  - Pre-emphasis first pre-tap polarity
  - Pre-emphasis first pre-tap magnitude
  - Pre-emphasis first post-tap polarity
  - Pre-emphasis first post-tap magnitude
  - Slew rate control
  - On-chip termination
  - High-speed compensation
- RX analog PMA settings:
  - Use default RX PMA analog settings
  - RX adaptation mode
  - CTLE AC Gain
  - CTLE EQ Gain
  - VGA DC Gain
  - RX on-chip termination

Note: Even if you do not select the **Use default TX PMA analog settings** and **Use default RX PMA analog settings** options in the Intel Stratix 10 device Native PHY IP, you can use these default settings as a starting point to tune the transceiver link. The on-chip termination settings are chosen by the Intel Quartus Prime software based on data rate when the **Use default TX PMA analog settings** and **Use default RX PMA analog settings** options are enabled. You can compile your Intel Quartus Prime design and inspect the fitter results to determine the default TX and RX termination settings for your Native PHY variant.

Note: The following settings can not be set through the Native PHY IP Parameter Editor. You must set these through the Intel Quartus Prime Pro Edition Assignment Editor:

- REFCLK I/O Standard
- REFCLK Termination
- TX serial pin I/O Standard
- RX serial pin I/O Standard

To improve performance, Intel Stratix 10 FPGAs use a High Speed Differential I/O. Select **High Speed Differential I/O** as the I/O standard for the Intel Stratix 10 transmitter and receiver pins in the Intel Quartus Prime Pro Edition Assignment Editor or Quartus Settings File (**.qsf**). The **.qsf** settings always take precedence over the settings selected in the Native PHY IP Parameter Editor.

The syntax is as follows:

```
set_instance_assignment -name IO_STANDARD "HIGH SPEED
DIFFERENTIAL I/O" -to <serial TX/RX pin name> -entity <name of
the top-level file>
```

Refer to the *Dedicated Reference Clock Settings* section for details on the I/O standard and termination settings for the dedicated reference clock.

Table 22. TX Analog PMA Settings Options

Parameter	Value	Description
TX PMA analog mode rules	**User Selection**(`cei_11100_lr` to `xfp_9950`)	Selects the analog protocol mode to pre-select the TX pin swing settings (VOD, Pre-emphasis, and slew rate). After loading the pre-selected values in the GUI, if one or more of the individual TX pin swing settings need to be changed, then select the **Provide sample QSF assignments** option to modify the settings through the QSF.
Use default TX PMA analog settings	**On/Off**	Selects whether to use default or custom TX PMA analog settings.

Output Swing Level (VOD)	17 to 31 (600 mV to VCCT or Transmitter Power Supply Voltage)	Selects the transmitter programmable output differential voltage swing. (Use the *Intel Stratix 10 L-Tile/H-Tile Pre-emphasis and Output Swing Estimator* to see how changing VOD affects your signal.)
		Note: **Although the GUI displays a range of 0-31, you must not select values lower than 17.**
		Syntax:
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_vod_output_swing_ctrl=<value>" -to <serial TX pin name>`
		`set_instance_assignment -name HSSI_PARAMETER`

continued...

Parameter	Value	Description

		`"pma_tx_buf_powermode_ac_tx_vod_no_jitcomp = TX_VOD_NO_JITCOMP_AC_L0" -to <serial TX pin name>`
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_powermode_dc_tx_vod_no_jitcomp = TX_VOD_NO_JITCOMP_DC_L0" -to <serial TX pin name>`
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_powermode_ac_tx_vod_w_jitcomp = TX_VOD_W_JITCOMP_AC_L20" -to <serial TX pin name>`
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_powermode_dc_tx_vod_w_jitcomp= TX_VOD_W_JITCOMP_DC_L20" -to <serial TX pin name>`

Pre-Emphasis First Pre-Tap Polarity	**negative/positive**	Selects the polarity of the first pre-tap for pre-emphasis. (Use the *Intel Stratix 10 L-Tile/H-Tile Pre-emphasis and Output Swing Estimator* to see how changing pre-emphasis affects your signal.)
		Syntax:
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_pre_emp_sign_pre_tap_1t=fir_pre_1t_<value>" -to <serial TX pin name>`
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_powermode_ac_pre_tap = TX_PRE_TAP_AC_ON" -to <serial TX pin name>`
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_powermode_dc_pre_tap = TX_PRE_TAP_DC_ON" -to <serial TX pin name>`

Pre-Emphasis First Pre-Tap Magnitude	0 to 15 (0 to -6 dB gain for positive sign, and 0 to 6 dB gain for negative sign)	Selects the magnitude of the first pre-tap for pre-emphasis. (Use the *Intel Stratix 10 L-Tile/H-Tile Pre-emphasis and Output Swing Estimator* to see how changing pre-emphasis affects your signal.) Syntax: `set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_pre_emp_switching_ctrl_pre_tap_1t=<value>" -to <serial TX pin name>`

Pre-Emphasis First Post-Tap Polarity	**negative/positive**	Selects the polarity of the first post-tap for pre-emphasis. (Use the *Intel Stratix 10 L-Tile/H-Tile Pre-emphasis and Output Swing Estimator* to see how changing pre-emphasis affects your signal.)
		Syntax:
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_pre_emp_sign_1st_post_tap=fir_post_1t_<value>" -to <serial TX pin name>`
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_compensation_posttap_en=enable" -to <serial TX pin name>`
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_compensation_en=enable" -to <serial TX pin name>`
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_powermode_ac_post_tap = TX_POST_TAP_W_JITCOMP_AC_ON" -to <serial TX pin name>`
		`set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_powermode_dc_post_tap = TX_POST_TAP_W_JITCOMP_DC_ON" -to <serial TX pin name>`

Pre-Emphasis First Post -Tap Magnitude	0 to 24 (0 to -14 dB gain for positive sign, and 0 to 14 dB gain for negative sign)	Selects the magnitude of the first post-tap for pre-emphasis. (Use the *Intel Stratix 10 L-Tile/H-Tile Pre-emphasis and Output Swing Estimator* to see how changing pre-emphasis affects your signal.) Syntax: `set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_pre_emp_switching_ctrl_1st_post_tap=<value>" -to <serial TX pin name>`
Slew Rate Control	**0** (slowest) - **5** (fastest)	Selects the slew rate of the TX output signal. Valid values span from slowest to the fastest rate.

continued...

Parameter	Value	Description
		Syntax: `set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_slew_rate_ctrl=slew_r<value>" -to <serial TX pin name>`
On-Chip Termination	• **r_r1** (100Ω) • **r_r2** (85Ω)	Selects the on-chip TX differential termination according to the on-board trace impedance at the TX output pin. Syntax: `set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_term_sel=<value>" -to <serial TX pin name>`
High Speed Compensation	**enable/disable**	Enables the power-distribution network (PDN) induced inter-symbol interference (ISI) compensation in the TX driver. When enabled, it reduces the PDN- induced ISI jitter, but increases the power consumption. Syntax: `set_instance_assignment -name HSSI_PARAMETER "pma_tx_buf_compensation_en=<value>" -to <serial TX pin name>`

Table 23. RX Analog PMA Settings Options

Parameter	Value	Description
RX PMA analog mode rules	**User Selection**(`analog_off` to `user_custom`)	Selects the analog protocol mode rules to pre-select the RX pin swing settings (VOD, Pre-emphasis, and Slew Rate).
Use default RX PMA analog settings	**On/Off**	Selects whether to use default or custom RX PMA analog settings. *Note:* When you disable this setting by selecting **Off**, you should select one of the available options in the Native PHY IP Parameter Editor as the PMA analog settings.
RX adaptation mode	• **Manual CTLE, Manual VGA, DFE Off** • **Adaptive CTLE, Adaptive VGA, DFE Off** • **Adaptive CTLE, Adaptive VGA, All-Tap Adaptive DFE** • **Adaptive CTLE, Adaptive VGA, 1-Tap Adaptive DFE** • **ctle_dfe_mode_2 (Adaptive mode for PCIe Gen3)**	Select manual CTLE if you intend to tune the analog front end of all the transceiver channels by sweeping combinations of the TX and RX EQ parameters together. Select one of the adaptive modes based on your system loss characteristics if you intend to use the Adaptation engine in the RX PMA. Only use `ctle_dfe_mode_2` for PCIe Gen3. When using any of the adaptive modes, refer to the *PMA Functions* section for more information about how to reconfigure across modes, and how to start and stop adaptation.
RX On-chip Termination	• **r_r1** (80 Ω) • **r_r2** (85 Ω) • **r_r3** (91 Ω) • **r_r4** (100 Ω) • **r_r5** (103.5 Ω) • **r_r6** (108.5 Ω) • **r_unused**	Specifies the on-chip termination value for the receiver according to the on-board trace impedance at the RX input pin. Syntax: `set_instance_assignment -name HSSI_PARAMETER "pma_rx_buf_term_sel=<value>" -to <serial RX pin name>`

continued...

Parameter	Value	Description
CTLE AC Gain	0 to 15 (-2 dB at the peak to +10 dB at the peak)	Specifies the CTLE broadband gain. Syntax: `set_instance_assignment -name HSSI_PARAMETER "pma_rx_buf_ctle_ac_gain=<value>" -to <serial RX pin name>`
CTLE EQ Gain	0 to 47 (0 dB to 16 dB)	Specifies the CTLE equalization setting. Syntax: `set_instance_assignment -name HSSI_PARAMETER "pma_rx_buf_ctle_eq_gain=<value>" -to <serial RX pin name>`
VGA DC Gain	0 to 31(-5 dB to +7 dB)	Specifies the VGA Gain for the receiver. Syntax: `set_instance_assignment -name HSSI_PARAMETER "pma_rx_buf_vga_dc_gain=<value>" -to <serial RX pin name>`

Table 24. Sample QSF Assignment Option

Parameter	Value	Description
Provide sample QSF assignments	**On/Off**	Selects the option to provide QSF assignments to the above configuration, in case one or more individual values need to change. The sample QSF assignments list has different sets of attributes depending on the enabled blocks in the currently-selected analog PMA settings.

Related Information

- Dedicated Reference Clock Pins on page 279
- PMA Functions on page 108
- Intel Stratix 10 Device Family Pin Connection Guidelines
- Intel Stratix 10 L-Tile/H-Tile Pre-emphasis and Output Swing Estimator

2.3.7. Enhanced PCS Parameters

This section defines parameters available in the Native PHY IP core GUI to customize the individual blocks in the Enhanced PCS.

The following tables describe the available parameters. Based on the selection of the **Transceiver Configuration Rule** , if the specified settings violate the protocol standard, the **Native PHY IP core Parameter Editor** prints error or warning messages.

Note: For detailed descriptions about the optional ports that you can enable or disable, refer to the *Enhanced PCS Ports* section.

Table 25. Enhanced PCS Parameters

Parameter	Range	Description
Enhanced PCS / PMA interface width	**32, 40, 64**	Specifies the interface width between the Enhanced PCS and the PMA.
FPGA fabric /Enhanced PCS interface width	**32, 40, 64, 66, 67**	Specifies the interface width between the Enhanced PCS and the FPGA fabric. The 66-bit FPGA fabric to PCS interface width uses 64-bits from the TX and RX parallel data. The block synchronizer determines the block boundary of the 66-bit word, with lower 2 bits from the control bus. The 67-bit FPGA fabric to PCS interface width uses the 64-bits from the TX and RX parallel data. The block synchronizer determines the block boundary of the 67-bit word with lower 3 bits from the control bus.
Enable 'Enhanced PCS' low latency mode	**On/Off**	Enables the low latency path for the Enhanced PCS. When you turn on this option, the individual functional blocks within the Enhanced PCS are bypassed to provide the lowest latency path from the PMA through the Enhanced PCS. When enabled, this mode is applicable for GX transceiver channels. Intel recommends not enabling it for GXT transceiver channels..

Table 26. Interlaken Frame Generator Parameters

Parameter	Range	Description
Enable Interlaken frame generator	**On / Off**	Enables the frame generator block of the Enhanced PCS.
Frame generator metaframe length	**5-8192**	Specifies the metaframe length of the frame generator. This metaframe length includes 4 framing control words created by the frame generator.
Enable Frame Generator Burst Control	**On / Off**	Enables frame generator burst. This determines whether the frame generator reads data from the TX FIFO based on the input of port `tx_enh_frame_burst_en`.
Enable tx_enh_frame port	**On / Off**	Enables the `tx_enh_frame` status output port. When the Interlaken frame generator is enabled, this signal indicates the beginning of a new metaframe. This is an asynchronous signal.
Enable tx_enh_frame_diag_status port	**On / Off**	Enables the `tx_enh_frame_diag_status` 2-bit input port. When the Interlaken frame generator is enabled, the value of this signal contains the status message from the framing layer diagnostic word. This signal is synchronous to `tx_clkout`.
Enable tx_enh_frame_burst_en port	**On / Off**	Enables the `tx_enh_frame_burst_en` input port. When burst control is enabled for the Interlaken frame generator, this signal is asserted to control the frame generator data reads from the TX FIFO. This signal is synchronous to `tx_clkout`.

Table 27. Interlaken Frame Synchronizer Parameters

Parameter	Range	Description
Enable Interlaken frame synchronizer	**On / Off**	When you turn on this option, the Enhanced PCS frame synchronizer is enabled.
Frame synchronizer metaframe length	**5-8192**	Specifies the metaframe length of the frame synchronizer.

continued...

Parameter	Range	Description
Enable rx_enh_frame port	**On / Off**	Enables the **rx_enh_frame status output port**. When the Interlaken frame synchronizer is enabled, this signal indicates the beginning of a new metaframe. This is an asynchronous signal.
Enable rx_enh_frame_lock port	**On / Off**	Enables the **rx_enh_frame_lock output port**. When the Interlaken frame synchronizer is enabled, this signal is asserted to indicate that the frame synchronizer has achieved metaframe delineation. This is an asynchronous output signal.
Enable rx_enh_frame_diag_status port	**On / Off**	Enables the**rx_enh_frame_diag_status output port**. When the Interlaken frame synchronizer is enabled, this signal contains the value of the framing layer diagnostic word (bits [33:32]). This is a 2 bit per lane output signal. It is latched when a valid diagnostic word is received. This is an asynchronous signal.

Table 28. Interlaken CRC32 Generator and Checker Parameters

Parameter	Range	Description
Enable Interlaken TX CRC-32 Generator	**On / Off**	When you turn on this option, the TX Enhanced PCS datapath enables the CRC32 generator function. CRC32 can be used as a diagnostic tool. The CRC contains the entire metaframe including the diagnostic word.
Enable Interlaken TX CRC-32 generator error insertion	**On / Off**	When you turn on this option, the error insertion of the interlaken CRC-32 generator is enabled. Error insertion is cycle-accurate. When this feature is enabled, the assertion of `tx_control[8]` or `tx_err_ins` signal causes the CRC calculation during that word is incorrectly inverted, and thus, the CRC created for that metaframe is incorrect.
Enable Interlaken RX CRC-32 checker	**On / Off**	Enables the CRC-32 checker function.
Enable rx_enh_crc32_err port	**On / Off**	When you turn on this option, the Enhanced PCS enables the **rx_enh_crc32_err port**. This signal is asserted to indicate that the CRC checker has found an error in the current metaframe. This is an asynchronous signal.

Table 29. 10GBASE-R BER Checker Parameters

Parameter	Range	Description
Enable rx_enh_highber port (10GBASE-R)	**On / Off**	Enables the **rx_enh_highber port**. For 10GBASE-R transceiver configuration rule, this signal is asserted to indicate a bit error rate higher than 10^{-4}. Per the 10GBASE-R specification, this occurs when there are at least 16 errors within 125 µs. This is an asynchronous signal.
Enable rx_enh_highber_clr_cnt port (10GBASE-R)	**On / Off**	Enables the **rx_enh_highber_clr_cnt input port**. For the 10GBASE-R transceiver configuration rule, this signal is asserted to clear the internal counter. This counter indicates the number of times the BER state machine has entered the "BER_BAD_SH" state. This is an asynchronous signal.
Enable rx_enh_clr_errblk_count port (10GBASE-R&FEC)	**On / Off**	Enables the **rx_enh_clr_errblk_count input port**. For the 10GBASE-R transceiver configuration rule, this signal is asserted to clear the internal counter. This counter indicates the number of the times the RX state machine has entered the RX_E state. For protocols with FEC block enabled, this signal is asserted to reset the status counters within the RX FEC block. This is an asynchronous signal.

Table 30. 64b/66b Encoder and Decoder Parameters

Parameter	Range	Description
Enable TX 64b/66b encoder (10GBASE-R)	**On** / **Off**	When you turn on this option, the Enhanced PCS enables the TX 64b/66b encoder.
Enable RX 64b/66b decoder (10GBASE-R)	**On** / **Off**	When you turn on this option, the Enhanced PCS enables the RX 64b/66b decoder.
Enable TX sync header error insertion	**On** / **Off**	When you turn on this option, the Enhanced PCS supports cycle-accurate error creation to assist in exercising error condition testing on the receiver. When error insertion is enabled and the error flag is set, the encoding sync header for the current word is generated incorrectly. If the correct sync header is 2'b01 (control type), 2'b00 is encoded. If the correct sync header is 2'b10 (data type), 2'b11 is encoded.

Table 31. Scrambler and Descrambler Parameters

Parameter	Range	Description
Enable TX scrambler (10GBASE-R/ Interlaken)	**On** / **Off**	Enables the scrambler function. This option is available for the Basic (Enhanced PCS) mode, Interlaken, and 10GBASE-R protocols. You can enable the scrambler in Basic (Enhanced PCS) mode when the block synchronizer is enabled and with 66:32, 66:40, or 66:64 gear box ratios.
TX scrambler seed (10GBASE-R/ Interlaken)	User-specified 58-bit value	You must provide a non-zero seed for the Interlaken protocol. For a multi-lane Interlaken Transceiver Native PHY IP, the first lane scrambler has this seed. For other lanes' scrambler, this seed is increased by 1 per each lane. The initial seed for 10GBASE-R is 0x03FFFFFFFFFFFFFF. This parameter is required for the 10GBASE-R and Interlaken protocols.
Enable RX descrambler (10GBASE-R/ Interlaken)	**On** / **Off**	Enables the descrambler function. This option is available for Basic (Enhanced PCS) mode, Interlaken, and 10GBASE-R protocols. You can enable the descrambler in Basic (Enhanced PCS) mode with the block synchronizer enabled and with 66:32, 66:40, or 66:64 gear box ratios.

Table 32. Interlaken Disparity Generator and Checker Parameters

Parameter	Range	Description
Enable Interlaken TX disparity generator	**On** / **Off**	When you turn on this option, the Enhanced PCS enables the disparity generator. This option is available for the Interlaken protocol.
Enable Interlaken RX disparity checker	**On** / **Off**	When you turn on this option, the Enhanced PCS enables the disparity checker. This option is available for the Interlaken protocol.
Enable Interlaken TX random disparity bit	**On** / **Off**	Enables the Interlaken random disparity bit. When enabled, a random number is used as disparity bit which saves one cycle of latency.

Table 33. Block Synchronizer Parameters

Parameter	Range	Description
Enable RX block synchronizer	**On** / **Off**	When you turn on this option, the Enhanced PCS enables the RX block synchronizer. This options is available for the Basic (Enhanced PCS) mode, Interlaken, and 10GBASE-R protocols.
Enable rx_enh_blk_lock port	**On** / **Off**	Enables the `rx_enh_blk_lock` port. When you enable the block synchronizer, this signal is asserted to indicate that the block delineation has been achieved.

Table 34. Gearbox Parameters

Parameter	Range	Description
Enable TX data bitslip	**On** / **Off**	When you turn on this option, the TX gearbox operates in bitslip mode. The **tx_enh_bitslip port** controls number of bits which TX parallel data slips before going to the PMA.
Enable TX data polarity inversion	**On** / **Off**	When you turn on this option, the polarity of TX data is inverted. This allows you to correct incorrect placement and routing on the PCB.
Enable RX data bitslip	**On** / **Off**	When you turn on this option, the Enhanced PCS RX block synchronizer operates in bitslip mode. When enabled, the **rx_bitslip port** is asserted on the rising edge to ensure that RX parallel data from the PMA slips by one bit before passing to the PCS.
Enable RX data polarity inversion	**On** / **Off**	When you turn on this option, the polarity of the RX data is inverted. This allows you to correct incorrect placement and routing on the PCB.
Enable tx_enh_bitslip port	**On** / **Off**	Enables the **tx_enh_bitslip port**. When TX bit slip is enabled, this signal controls the number of bits which TX parallel data slips before going to the PMA.
Enable rx_bitslip port	**On** / **Off**	Enables the **rx_bitslip port**. When RX bit slip is enabled, the `rx_bitslip` signal is asserted on the rising edge to ensure that RX parallel data from the PMA slips by one bit before passing to the PCS. This port is shared between Standard PCS and Enhanced PCS.

Table 35. KR-FEC Parameters

Parameter	Range	Description
Enable RX KR-FEC error marking	**On/Off**	When you turn on this option, the decoder asserts both sync bits (2'b11) when it detects an uncorrectable error. This feature increases the latency through the KR-FEC decoder.
Error marking type	**10G, 40G**	Specifies the error marking type (10G or 40G).
Enable KR-FEC TX error insertion	**On/Off**	Enables the error insertion feature of the KR-FEC encoder. This feature allows you to insert errors by corrupting data starting a bit 0 of the current word.
KR-FEC TX error insertion spacing	**User Input** (1 bit to 15 bit)	Specifies the spacing of the KR-FEC TX error insertion.
Enable tx_enh_frame port	**On/Off**	Enables the **tx_enh_frame port**. Asynchronous status flag output of the TX KR-FEC that signifies the beginning of the generated KR-FEC frame.
Enable rx_enh_frame port	**On/Off**	Enables the **rx_enh_frame port**. Asynchronous status flag output of the RX KR-FEC that signifies the beginning of the received KR-FEC frame.
Enable rx_enh_frame_diag_status port	**On/Off**	Enables the **rx_enh_frame_diag_status port**. Asynchronous status flag output of the RX KR-FEC that indicates the status of the current received KR-FEC frame. • 00: No error • 01: Correctable error • 10: Uncorrectable error • 11: Reset condition/pre-lock condition

Related Information

Enhanced PCS Ports on page 73

2.3.8. Standard PCS Parameters

This section provides descriptions of the parameters that you can specify to customize the Standard PCS.

For specific information about configuring the Standard PCS for these protocols, refer to the sections of this user guide that describe support for these protocols.

Table 36. Standard PCS Parameters

Note: For detailed descriptions of the optional ports that you can enable or disable, refer to the *Standard PCS Ports* section.

Parameter	Range	Description
Standard PCS/PMA interface width	**8, 10, 16, 20**	Specifies the data interface width between the Standard PCS and the transceiver PMA.
FPGA fabric/Standard TX PCS interface width	**8, 10, 16, 20, 32, 40**	Shows the FPGA fabric to TX PCS interface width. This value is automatically determined by the current configuration of individual blocks within the Standard TX PCS datapath.
FPGA fabric/Standard RX PCS interface width	**8, 10, 16, 20, 32, 40**	Shows the FPGA fabric to RX PCS interface width. This value is automatically determined by the current configuration of individual blocks within the Standard RX PCS datapath.
Enable 'Standard PCS' low latency mode	**On** / **Off**	Enables the low latency path for the Standard PCS. Some of the functional blocks within the Standard PCS are bypassed to provide the lowest latency. You cannot turn on this parameter while using the **Basic/Custom w/Rate Match (Standard PCS)** specified for **Transceiver configuration rules**.

Table 37. Byte Serializer and Deserializer Parameters

Parameter	Range	Description
TX byte serializer mode	**Disabled** **Serialize x2** **Serialize x4**	Specifies the TX byte serializer mode for the Standard PCS. The transceiver architecture allows the Standard PCS to operate at double or quadruple the data width of the PMA serializer. The byte serializer allows the PCS to run at a lower internal clock frequency to accommodate a wider range of FPGA interface widths. **Serialize x4** is only applicable for PCIe protocol implementation.
RX byte deserializer mode	**Disabled** **Deserialize x2** **Deserialize x4**	Specifies the mode for the RX byte deserializer in the Standard PCS. The transceiver architecture allows the Standard PCS to operate at double or quadruple the data width of the PMA deserializer. The byte deserializer allows the PCS to run at a lower internal clock frequency to accommodate a wider range of FPGA interface widths. **Deserialize x4** is only applicable for PCIe protocol implementation.

Table 38. 8B/10B Encoder and Decoder Parameters

Parameter	Range	Description
Enable TX 8B/10B encoder	**On** / **Off**	When you turn on this option, the Standard PCS enables the TX 8B/10B encoder.
Enable TX 8B/10B disparity control	**On** / **Off**	When you turn on this option, the Standard PCS includes disparity control for the 8B/10B encoder. You can force the disparity of the 8B/10B encoder using the `tx_forcedisp` control signal.
Enable RX 8B/10B decoder	**On** / **Off**	When you turn on this option, the Standard PCS includes the 8B/10B decoder.

Table 39. Rate Match FIFO Parameters

Parameter	Range	Description
RX rate match FIFO mode	**Disabled** **Basic 10-bit PMA** **Basic 20-bit PMA** **GbE** **PIPE** **PIPE 0ppm**	Specifies the operation of the RX rate match FIFO in the Standard PCS. Rate Match FIFO in Basic (Single Width) Mode Rate Match FIFO Basic (Double Width) Mode Rate Match FIFO for GbE Transceiver Channel Datapath for PIPE
RX rate match insert/ delete -ve pattern (hex)	User-specified 20 bit pattern	Specifies the -ve (negative) disparity value for the RX rate match FIFO as a hexadecimal string.
RX rate match insert/ delete +ve pattern (hex)	User-specified 20 bit pattern	Specifies the +ve (positive) disparity value for the RX rate match FIFO as a hexadecimal string.
Enable rx_std_rmfifo_full port	**On** / **Off**	Enables the optional `rx_std_rmfifo_full` port.
Enable rx_std_rmfifo_empty port	**On** / **Off**	Enables the `rx_std_rmfifo_empty` port.
PCI Express Gen3 rate match FIFO mode	**Bypass** **0 ppm** **600 ppm**	Specifies the PPM tolerance for the PCI Express Gen3 rate match FIFO. It is bypassed by default.

Table 40. Word Aligner and Bitslip Parameters

Parameter	Range	Description
Enable TX bitslip	**On** / **Off**	When you turn on this option, the PCS includes the bitslip function. The outgoing TX data can be slipped by the number of bits specified by the `tx_std_bitslipboundarysel` control signal.
Enable tx_std_bitslipboundarysel port	**On** / **Off**	Enables the `tx_std_bitslipboundarysel` control signal.
RX word aligner mode	**bitslip** **manual (FPGA Fabric controlled)** **synchronous state machine** **deterministic latency**	Specifies the RX word aligner mode for the Standard PCS. The word aligned width depends on the PCS and PMA width, and whether or not 8B/10B is enabled. Refer to "Word Aligner" for more information.
RX word aligner pattern length	**7**, **8**, **10**, **16**, **20**, **32**, **40**	Specifies the length of the pattern the word aligner uses for alignment. Refer to "RX Word Aligner Pattern Length" table in "Word Aligner". It shows the possible values of "Rx Word Aligner Pattern Length" in all available word aligner modes.
RX word aligner pattern (hex)	User-specified	Specifies the word alignment pattern up to 16 characters in hex.
Number of word alignment patterns to achieve sync	**0-255**	Specifies the number of valid word alignment patterns that must be received before the word aligner achieves synchronization lock. The default is 3.
Number of invalid words to lose sync	**0-63**	Specifies the number of invalid data codes or disparity errors that must be received before the word aligner loses synchronization. The default is 3.
Number of valid data words to decrement error count	**0-255**	Specifies the number of valid data codes that must be received to decrement the error counter. If the word aligner receives enough valid data codes to decrement the error count to 0, the word aligner returns to synchronization lock.

continued...

Parameter	Range	Description
Enable fast sync status reporting for deterministic Latency SM	**On** / **Off**	When enabled, the `rx_syncstatus` asserts high immediately after the deserializer has completed slipping the bits to achieve word alignment. When it is not selected, `rx_syncstatus` asserts after the cycle slip operation is complete and the word alignment pattern is detected by the PCS (i.e. `rx_patterndetect` is asserted). This parameter is only applicable when the selected protocol is CPRI (Auto).
Enable rx_std_wa_patternalign port	**On** / **Off**	Enables the `rx_std_wa_patternalign` port. When the word aligner is configured in manual mode and when this signal is enabled, the word aligner aligns to next incoming word alignment pattern.
Enable rx_std_wa_a1a2size port	**On** / **Off**	Enables the optional `rx_std_wa_a1a2size` control input port.
Enable rx_std_bitslipboundarysel port	**On** / **Off**	Enables the optional `rx_std_bitslipboundarysel` status output port.
Enable rx_bitslip port	**On** / **Off**	Enables the `rx_bitslip` port. This port is shared between the Standard PCS and Enhanced PCS.

Table 41. Bit Reversal and Polarity Inversion

Parameter	Range	Description
Enable TX bit reversal	**On** / **Off**	When you turn on this option, the 8B/10B Encoder reverses TX parallel data before transmitting it to the PMA for serialization. The transmitted TX data bit order is reversed. The normal order is LSB to MSB. The reverse order is MSB to LSB. TX bit reversal ports are not available but can be changed via soft registers. RX bit reversal ports are available.
Enable TX byte reversal	**On** / **Off**	When you turn on this option, the 8B/10B Encoder reverses the byte order before transmitting data. This function allows you to reverse the order of bytes that were erroneously swapped. The PCS can swap the ordering of either one of the 8- or 10-bit words, when the PCS/PMA interface width is 16 or 20 bits. This option is not valid under certain **Transceiver configuration rules**. TX byte reversal ports are not available but can be changed via soft registers. RX bit reversal ports are available.
Enable TX polarity inversion	**On** / **Off**	When you turn on this option, the `tx_std_polinv` port controls polarity inversion of TX parallel data to the PMA. When you turn on this parameter, you also need to turn on the **Enable tx_polinv port**.
Enable tx_polinv port	**On** / **Off**	When you turn on this option, the `tx_polinv` input control port is enabled. You can use this control port to swap the positive and negative signals of a serial differential link, if they were erroneously swapped during board layout.
Enable RX bit reversal	**On** / **Off**	When you turn on this option, the word aligner reverses RX parallel data. The received RX data bit order is reversed. The normal order is LSB to MSB. The reverse order is MSB to LSB. When you enable **Enable RX bit reversal**, you must also enable **Enable rx_std_bitrev_ena port**.
Enable rx_std_bitrev_ena port	**On** / **Off**	When you turn on this option and assert the `rx_std_bitrev_ena` control port, the RX data order is reversed. The normal order is LSB to MSB. The reverse order is MSB to LSB.

continued...

Parameter	Range	Description
Enable RX byte reversal	**On / Off**	When you turn on this option, the word aligner reverses the byte order, before storing the data in the RX FIFO. This function allows you to reverse the order of bytes that are erroneously swapped. The PCS can swap the ordering of either one of the 8- or 10-bit words, when the PCS / PMA interface width is 16 or 20 bits. This option is not valid under certain **Transceiver configuration rules**. When you enable **Enable RX byte reversal**, you must also select the **Enable rx_std_byterev_ena port**.
Enable rx_std_byterev_ena port	**On / Off**	When you turn on this option and assert the `rx_std_byterev_ena` input control port, the order of the individual 8- or 10-bit words received from the PMA is swapped.
Enable RX polarity inversion	**On / Off**	When you turn on this option, the `rx_std_polinv` port inverts the polarity of RX parallel data. When you turn on this parameter, you also need to enable **Enable rx_polinv port**.
Enable rx_polinv port	**On / Off**	When you turn on this option, the `rx_polinv` input is enabled. You can use this control port to swap the positive and negative signals of a serial differential link if they were erroneously swapped during board layout.
Enable rx_std_signaldetect port	**On / Off**	When you turn on this option, the optional `rx_std_signaldetect` output port is enabled. This signal is required for the PCI Express protocol. If enabled, the signal threshold detection circuitry senses whether the signal level present at the RX input buffer is above the signal detect threshold voltage that you specified.

Table 42. PCIe Ports

Parameter	Range	Description
Enable PCIe dynamic datarate switch ports	**On / Off**	When you turn on this option, the `pipe_rate`, `pipe_sw`, and `pipe_sw_done` ports are enabled. You should connect these ports to the PLL IP core instance in multi-lane PCIe Gen2 and Gen3 configurations. The `pipe_sw` and `pipe_sw_done` ports are only available for multi-lane bonded configurations.
Enable PCIe electrical idle control and status ports	**On / Off**	When you turn on this option, the `pipe_rx_eidleinfersel` and `pipe_rx_elecidle` ports are enabled. These ports are used for PCI Express configurations.
Enable PCIe pipe_hclk_in and pipe_hclk_out ports	**On / Off**	When you turn on this option, the `pipe_hclk_in`, and `pipe_hclk_out` ports are enabled. These ports must be connected to the PLL IP core instance for the PCI Express configurations.

Related Information

- Standard PCS Ports on page 80
- Intel Stratix 10 Standard PCS Architecture on page 366
- Intel Stratix 10 (L/H-Tile) Word Aligner Bitslip Calculator
  Use this tool to calculate the number of slips you require to achieve alignment based on the word alignment pattern and length.

2.3.9. PCS Direct Datapath Parameters

Table 43. PCS Direct Datapath Parameters

Parameter	Range	Description
PCS Direct interface width	8, 10, 16, 20, 32, 40, 64	Specifies the data interface width between the FPGA Fabric width and the transceiver PMA.

2.3.10. Dynamic Reconfiguration Parameters

Dynamic reconfiguration allows you to change the behavior of the transceiver channels and PLLs without powering down the device.

Each transceiver channel and PLL includes an Avalon memory-mapped interface slave for reconfiguration. This interface provides direct access to the programmable address space of each channel and PLL. Because each channel and PLL includes a dedicated Avalon memory-mapped interface slave, you can dynamically modify channels either concurrently or sequentially. If your system does not require concurrent reconfiguration, you can parameterize the Transceiver Native PHY IP to share a single reconfiguration interface.

You can use dynamic reconfiguration to change many functions and features of the transceiver channels and PLLs. For example, you can change the reference clock input to the TX PLL. You can also change between the Standard and Enhanced datapaths.

To enable Intel Stratix 10 transceiver toolkit capability in the Native PHY IP core, you must enable the following options:

- **Enable dynamic reconfiguration**
- **Enable Native PHY Debug Master Endpoint**
- **Enable capability registers**
- **Enable control and status registers**
- **Enable PRBS (Pseudo Random Binary Sequence) soft accumulators**

Table 44. Dynamic Reconfiguration

Parameter	Value	Description
Enable dynamic reconfiguration	**On/Off**	When you turn on this option, the dynamic reconfiguration interface is enabled.
Enable Native PHY Debug Master Endpoint	**On/Off**	When you turn on this option, the Transceiver Native PHY IP includes an embedded Native PHY Debug Master Endpoint (NPDME) that connects internally to the Avalon memory-mapped interface slave for dynamic reconfiguration. The NPDME can access the reconfiguration space of the transceiver. It can perform certain test and debug functions via JTAG using the System Console. This option requires you to enable the **Share reconfiguration interface** option for configurations using more than one channel.
Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE	**On/Off**	When enabled, the `reconfig_waitrequest` does not indicate the status of Avalon memory-mapped interface arbitration with PreSICE. The Avalon memory-mapped interface arbitration status

continued...

Parameter	Value	Description
		is reflected in a soft status register bit. This feature requires that the **Enable control and status registers** feature under **Optional Reconfiguration Logic** be enabled.
Share reconfiguration interface	**On/Off**	When you turn on this option, the Transceiver Native PHY IP presents a single Avalon memory-mapped interface slave for dynamic reconfiguration for all channels. In this configuration, the upper [*n*-*1*:11] address bits of the reconfiguration address bus specify the channel. The channel numbers are binary encoded. Address bits [10:0] provide the register offset address within the reconfiguration space for a channel.
Enable rcfg_tx_digitalreset_release_ctrl port	**On/Off**	Enables the `rcfg_tx_digitalreset_release_ctrl` port that dynamically controls the TX PCS reset release sequence. This port usage is mandatory when reconfiguring to or from Enhanced PCS Configurations with TX PCS Gearbox ratios of either 32:67, 40:67, and 64:67.

Table 45. Optional Reconfiguration Logic

Parameter	Value	Description
Enable capability registers	**On/Off**	Enables capability registers that provide high level information about the configuration of the transceiver channel.
Set user-defined IP identifier	**User-defined**	Sets a user-defined numeric identifier that can be read from the `user_identifier` offset when the capability registers are enabled.
Enable control and status registers	**On/Off**	Enables soft registers to read status signals and write control signals on the PHY interface through the embedded debug.
Enable PRBS (Pseudo Random Binary Sequence) soft accumulators	**On/Off**	Enables soft logic for performing PRBS bit and error accumulation when the hard PRBS generator and checker are used.

Table 46. Configuration Files

Parameter	Value	Description
Configuration file prefix	*<prefix>*	Here, the file prefix to use for generated configuration files is specified. Each variant of the Transceiver Native PHY IP should use a unique prefix for configuration files.
Generate SystemVerilog package file	**On/Off**	When you turn on this option, the Transceiver Native PHY IP generates a SystemVerilog package file, **reconfig_parameters.sv**. This file contains parameters defined with the attribute values required for reconfiguration.
Generate C header file	**On/Off**	When you turn on this option, the Transceiver Native PHY IP generates a C header file, **reconfig_parameters.h**. This file contains macros defined with the attribute values required for reconfiguration.
Generate MIF (Memory Initialize File)	**On/Off**	When you turn on this option, the Transceiver Native PHY IP generates a MIF, **reconfig_parameters.mif**. This file contains the attribute values required for reconfiguration in a data format.

Table 47. Configuration Profiles

Parameter	Value	Description
Enable multiple reconfiguration profiles	**On/Off**	When enabled, you can use the GUI to store multiple configurations. This information is used by Quartus to include the necessary timing arcs for all configurations during timing driven compilation. The Native PHY generates reconfiguration files for all of the stored

continued...

Parameter	Value	Description
		profiles. The Native PHY also checks your multiple reconfiguration profiles for consistency to ensure you can reconfigure between them. Among other things this checks that you have exposed the same ports for each configuration.[8]
Enable embedded reconfiguration streamer	**On/Off**	Enables the embedded reconfiguration streamer, which automates the dynamic reconfiguration process between multiple predefined configuration profiles. This is optional and increases logic utilization. The PHY includes all of the logic and data necessary to dynamically reconfigure between pre-configured profiles.
Generate reduced reconfiguration files	**On/Off**	When enabled, The Native PHY generates reconfiguration report files containing only the attributes or RAM data that are different between the multiple configured profiles. The reconfiguration time decreases with the use of reduced .mif files.
Number of reconfiguration profiles	1-8	Specifies the number of reconfiguration profiles to support when multiple reconfiguration profiles are enabled.
Store current configuration to profile	0-7	Selects which reconfiguration profile to store/load/clear/refresh, when clicking the relevant button for the selected profile.
Store configuration to selected profile	-	Clicking this button saves or stores the current Native PHY parameter settings to the profile specified by the **Selected reconfiguration profile** parameter.
Load configuration from selected profile	-	Clicking this button loads the current Native PHY with parameter settings from the stored profile specified by the **Selected reconfiguration profile** parameter.
Clear selected profile	-	Clicking this button clears or erases the stored Native PHY parameter settings for the profile specified by the **Selected reconfiguration profile** parameter. An empty profile defaults to the current parameter settings of the Native PHY.
Clear all profiles	-	Clicking this button clears the Native PHY parameter settings for all the profiles.
Refresh selected profile	-	Clicking this button is equivalent to clicking the **Load configuration from selected profile** and **Store configuration to selected profile** buttons in sequence. This operation loads the Native PHY parameter settings from stored profile specified by the **Selected reconfiguration profile** parameter and subsequently stores or saves the parameters back to the profile.

Related Information

Reconfiguration Interface and Dynamic Reconfiguration on page 394

2.3.11. Generation Options Parameters

Table 48. Generation Options

Parameter	Value	Description
Generate parameter documentation file	**On/Off**	When you turn on this option, generation produces a Comma-Separated Value (**.csv**) file with descriptions of the Transceiver Native PHY IP parameters.

(8) For more information on timing closure, refer to the *Reconfiguration Interface and Dynamic Reconfiguration* chapter.

Send Feedback

2.3.12. PMA, Calibration, and Reset Ports

This section describes the PMA and calibration ports for the Transceiver Native PHY IP core.

In the following tables, the variables represent these parameters:

- <n>—The number of lanes
- <d>—The serialization factor
- <s>—The symbol size
- <p>—The number of PLLs

Table 49. TX PMA Ports

Name	Direction	Clock Domain	Description
`tx_serial_data[<n>-1:0]`	Input	N/A	This is the serial data output of the TX PMA.
`tx_serial_clk0`	Input	Clock	This is the serial clock from the TX PLL. The frequency of this clock depends on the datarate and clock division factor. This clock is for non bonded channels only. For bonded channels use the `tx_bonding_clocks` clock TX input.
`tx_bonding_clocks[<n><6>-1:0]`	Input	Clock	This is a 6-bit bus which carries the low speed parallel clock per channel. These clocks are outputs from the master CGB. Use these clocks for bonded channels only.
Optional Ports			
`tx_serial_clk1` `tx_serial_clk2` `tx_serial_clk3` `tx_serial_clk4`	Inputs	Clocks	These are the serial clocks from the TX PLL. These additional ports are enabled when you specify more than one TX PLL.
`tx_pma_iqtxrx_clkout`	Output	Clock	This port is available if you turn on **Enable tx_pma_iqtxrx_clkout** port in the **Transceiver Native PHY IP core Parameter Editor**. This output clock can be used to cascade the TX PMA output clock to the input of a PLL in the same tile.
`tx_pma_elecidle[<n>-1:0]`	Input	Asynchronous FSR[9]	When you assert this signal, the transmitter is forced to electrical idle. This port has no effect when you configure the transceiver for the PCI Express protocol.

Table 50. RX PMA Ports

Name	Direction	Clock Domain	Description
`rx_serial_data[<n>-1:0]`	Input	N/A	Specifies serial data input to the RX PMA.
`rx_cdr_refclk0`	Input	Clock	Specifies reference clock input to the RX clock data recovery (CDR) circuitry.
Optional Ports			

continued...

(9) For a detailed description of FSR and SSR signals, please go to the *Asynchronous Data Transfer* section in the *Other Protocols* chapter.

Name	Direction	Clock Domain	Description
`rx_cdr_refclk1–rx_cdr_refclk4`	Input	Clock	Specifies reference clock inputs to the RX clock data recovery (CDR) circuitry. These ports allow you to change the CDR datarate.
`rx_pma_iqtxrx_clkout`	Output	Clock	This port is available if you turn on **Enable rx_pma_iqtxrx_clkout port** in the **Transceiver Native PHY IP core Parameter Editor**. This output clock can be used to cascade the RX PMA output clock to the input of a PLL.
`rx_pma_clkslip`	Input	Clock SSR[9]	When asserted, causes the deserializer to either skip one serial bit or pauses the serial clock for one cycle to achieve word alignment.
`rx_is_lockedtodata[<n>-1:0]`	Output	`rx_clkout`	When asserted, indicates that the CDR PLL is in locked-to-data mode. When continuously asserted and does not switch between asserted and deasserted, you can confirm that it is actually locked to data.
`rx_is_lockedtoref[<n>-1:0]`	Output	`rx_clkout`	When asserted, indicates that the CDR PLL is in locked-to-reference mode.
`rx_set_locktodata[<n>-1:0]`	Input	Asynchronous	This port provides manual control of the RX CDR circuitry. When asserted, the CDR switches to the lock-to-data mode. Refer to the *Manual Lock Mode* section for more details.
`rx_set_locktoref[<n>-1:0]`	Input	Asynchronous	This port provides manual control of the RX CDR circuitry. When asserted, the CDR switches to the lock-to-reference mode. Refer to the *Manual Lock Mode* section for more details.
`rx_prbs_done[<n>-1:0]`	Output	`rx_coreclkin` or `rx_clkout` SSR[9]	When asserted, indicates the verifier has aligned and captured consecutive PRBS patterns and the first pass through a polynomial is complete.
`rx_prbs_err[<n>-1:0]`	Output	`rx_coreclkin` or `rx_clkout` SSR[9]	When asserted, indicates an error only after the `rx_prbs_done` signal has been asserted. This signal gets asserted for three parallel clock cycles for every error that occurs. Errors can only occur once per word.
`rx_prbs_err_clr[<n>-1:0]`	Input	`rx_coreclkin` or `rx_clkout` SSR[9]	When asserted, clears the PRBS pattern and deasserts the `rx_prbs_done` signal.
`rx_std_signaldetect[<n>-1:0]`	Output	Asynchronous	When enabled, the signal threshold detection circuitry senses whether the signal level present at the RX input buffer is above the signal detect threshold voltage. This signal is required for the PCI Express, SATA and SAS protocols.

Table 51. RX PMA Ports-PMA QPI Options

Name	Direction	Clock Domain	Description
`rx_seriallpbken[<n>-1:0]`	Input	Asynchronous SSR [10]	This port is available if you turn on **Enable rx_seriallpbken** port in the **Transceiver Native PHY IP core Parameter Editor**. The assertion of this signal enables the TX to RX serial loopback path within the transceiver. This signal can be enabled in Duplex or Simplex mode. If enabled in Simplex mode, you must drive the signal on both the TX and RX instances from the same source. Otherwise the design fails compilation.

(10) For a detailed description of FSR and SSR signals, please go to the *Asynchronous Data Transfer* section in the *Other Protocols* chapter.

Table 52. Calibration Status Ports

Name	Direction	Clock Domain	Description
`tx_cal_busy[<n>-1:0]`	Output	Asynchronous SSR[9]	When asserted, indicates that the initial TX calibration is in progress. For both initial and manual recalibration, this signal is asserted during calibration and deasserts after calibration is completed. You must hold the channel in reset until calibration completes.
`rx_cal_busy[<n>-1:0]`	Output	Asynchronous SSR[9]	When asserted, indicates that the initial RX calibration is in progress. For both initial and manual recalibration, this signal is asserted during calibration and deasserts after calibration is completed.

Table 53. Reset Ports

Name	Direction	Clock Domain[11]	Description
`tx_analogreset[<n>-1:0]`	Input	Asynchronous	Resets the PMA TX portion of the transceiver PHY.
`tx_digitalreset[<n>-1:0]`	Input	Asynchronous	Resets the PCS TX portion of the transceiver PHY.[12]
`rx_analogreset[<n>-1:0]`	Input	Asynchronous	Resets the PMA RX portion of the transceiver PHY.
`rx_digitalreset[<n>-1:0]`	Input	Asynchronous	Resets the PCS RX portion of the transceiver PHY.[13]
`tx_analogreset_stat[<n>-1:0]`	Output	Asynchronous	TX PMA reset status port.
`rx_analogreset_stat[<n>-1:0]`	Output	Asynchronous	RX PMA reset status port.
`tx_digitalreset_stat[<n>-1:0]`	Output	Asynchronous	TX PCS reset status port.
`rx_digitalreset_stat[<n>-1:0]`	Output	Asynchronous	RX PCS reset status port.

continued...

(11) Although the reset ports are not synchronous to any clock domain, Intel recommends that you synchronize the reset ports with the system clock.

(12) For non-bonded configurations: there is one bit per TX channel. For bonded configurations: there is one bit per PHY instance.

(13) For non-bonded configurations: there is one bit per RX channel. For bonded configurations: there is one bit per PHY instance.

Name	Direction	Clock Domain[11]	Description
`tx_dll_lock`	Output	Asynchronous	TX PCS delay locked loop status port. This port is available when the RX Core FIFO is operating in Interlaken or Basic mode.
Optional Reset Port			
`rcfg_tx_digitalreset_release_ctrl[<n>-1:0]`[14]	Input	Asynchronous	This port usage is mandatory when reconfiguring to or from Enhanced PCS Configurations with TX PCS Gearbox ratios of either 67:32, 67:40, and 67:64.

Related Information

- Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP Interfaces on page 335
- Asynchronous Data Transfer on page 144
- Manual Lock Mode on page 352

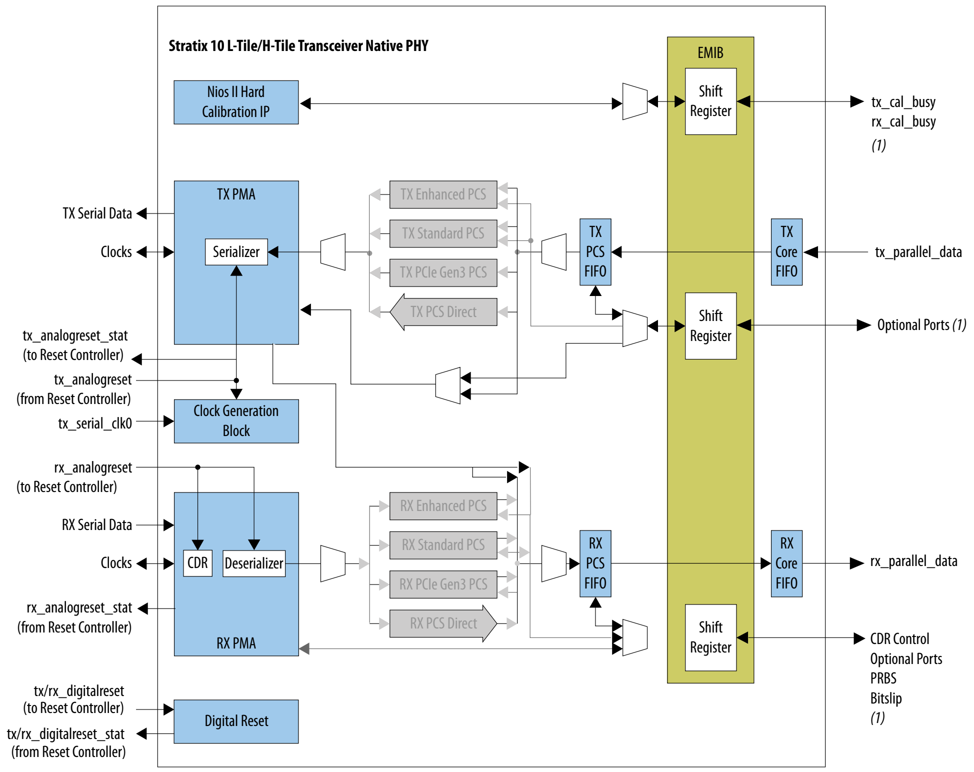
2.3.13. PCS-Core Interface Ports

This section defines the PCS-Core interface ports common to the Enhanced PCS, Standard PCS, PCIe Gen3 PCS, and PCS Direct configurations.

(14) For `rcfg_tx_digitalreset_release_ctrl` timing diagrams, refer to the "Special TX PCS Reset Release Sequence" under *Resetting Transceiver Channels* chapter.

(11) Although the reset ports are not synchronous to any clock domain, Intel recommends that you synchronize the reset ports with the system clock.

Figure 25. PCS-Core Interface Ports



Note:
1. The number of ports reflected may be greater than shown. Refer to the port description table for enhanced PCS, standard PCS, PCS direct, and PCI Express PIPE interface.

Each transceiver channel's transmit and receive 80-bit parallel data interface active and inactive ports depends on specific configuration parameters such as PMA width, FPGA Fabric width, and whether double rate transfer mode is enabled or disabled. The labeled inputs and outputs to the PMA and PCS modules represent buses, not individual signals. In the following tables, the variables represent the following parameters:

- <*n*>—The number of lanes
- <*d*>—The serialization factor
- <*s*>— The symbol size
- <*p*>—The number of PLLs

(11) Although the reset ports are not synchronous to any clock domain, Intel recommends that you synchronize the reset ports with the system clock.

Table 54. TX PCS: Parallel Data, Control, and Clocks

Name	Direction	Clock Domain	Description
`tx_parallel_data[<n>80-1:0]`	Input	Synchronous to the clock driving the write side of the FIFO (`tx_coreclkin` or `tx_clkout`)	TX parallel data inputs from the FPGA fabric to the TX PCS. If you select **Enable simplified data interface** in the **Transceiver Native PHY IP core Parameter Editor**, `tx_parallel_data` includes only the bits required for the configuration you specify. The data ports that are not active must be set to logical state zero. To determine which ports are active, refer to *Transceiver PHY PCS-to-Core Interface Port Mapping* section.
`unused_tx_parallel_data`	Input	`tx_clkout`	Port is enabled, when you enable **Enable simplified data interface**. Connect all of these bits to 0. When **Enable simplified data interface** is not set, the unused bits are a part of `tx_parallel_data`. Refer to *Transceiver PHY PCS-to-Core Interface Port Mapping* to identify the ports you need to set to logical state zero.
`tx_control[<n><3>-1:0]` or `tx_control[<n><8>-1:0]`	Input	Synchronous to the clock driving the write side of the FIFO (`tx_coreclkin` or `tx_clkout`)	The `tx_control` ports have different functionality depending on the Enhanced PCS transceiver configuration rule selected. When **Enable simplified data interface** is not set, `tx_control` is part of `tx_parallel_data`. Refer to the *Enhanced PCS TX and RX Control Ports* section for more details. Refer to *Transceiver PHY PCS-to-Core Interface Port Mapping* section for port mappings of `tx_control` ports based on specific configurations.
`tx_word_marking_bit`	Input	Synchronous to the clock driving the write side of the FIFO (`tx_coreclkin` `ortx_clkout`)	This port is required if double rate transfer mode is enabled. A logic state of Zero on this port indicates the data on `tx_parallel_data` bus contains the Lower Significant Word. A logic state of One on this port indicates the data on `tx_parallel_data` bus contains the Upper Significant Word. Note that **Enable simplified data interface** must be disabled for double rate transfer mode to be enabled and therefore, `tx_word_marking` bit always appears as part of `tx_parallel_data`. Refer to *Transceiver PHY PCS-to-Core Interface Port Mapping* section for port mappings of `tx_word_marking_bit`.
`tx_coreclkin`	Input	Clock	The FPGA fabric clock. Drives the write side of the TX FIFO. For the Interlaken protocol, the frequency of this clock could be from datarate/67 to datarate/32. Using frequency lower than this range can cause the TX FIFO to underflow and result in data corruption.
`tx_coreclkin2`	Input	Clock	Enable this clock port to provide a fifo read clock when you have double rate transfer enabled with a PMA width of 20 without byte serialization.
`tx_clkout`	Output	Clock	User has the option to select the clock source for this port between PCS clkout, PCS clkout x2, and `pma_div_clkout`. A valid clock source must be selected based on intended configuration. PCS clkout is a parallel clock generated by the local CGB for non bonded configurations, and master CGB for bonded configurations. This clocks the blocks of the TX PCS. The frequency of this clock is equal to the datarate divided by PCS/PMA interface width. PCS clkout x2 is a parallel clock generated at twice the frequency of PCS clkout for double

continued...

Send Feedback

Name	Direction	Clock Domain	Description
			transfer rate mode configurations. The frequency of `pma_div_clkout` is the divided version of the TX PMA parallel clock.
`tx_clkout2`	Output	Clock	User has the option to select the clock source for this port between PCS clkout, PCS clkout x2, and pma_div_clkout. A valid clock source must be selected based on intended configuration. In some cases, double rate transfer and bonding, for example, you may be required to enable this port for data transfer across the EMIB. PCS clkout is a parallel clock generated by the local CGB for non bonded configurations, and master CGB for bonded configurations. This clocks the blocks of the TX PCS. The frequency of this clock is equal to the datarate divided by PCS/PMA interface width. PCS clkout x2 is a parallel clock generated at twice the frequency of PCS clkout for double transfer rate mode configurations. The frequency of `pma_div_clkout` is the divided version of the TX PMA parallel clock.

Table 55. RX PCS-Core Interface Ports: Parallel Data, Control, and Clocks

Name	Direction	Clock Domain	Description
`rx_parallel_data[<n>80-1:0]`	Output	Synchronous to the clock driving the read side of the FIFO (`rx_coreclkin` or `rx_clkout`)	RX parallel data from the RX PCS to the FPGA fabric. If you select, **Enable simplified data interface** in the Transceiver Native PHY IP GUI, `rx_parallel_data` includes only the bits required for the configuration you specify. Otherwise, this interface is 80 bits wide. To determine which ports are active for specific transceiver configurations, refer to *Transceiver PHY PCS-to-Core Interface Port Mapping*. You can leave the unusual ports floating or not connected.
`unused_rx_parallel_data`	Output	`rx_clkout`	This signal specifies the unused data ports when you turn on **Enable simplified data interface**. When simplified data interface is not set, the unused ports are a part of `rx_parallel_data`. To determine which ports are active for specific transceiver configurations, refer to *Transceiver PHY PCS-to-Core Interface Port Mapping*. You can leave the unused data outputs floating or not connected.
`rx_control[<n><8>-1:0]`	Output	Synchronous to the clock driving the read side of the FIFO (`rx_coreclkin` or `rx_clkout`)	The `rx_control` ports have different functionality depending on the Enhanced PCS transceiver configuration rule selected. When Enable simplified data interface is not set, `rx_control` is part of `rx_parallel_data`. Refer to the *Enhanced PCS TX and RX Control Ports* section for more details. To determine which ports are active for specific transceiver configurations, refer to *Transceiver PHY PCS-to-Core Interface Port Mapping*.
`rx_word_marking_bit`	Output	Synchronous to the clock driving the read side of the FIFO (`rx_coreclkin` or `rx_clkout`)	This port is required if double rate transfer mode is enabled. A logic state of Zero on this port indicates the data on `rx_parallel_data` bus contains the Lower Significant Word. A logic state of One on this port indicates the data on `rx_parallel_data` bus contains the Upper Significant Word. Note that **Enable simplified data interface** must be disabled for double rate transfer mode to be enabled and therefore, `rx_word_marking` bit always appears as part of `rx_parallel_data`. Refer to *Transceiver PHY PCS-to-Core Interface Port Mapping* section for port mappings of `rx_word_marking_bit`.

continued...

Name	Direction	Clock Domain	Description
`rx_coreclkin`	Input	Clock	The FPGA fabric clock. Drives the read side of the RX FIFO. For Interlaken protocol, the frequency of this clock could range from datarate/67 to datarate/32.
`rx_clkout`	Output	Clock	User has the option to select the clock source for this port between PCS clkout, PCS clkout x2, and `pma_div_clkout`. A valid clock source must be selected based on intended configuration. The PCS clkout is the low speed parallel clock recovered by the transceiver RX PMA, that clocks the blocks in the RX PCS. The frequency of this clock is equal to datarate divided by PCS/PMA interface width. PCS clkout x2 is a parallel clock generated at twice the frequency of PCS clkout for double transfer rate mode configurations. The frequency of `pma_div_clkout` is the divided version of the RX PMA parallel clock.
`rx_clkout2`	Output	Clock	User has the option to select the clock source for this port between PCS clkout, PCS clkout x2, and `pma_div_clkout`. A valid clock source must be selected based on intended configuration. The PCS clkout is the low speed parallel clock recovered by the transceiver RX PMA, that clocks the blocks in the RX PCS. The frequency of this clock is equal to datarate divided by PCS/PMA interface width. PCS clkout x2 is a parallel clock generated at twice the frequency of PCS clkout for double transfer rate mode configurations. The frequency of `pma_div_clkout` is the divided version of the RX PMA parallel clock.

Table 56. TX PCS-Core Interface FIFO

Name	Direction	Clock Domain	Description
`tx_fifo_wr_en[<n>-1:0]`	Input	Synchronous to the clock driving the write side of the FIFO (`tx_coreclkin` or `tx_clkout`)	Assertion of this signal indicates that the TX data is valid. For Basic and Interlaken, you need to control this port based on TX Core FIFO flags so that the FIFO does not underflow or overflow. Refer to *Enhanced PCS FIFO Operation* for more details.
`tx_enh_data_valid[<n>-1:0]`	Input	Synchronous to the clock driving the write side of the FIFO (`tx_coreclkin` or `tx_clkout`)	Assertion of this signal indicates that the TX data is valid. For transceiver configuration rules using 10GBASE-R, 10GBASE-R 1588, 10GBASE-R w/KR FEC, 40GBASE-R w/KR FEC, Basic w/KR FEC, or double rate transfer mode, you must control this signal based on the gearbox ratio. You must also use this signal instead of `tx_fifo_wr_en` whenever the transceiver Enhanced PCS gearbox is not set to a 1:1 ratio such as 66:40 or 64:32 as an example, except in the case of when the RX Core FIFO is configured in Interlaken or Basic mode in which case, you must use `tx_fifo_wr_en` instead. Refer to *Enhanced PCS FIFO Operation* for more details.
`tx_fifo_full[<n>-1:0]`	Output	Synchronous to the clock driving the write side of the FIFO (`tx_coreclkin` or `tx_clkout`)	Assertion of this signal indicates the TX Core FIFO is full. Because the depth is always constant, you can ignore this signal for the phase compensation mode. Refer to *Enhanced PCS FIFO Operation* for more details.
`tx_fifo_pfull[<n>-1:0]`	Output	Synchronous to the clock driving the write side of	This signal gets asserted when the TX Core FIFO reaches its partially full threshold that is set through the Native PHY IP core **PCS-Core Interface** tab. Because the depth is always constant, you can ignore this signal for the phase compensation mode.

continued...

Name	Direction	Clock Domain	Description
		the FIFO `tx_coreclkin` or `tx_clkout`	Refer to *Enhanced PCS FIFO Operation* for more details.
`tx_fifo_empty[<n>-1:0]`	Output	Asynchronous	When asserted, indicates that the TX Core FIFO is empty. This signal gets asserted for 2 to 3 clock cycles. Because the depth is always constant, you can ignore this signal for the phase compensation mode. Refer to *Enhanced PCS FIFO Operation* for more details.
`tx_fifo_pempty[<n>-1:0]`	Output	Asynchronous	When asserted, indicates that the TX Core FIFO has reached its specified partially empty threshold that is set through the Native PHY IP core **PCS-Core Interface** tab. When you turn this option on, the Enhanced PCS enables the `tx_fifo_pempty` port, which is asynchronous. This signal gets asserted for 2 to 3 clock cycles. Because the depth is always constant, you can ignore this signal for the phase compensation mode. Refer to *Enhanced PCS FIFO Operation* for more details.

Table 57. RX PCS-Core Interface FIFO

Name	Direction	Clock Domain	Description
`rx_data_valid[<n>-1:0]`	Output	Synchronous to the clock driving the read side of the FIFO `rx_coreclkin` or `rx_clkout`	When asserted, indicates that `rx_parallel_data` is valid. Discard invalid RX parallel data when`rx_data_valid` signal is low. Refer to *Enhanced PCS FIFO Operation* for more details.
`rx_enh_data_valid[<n>-1:0]`	Output	Synchronous to the clock driving the read side of the FIFO `rx_coreclkin` or `rx_clkout`	When asserted, indicates that `rx_parallel_data` is valid. Discard invalid RX parallel data when `rx_enh_data_valid` is low. You must use this signal instead of `rx_data_valid` whenever the transceiver Enhanced PCS gearbox is not set to a 1:1 ratio such as 66:40 or 64:32 as an example, except in the case of when the RX Core FIFO is configured in Interlaken or Basic mode in which case, you must use `rx_data_valid` instead. Refer to *Enhanced PCS FIFO Operation* for more details.
`rx_fifo_full[<n>-1:0]`	Output	Asynchronous	When asserted, indicates that the RX Core FIFO is full. This signal gets asserted for 2 to 3 clock cycles.Because the depth is always constant, you can ignore this signal for the phase compensation mode. Refer to *Enhanced PCS FIFO Operation* for more details.
`rx_fifo_pfull[<n>-1:0]`	Output	Asynchronous	When asserted, indicates that the RX Core FIFO has reached its specified partially full threshold. This signal gets asserted for 2 to 3 clock cycles. Because the depth is always constant, you can ignore this signal for the phase compensation mode. Refer to *Enhanced PCS FIFO Operation* for more details.
`rx_fifo_empty[<n>-1:0]`	Output	Synchronous to the clock driving the read side of the FIFO `rx_coreclkin` or `rx_clkout`	When asserted, indicates that the RX Core FIFO is empty. Because the depth is always constant, you can ignore this signal for the phase compensation mode. Refer to *Enhanced PCS FIFO Operation* for more details.

continued...

Name	Direction	Clock Domain	Description
`rx_fifo_pempty[<n>-1:0]`	Output	Synchronous to the clock driving the read side of the FIFO `rx_coreclkin` or `rx_clkout`	When asserted, indicates that the RX Core FIFO has reached its specified partially empty threshold. Because the depth is always constant, you can ignore this signal for the phase compensation mode. Refer to *Enhanced PCS FIFO Operation* for more details.
`rx_fifo_del[<n>-1:0]`	Output	Synchronous to the clock driving the read side of the FIFO `rx_coreclkin` or `rx_clkout`	When asserted, indicates that a word has been deleted from the RX Core FIFO. This signal gets asserted for 2 to 3 clock cycles. This signal is used for the 10GBASE-R protocol.
`rx_fifo_insert[<n>-1:0]`	Output	Synchronous to the clock driving the read side of the FIFO `rx_coreclkin` or `rx_clkout`	When asserted, indicates that a word has been inserted into the RX Core FIFO. This signal is used for the 10GBASE-R protocol.
`rx_fifo_rd_en[<n>-1:0]`	Input	Synchronous to the clock driving the read side of the FIFO `rx_coreclkin` or `rx_clkout`	For RX Core FIFO Interlaken and Basic configurations, when this signal is asserted, a word is read from the RX Core FIFO. You need to control this signal based on RX Core FIFO flags so that the FIFO does not underflow or overflow.
`rx_fifo_align_clr[<n>-1:0]`	Input	Synchronous to the clock driving the read side of the FIFO `rx_coreclkin` or `rx_clkout` FSR (15)	When asserted, the RX Core FIFO resets and begins searching for a new alignment pattern. This signal is only valid for the Interlaken protocol. Assert this signal for at least 4 cycles.

Table 58. Latency Measurement/Adjustment

Name	Direction	Clock Domain	Description
`latency_sclk`	Input	clock	Latency measurement input reference clock.
`tx_fifo_latency_pulse`	Output	`tx_coreclkin`	Latency pulse of TX core FIFO from latency measurement.
`tx_pcs_fifo_latency_pulse`	Output	`tx_clkout`	Latency pulse of TX PCS FIFO from latency measurement.
`rx_fifo_latency_pulse`	Output	`rx_coreclkin`	Latency pulse of RX core FIFO from latency measurement.
`rx_pcs_fifo_latency_pulse;`	Output	`rx_clkout`	Latency pulse of RX PCS FIFO from latency measurement.

Related Information

- Special TX PCS Reset Release Sequence on page 327
- How to Enable Low Latency in Basic (Enhanced PCS) on page 146
- Transceiver PHY PCS-to-Core Interface Reference Port Mapping on page 86

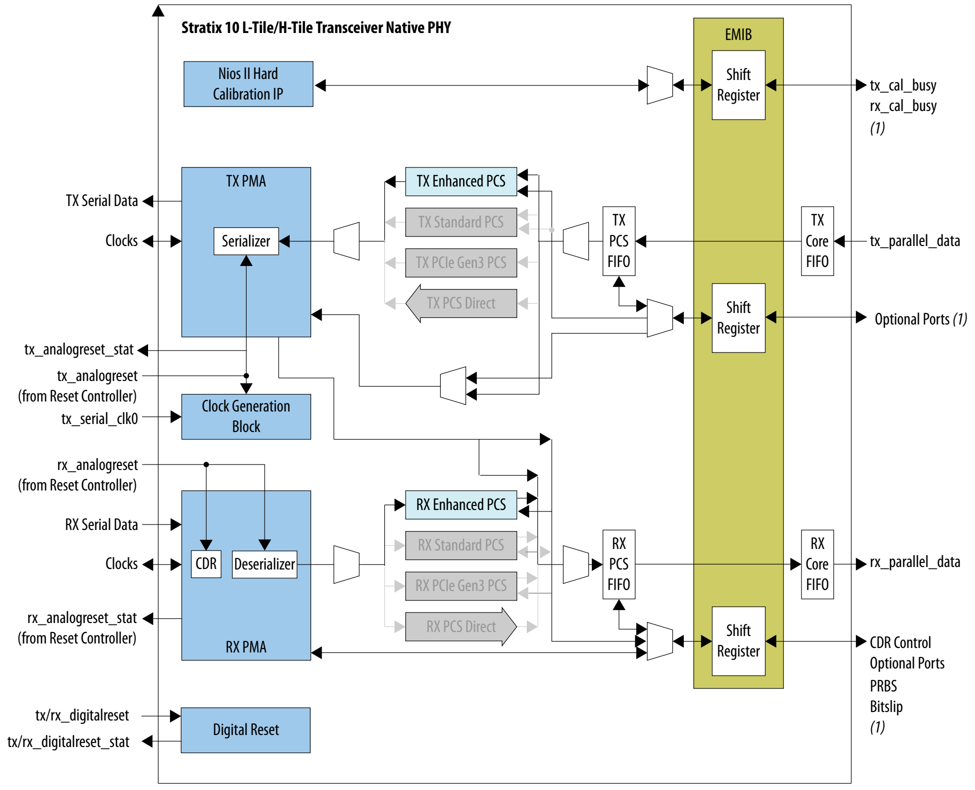
(15) For a detailed description of FSR and SSR signals, please go to the *Asynchronous Data Transfer* section.

- Enhanced PCS TX and RX Control Ports on page 76
- Asynchronous Data Transfer on page 144

2.3.14. Enhanced PCS Ports

Figure 26. Enhanced PCS Interfaces

The labeled inputs and outputs to the PMA and PCS modules represent buses, not individual signals.



Note:
1. The number of ports reflected may be greater than shown. Refer to the port description table for enhanced PCS, standard PCS, PCS direct, and PCI Express PIPE interface.

In the following tables, the variables represent these parameters:

- <*n*>—The number of lanes
- <*d*>—The serialization factor
- <*s*>— The symbol size
- <*p*>—The number of PLLs

Table 59. Interlaken Frame Generator, Synchronizer, and CRC32

Name	Direction	Clock Domain	Description
`tx_enh_frame[<n>-1:0]`	Output	`tx_clkout`	Asserted for 2 or 3 parallel clock cycles to indicate the beginning of a new metaframe.
`tx_err_ins`	Input	`tx_coreclkin`	For the Interlaken protocol, you can use this bit to insert the synchronous header and CRC32 errors if you have turned on **Enable simplified data interface**. When asserted, the synchronous header for that cycle word is replaced with a corrupted one. A CRC32 error is also inserted if **Enable Interlaken TX CRC-32 generator error insertion** is turned on. The corrupted sync header is 2'b00 for a control word, and 2'b11 for a data word. For CRC32 error insertion, the word used for CRC calculation for that cycle is incorrectly inverted, causing an incorrect CRC32 in the Diagnostic Word of the Metaframe. Note that a synchronous header error and a CRC32 error cannot be created for the Framing Control Words because the Frame Control Words are created in the frame generator embedded in TX PCS. Both the synchronous header error and the CRC32 errors are inserted if the CRC-32 error insertion feature is enabled in the Transceiver Native PHY IP Core GUI.
`tx_dll_lock`	Output	`tx_clkout`	User should monitor this lock status when the TX Core FIFO is configured in Interlaken or Basic mode of operation. For `tx_dll_lock` timing diagrams, refer to the *Special TX PCS Reset Release Sequence* under *Resetting Transceiver Channels* chapter. .
`tx_enh_frame_diag_status[2<n>-1:0]`	Input	`tx_clkout`	Drives the lane status message contained in the framing layer diagnostic word (bits[33:32]). This message is inserted into the next diagnostic word generated by the frame generator block. This bus must be held constant for 5 clock cycles before and after the `tx_enh_frame` pulse. The following encodings are defined: • Bit[1]: When 1, indicates the lane is operational. When 0, indicates the lane is not operational. • Bit[0]: When 1, indicates the link is operational. When 0, indicates the link is not operational.
`tx_enh_frame_burst_en[<n>-1:0]`	Input	`tx_clkout`	If **Enable frame burst** is enabled, this port controls frame generator data reads from the TX FIFO to the frame generator. It is latched once at the beginning of each Metaframe. If the value of `tx_enh_frame_burst_en` is 0, the frame generator does not read data from the TX FIFO for current Metaframe. Instead, the frame generator inserts SKIP words as the payload of Metaframe. When `tx_enh_frame_burst_en` is 1, the frame generator reads data from the TX FIFO for the current Metaframe. This port must be held constant for 5 clock cycles before and after the `tx_enh_frame` pulse.
`rx_enh_frame[<n>-1:0]`	Output	`rx_clkout`	When asserted, indicates the beginning of a new received Metaframe. This signal is pulse stretched.
`rx_enh_frame_lock[<n>-1:0]`	Output	`rx_clkout` SSR(16)	When asserted, indicates the Frame Synchronizer state machine has achieved Metaframe delineation. This signal is pulse stretched.

continued...

Name	Direction	Clock Domain	Description
`rx_enh_frame_diag_status[2 <n>-1:0]`	Output	`rx_clkout` SSR(16)	Drives the lane status message contained in the framing layer diagnostic word (bits[33:32]). This signal is latched when a valid diagnostic word is received in the end of the Metaframe while the frame is locked. The following encodings are defined: • Bit[1]: When 1, indicates the lane is operational. When 0, indicates the lane is not operational. • Bit[0]: When 1, indicates the link is operational. When 0, indicates the link is not operational.
`rx_enh_crc32_err[<n>-1:0]`	Output	`rx_clkout` FSR(16)	When asserted, indicates a CRC error in the current Metaframe. Asserted at the end of current Metaframe. This signal gets asserted for 2 or 3 cycles.

Table 60. 10GBASE-R BER Checker

Name	Direction	Clock Domain	Description
`rx_enh_highber[<n>-1:0]`	Output	`rx_clkout` SSR(16)	When asserted, indicates a bit error rate that is greater than 10^{-4}. For the 10GBASE-R protocol, this BER rate occurs when there are at least 16 errors within 125 µs. This signal gets asserted for 2 to 3 clock cycles.
`rx_enh_highber_clr_cnt[<n>-1:0]`	Input	`rx_clkout` SSR(16)	When asserted, clears the internal counter that indicates the number of times the BER state machine has entered the BER_BAD_SH state.
`rx_enh_clr_errblk_count[<n>-1:0]` (10GBASE-R and FEC)	Input	`rx_clkout` SSR(16)	When asserted the error block counter resets to 0. Assertion of this signal clears the internal counter that counts the number of times the RX state machine has entered the RX_E state. In modes where the FEC block is enabled, the assertion of this signal resets the status counters within the RX FEC block.

Table 61. Block Synchronizer

Name	Direction	Clock Domain	Description
`rx_enh_blk_lock<n>-1:0]`	Output	`rx_clkout` SSR(16)	When asserted, indicates that block synchronizer has achieved block delineation. This signal is used for 10GBASE-R and Interlaken.

Table 62. Gearbox

Name	Direction	Clock Domain	Description
`rx_bitslip[<n>-1:0]`	Input	`rx_clkout` SSR(16)	The `rx_parallel_data` slips 1 bit for every positive edge of the `rx_bitslip` input. Keep the `rx_bitslip` pulse high for at least 200 ns and each pulse 400 ns apart to ensure the data is slipped. The maximum shift is < *pcswidth -1*> bits, so that if the PCS is 64 bits wide, you can shift 0-63 bits.
`tx_enh_bitslip[<n>-1:0]`	Input	`rx_clkout` SSR(16)	The value of this signal controls the number of bits to slip the `tx_parallel_data` before passing to the PMA.

(16) For a detailed description of FSR and SSR signals, please go to the *Asynchronous Data Transfer* section.

Table 63. KR-FEC

Name	Direction	Clock Domain	Description
`tx_enh_frame[<n>-1:0]`	Output	`Asynchronous`	Asynchronous status flag output of TX KR-FEC that signifies the beginning of generated KR FEC frame
`rx_enh_frame[<n>-1:0]`	Output	`rx_clkout` SSR[16]	Asynchronous status flag output of RX KR-FEC that signifies the beginning of received KR FEC frame
`rx_enh_frame_diag_status[2<n>-1:0]`	Output	`rx_clkout` SSR[16]	Asynchronous status flag output of RX KR-FEC that indicates the status of the current received frame. • 00: No error • 01: Correctable Error • 10: Un-correctale error • 11: Reset condition/pre-lock condition

Related Information

- ATX PLL IP Core - Parameters, Settings, and Ports on page 260
- CMU PLL IP Core - Parameters, Settings, and Ports on page 275
- fPLL IP Core - Parameters, Settings, and Ports on page 269
- Ports and Parameters on page 419
- Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP Interfaces on page 335
- Special TX PCS Reset Release Sequence on page 327
- Enhanced PCS TX and RX Control Ports on page 76
- Asynchronous Data Transfer on page 144

2.3.14.1. Enhanced PCS TX and RX Control Ports

This section describes the `tx_control` and `rx_control` bit encodings for different protocol configurations.

When Enable simplified data interface is ON, all of the unused ports shown in the tables below, appear as a separate port. For example: It appears as `unused_tx_control`/ `unused_rx_control` port.

Note: When using double rate transfer, refer to the *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* section.

Enhanced PCS TX Control Port Bit Encodings

Table 64. Bit Encodings for Interlaken

Name	Bit	Functionality	Description
`tx_control`	[1:0]	Synchronous header	The value 2'b01 indicates a data word. The value 2'b10 indicates a control word.
	[2]	Inversion control	A logic low indicates that the built-in disparity generator block in the Enhanced PCS maintains the Interlaken running disparity.
	[7:3]	Unused	
	[8]	Insert synchronous header error or CRC32	You can use this bit to insert synchronous header error or CRC32 errors. The functionality is similar to `tx_err_ins`. Refer to `tx_err_ins` signal description in *Interlaken Frame Generator, Synchronizer and CRC32* table for more details.

Table 65. Bit Encodings for 10GBASE-R , 10GBASE-R 1588, 10GBASE-R with KR FEC

Name	Bit	Functionality
`tx_control`	[0]	XGMII control signal for `parallel_data[7:0]`
	[1]	XGMII control signal for `parallel_data[15:8]`
	[2]	XGMII control signal for `parallel_data[23:16]`
	[3]	XGMII control signal for `parallel_data[31:24]`
	[4]	XGMII control signal for `parallel_data[39:32]`
	[5]	XGMII control signal for `parallel_data[47:40]`
	[6]	XGMII control signal for `parallel_data[55:48]`
	[7]	XGMII control signal for `parallel_data[63:56]`
	[8]	Unused

Table 66. Bit Encodings for Basic (Enhanced PCS) with 66-bit word, Basic with KR FEC, 40GBASE-R with KR FEC

For Basic (Enhanced PCS) with 66-bit word, Basic with KR FEC, 40GBASE-R with KR FEC, the total word length is 66-bit with 64-bit data and 2-bit synchronous header.

Name	Bit	Functionality	Description
`tx_control`	[1:0]	Synchronous header	The value 2'b01 indicates a data word. The value 2'b10 indicates a control word.
	[8:2]	Unused	

Table 67. Bit Encodings for Basic (Enhanced PCS) with 67-bit word

In this case, the total word length is 67-bit with 64-bit data and 2-bit synchronous header and inversion bit for disparity control.

Name	Bit	Functionality	Description
`tx_control`	[1:0]	Synchronous header	The value 2'b01 indicates a data word. The value 2'b10 indicates a control word.
	[2]	Inversion control	A logic low indicates that built-in disparity generator block in the Enhanced PCS maintains the running disparity.

Enhanced PCS RX Control Port Bit Encodings

Table 68. Bit Encodings for Interlaken

Name	Bit	Functionality	Description
rx_control	[1:0]	Synchronous header	The value 2'b01 indicates a data word. The value 2'b10 indicates a control word.
	[2]	Inversion control	A logic low indicates that the built-in disparity generator block in the Enhanced PCS maintains the Interlaken running disparity. In the current implementation, this bit is always tied logic low (1'b0).
	[3]	Payload word location	A logic high (1'b1) indicates the payload word location in a metaframe.
	[4]	Synchronization word location	A logic high (1'b1) indicates the synchronization word location in a metaframe.
	[5]	Scrambler state word location	A logic high (1'b1) indicates the scrambler word location in a metaframe.
	[6]	SKIP word location	A logic high (1'b1) indicates the SKIP word location in a metaframe.
	[7]	Diagnostic word location	A logic high (1'b1) indicates the diagnostic word location in a metaframe.
	[8]	Synchronization header error, metaframe error, or CRC32 error status	A logic high (1'b1) indicates synchronization header error, metaframe error, or CRC32 error status.
	[9]	Block lock and frame lock status	A logic high (1'b1) indicates that block lock and frame lock have been achieved.

Table 69. Bit Encodings for 10GBASE-R , 10GBASE-R 1588, 10GBASE-R with KR FEC

Name	Bit	Functionality
rx_control	[0]	XGMII control signal for parallel_data[7:0]
	[1]	XGMII control signal for parallel_data[15:8]
	[2]	XGMII control signal for parallel_data[23:16]
	[3]	XGMII control signal for parallel_data[31:24]
	[4]	XGMII control signal for parallel_data[39:32]
	[5]	XGMII control signal for parallel_data[47:40]
	[6]	XGMII control signal for parallel_data[55:48]
	[7]	XGMII control signal for parallel_data[63:56]
	[9:8]	Unused

Table 70. Bit Encodings for Basic (Enhanced PCS) with 66-bit word, Basic with KR FEC, 40GBASE-R with KR FEC

For Basic (Enhanced PCS) with 66-bit word, Basic with KR FEC, 40GBASE-R with KR FEC, the total word length is 66-bit with 64-bit data and 2-bit synchronous header.

Name	Bit	Functionality	Description
`rx_control`	[1:0]	Synchronous header	The value 2'b01 indicates a data word. The value 2'b10 indicates a control word.
	[7:2]	Unused	
	[9:8]	Unused	

Table 71. Bit Encodings for Basic (Enhanced PCS) with 67-bit word

In this case, the total word length is 67-bit with 64-bit data and 2-bit synchronous header and inversion bit for disparity control.

Name	Bit	Functionality	Description
`rx_control`	[1:0]	Synchronous header	The value 2'b01 indicates a data word. The value 2'b10 indicates a control word.
	[2]	Inversion control	A logic low indicates that built-in disparity generator block in the Enhanced PCS maintains the running disparity.

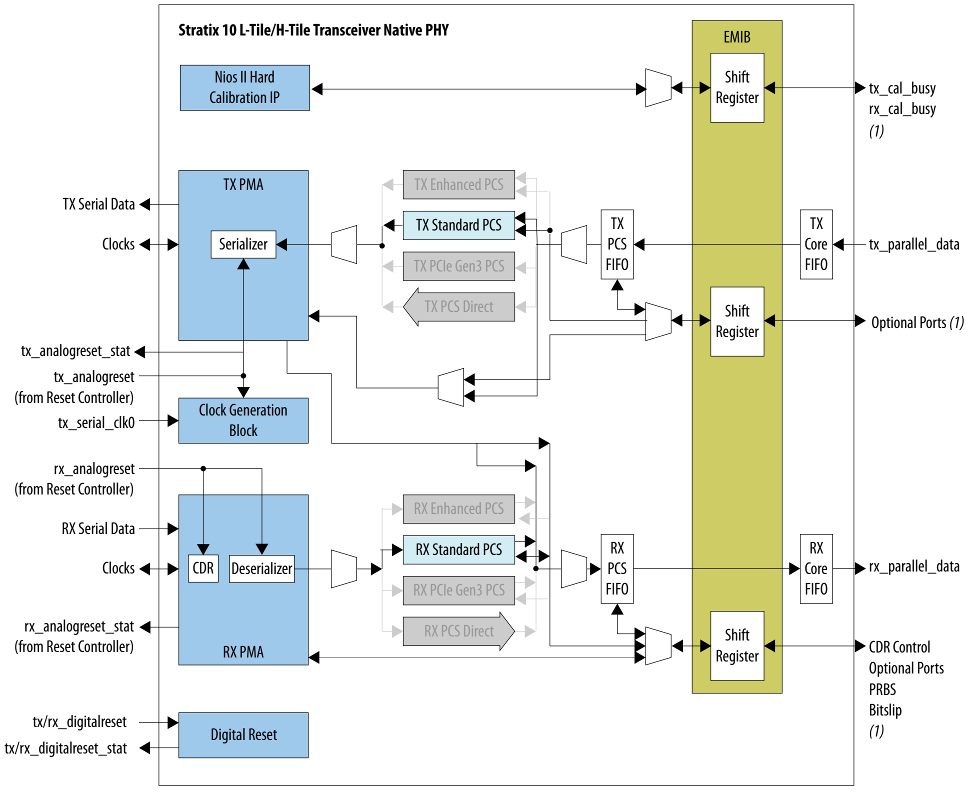
Related Information

- Enhanced PCS Ports on page 73
- Transceiver PHY PCS-to-Core Interface Reference Port Mapping on page 86

2.3.15. Standard PCS Ports

Figure 27. Transceiver Channel using the Standard PCS Ports

Standard PCS ports appear if you have selected either one of the transceiver configuration modes that use the Standard PCS .



Note:
1. The number of ports reflected may be greater than shown. Refer to the port description table for enhanced PCS, standard PCS, PCS direct, and PCI Express PIPE interface.

In the following tables, the variables represent these parameters:

- *<n>*—The number of lanes
- *<w>*—The width of the interface
- *<d>*—The serialization factor
- *<s>*— The symbol size
- *<p>*—The number of PLLs

Table 72. Rate Match FIFO

Name	Direction	Clock Domain	Description
`rx_std_rmfifo_full[<n>-1:0]`	Output	Asynchronous SSR[17]	Rate match FIFO full flag. When asserted the rate match FIFO is full. You must synchronize this signal. This port is only used for GigE mode.

continued...

Send Feedback

Name	Direction	Clock Domain	Description
rx_std_rmfifo_empty[<n>-1:0]	Output	Asynchronous SSR[17]	Rate match FIFO empty flag. When asserted, match FIFO is empty. You must synchronize this signal. This port is only used for GigE mode.
rx_rmfifostatus[<2*n>-1:0]	Output	Asynchronous	Indicates FIFO status. The following encodings are defined: • 2'b00: Normal operation • 2'b01: Deletion, `rx_std_rmfifo_full = 1` • 2'b10: Insertion, `rx_std_rmfifo_empty = 1` • 2'b11: Full. If simplified data interface is disabled, `rx_rmfifostatus` is a part of `rx_parallel_data`. For most configurations, `rx_rmfifostatus` corresponds to `rx_parallel_data[14:13]`. Refer to section *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* to identify the port mappings to `rx_parallel_data` for your specific transceiver configurations.

Table 73. 8B/10B Encoder and Decoder

Name	Direction	Clock Domain	Description
tx_datak	Input	tx_clkout	`tx_datak` is exposed if 8B/10B enabled and simplified data interface is set.When 1, indicates that the 8B/10B encoded word of tx_parallel_data is control. When 0, indicates that the 8B/10B encoded word of `tx_parallel_data` is data. For most configurations with simplified data interface disabled, `tx_datak` corresponds to `tx_parallel_data[8]`. For PMA width of 10-bit with double rate transfer mode disabled or PMA width of 20-bit with double rate transfer mode enabled and the Byte Serializer is enabled, `tx_datak` corresponds to `tx_parallel_data[8]` and `tx_parallel_data[19]`. For PMA width of 20-bit with double rate transfer mode is disabled and Byte Serializer enabled, `tx_datak` corresponds to `tx_parallel_data[8]`, `tx_parallel_data[19]`, `tx_parallel_data[48]`, and `tx_parallel_data[59]`. If simplified data interface is disabled, `rx_rmfifostatus` is a part of `rx_parallel_data`. For most configurations, `rx_rmfifostatus[1:0]` corresponds to `rx_parallel_data[14:13]`. Refer to section *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* to identify the port mappings to `rx_parallel_data` for your specific transceiver configurations.
tx_forcedisp[<n>(<w>/<s>-1:0]	Input	Asynchronous	`tx_forcedisp` is only exposed if 8B/10B, 8B/10B disparity control, and simplified data interface has been enabled. This signal allows you to force the disparity of the 8B/10B encoder. When "1", forces the disparity of the output data to the value driven on `tx_dispval`. When "0", the current running disparity continues. For most configurations with simplified data interface disabled, `tx_forcedisp` corresponds to `tx_parallel_data[9]`.

continued...

(17) For a detailed description of FSR and SSR signals, please go to the *Asynchronous Data Transfer* section.

Name	Direction	Clock Domain	Description
			For PMA width of 10-bit with double rate transfer mode disabled or PMA width of 20-bit with double rate transfer mode enabled and the Byte Serializer is enabled, `tx_forcedisp` corresponds to `tx_parallel_data[9]` and `tx_parallel_data[20]`. For PMA width of 20-bit with double rate transfer mode is disabled and Byte Serializer enabled, `tx_forcedisp` corresponds to `tx_parallel_data[9]`, `tx_parallel_data[20]`, `tx_parallel_data[49]`, and `tx_parallel_data[60]`. If simplified data interface is disabled, `rx_rmfifostatus` is a part of `rx_parallel_data`. For most configurations, `rx_rmfifostatus` corresponds to `rx_parallel_data[14:13]`. Refer to section *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* to identify the port mappings to `rx_parallel_data` for your specific transceiver configurations.
`tx_dispval[<n>(<w>/<s>-1:0]`	Input	Asynchronous	`tx_dispval` is exposed if 8B/10B, 8B/10B disparity control, and simplified data interface has been enabled. Specifies the disparity of the data. When 0, indicates positive disparity, and when 1, indicates negative disparity. For most configurations with simplified data interface disabled, `tx_dispval` corresponds to `tx_parallel_data[10]`. For PMA width of 10-bit with double rate transfer mode disabled or PMA width of 20-bit with double rate transfer mode enabled and the Byte Serializer is enabled, `tx_forcedisp` corresponds to `tx_parallel_data[10]` and `tx_parallel_data[21]`. For PMA width of 20-bit with double rate transfer mode is disabled and Byte Serializer enabled, `tx_dispval` corresponds to `tx_parallel_data[10]`, `tx_parallel_data[21]`, `tx_parallel_data[50]`, and `tx_parallel_data[61]`. If simplified data interface is disabled, `rx_rmfifostatus` is a part of `rx_parallel_data`. For most configurations, `rx_rmfifostatus` corresponds to `rx_parallel_data[14:13]`. Refer to section *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* to identify the port mappings to `rx_parallel_data` for your specific transceiver configurations.
`rx_datak[<n><w>/<s>-1:0]`	Output	`rx_clkout`	`rx_datak` is exposed if 8B/10B is enabled and simplified data interface is set. When 1, indicates that the 8B/10B decoded word of rx_parallel_data is control. When 0, indicates that the 8B/10B decoded word of `rx_parallel_data` is data. For most configurations with simplified data interface disabled, `rx_datak` corresponds to `rx_parallel_data[8]`. For PMA width of 10-bit with double rate transfer mode disabled or PMA width of 20-bit with double rate transfer mode enabled and the Byte Serializer is enabled, `rx_datak` corresponds to `rx_parallel_data[8]` and `rx_parallel_data[24]`. For PMA width of 20-bit with double rate transfer mode is disabled and Byte Serializer enabled, `rx_datak` corresponds to `rx_parallel_data[8]`, `rx_parallel_data[24]`, `rx_parallel_data[48]`, and `tx_parallel_data[64]`.

continued...

Name	Direction	Clock Domain	Description
			If simplified data interface is disabled, `rx_rmfifostatus` is a part of `rx_parallel_data`. For most configurations, `rx_rmfifostatus` corresponds to `rx_parallel_data[14:13]`. Refer to section *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* to identify the port mappings to `rx_parallel_data` for your specific transceiver configurations.
`rx_errdetect[<n><w>/<s>-1:0]`	Output	Synchronous to the clock driving the read side of the FIFO (`rx_coreclkin` or `rx_clkout`)	When asserted, indicates a code group violation detected on the received code group. Used along with `rx_disperr` signal to differentiate between code group violation and disparity errors. The following encodings are defined for `rx_errdetect/rx_disperr`: • 2'b00: no error • 2'b10: code group violation • 2'b11: disparity error. For most configurations with simplified data interface disabled, `rx_errdetect` corresponds to `rx_parallel_data[9]`. For PMA width of 10-bit with double rate transfer mode disabled or PMA width of 20-bit with double rate transfer mode enabled and the Byte Deserializer is enabled, `rx_errdetect` corresponds to `rx_parallel_data[9]` and `rx_parallel_data[25]`. For PMA width of 20-bit with double rate transfer mode is disabled and Byte Deserializer enabled, `rx_errdetect` corresponds to `rx_parallel_data[9]`, `rx_parallel_data[25]`, `rx_parallel_data[49]`, and `rx_parallel_data[65]`. If simplified data interface is disabled, `rx_rmfifostatus` is a part of `rx_parallel_data`. For most configurations, `rx_rmfifostatus` corresponds to `rx_parallel_data[14:13]`. Refer to section *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* to identify the port mappings to `rx_parallel_data` for your specific transceiver configurations.
`rx_disperr[<n><w>/<s>-1:0]`	Output	Synchronous to the clock driving the read side of the FIFO (`rx_coreclkin` or `rx_clkout`)	When asserted, indicates a disparity error on the received code group. For most configurations with simplified data interface disabled, `rx_disperr` corresponds to `rx_parallel_data[11]`. For PMA width of 10-bit with double rate transfer mode disabled or PMA width of 20-bit with double rate transfer mode enabled and the Byte Deserializer is enabled, `rx_disperr` corresponds to `rx_parallel_data[11]` and `rx_parallel_data[27]`. For PMA width of 20-bit with double rate transfer mode is disabled and Byte Deserializer enabled, `rx_disperr` corresponds to `rx_parallel_data[11]`, `rx_parallel_data[27]`, `rx_parallel_data[51]`, and `rx_parallel_data[67]`. If simplified data interface is disabled, `rx_rmfifostatus` is a part of `rx_parallel_data`. For most configurations, `rx_rmfifostatus` corresponds to `rx_parallel_data[14:13]`. Refer to section *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* to identify the port mappings to `rx_parallel_data` for your specific transceiver configurations.

continued...

Name	Direction	Clock Domain	Description
rx_runningdisp[<n><w>/<s>-1:0]	Output	Synchronous to the clock driving the read side of the FIFO (rx_coreclkin or rx_clkout)	When high, indicates that rx_parallel_data was received with negative disparity. When low, indicates that rx_parallel_data was received with positive disparity. For most configurations with simplified data interface disabled, rx_runningdisp corresponds to rx_parallel_data[15]. For PMA width of 10-bit with double rate transfer mode disabled or PMA width of 20-bit with double rate transfer mode enabled and the Byte Deserializer is enabled, rx_runningdisp corresponds to rx_parallel_data[15] and rx_parallel_data[31]. For PMA width of 20-bit with double rate transfer mode is disabled and Byte Deserializer enabled, rx_runningdisp corresponds to rx_parallel_data[15], rx_parallel_data[31], rx_parallel_data[55], and rx_parallel_data[71]. If simplified data interface is disabled, rx_rmfifostatus is a part of rx_parallel_data. For most configurations, rx_rmfifostatus corresponds to rx_parallel_data[14:13]. Refer to section *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* to identify the port mappings to rx_parallel_data for your specific transceiver configurations.
rx_patterndetect[<n><w>/<s>-1:0]	Output	Asynchronous	When asserted, indicates that the programmed word alignment pattern has been detected in the current word boundary. Refer to "Word Alignment Using the Standard PCS" section for more details. For most configurations with simplified data interface disabled, rx_patterndetect corresponds to rx_parallel_data[12]. For PMA width of 10-bit with double rate transfer mode disabled or PMA width of 20-bit with double rate transfer mode enabled and the Byte Deserializer is enabled, rx_patterndetect corresponds to rx_parallel_data[12] and rx_parallel_data[28]. For PMA width of 20-bit with double rate transfer mode is disabled and Byte Deserializer enabled, rx_patterndetect corresponds to rx_parallel_data[12], rx_parallel_data[28], rx_parallel_data[52], and rx_parallel_data[68]. If simplified data interface is disabled, rx_rmfifostatus is a part of rx_parallel_data. For most configurations, rx_rmfifostatus corresponds to rx_parallel_data[14:13]. Refer to section *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* to identify the port mappings to rx_parallel_data for your specific transceiver configurations.
rx_syncstatus[<n><w>/<s>-1:0]	Output	Asynchronous	When asserted, indicates that the conditions required for synchronization are being met. Refer to "Word Alignment Using the Standard PCS" section for more details. rx_syncstatus is bus dependent on the width of the parallel data. For example, when the parallel data width is 32 bits, then rx_syncstatus is a 4 bit bus. The final expected value is 1'hf, indicating the control character is identified at the correct location in the 32 bit parallel word. For most configurations with simplified data interface disabled, rx_syncstatus corresponds to rx_parallel_data[10].

Name	Direction	Clock Domain	Description
			For PMA width of 10-bit with double rate transfer mode disabled or PMA width of 20-bit with double rate transfer mode enabled and the Byte Deserializer is enabled, `rx_syncstatus` corresponds to `rx_parallel_data[10]` and `rx_parallel_data[26]`. For PMA width of 20-bit with double rate transfer mode is disabled and Byte Deserializer enabled, `rx_syncstatus` corresponds to `rx_parallel_data[10]`, `rx_parallel_data[26]`, `rx_parallel_data[50]`, and `rx_parallel_data[66]`. If simplified data interface is disabled, `rx_rmfifostatus` is a part of `rx_parallel_data`. For most configurations, `rx_rmfifostatus` corresponds to `rx_parallel_data[14:13]`. Refer to section *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* to identify the port mappings to `rx_parallel_data` for your specific transceiver configurations.

Table 74. Word Aligner and Bitslip

Name	Direction	Clock Domain	Description
`tx_std_bitslipboundarysel[5 <n>-1:0]`	Input	Asynchronous SSR(17)	Bitslip boundary selection signal. Specifies the number of bits that the TX bit slipper must slip.
`rx_std_bitslipboundarysel[5 <n>-1:0]`	Output	Synchronous to `rx_clkout`	This port is used in deterministic latency word aligner mode. It reports the number of bits that the RX block slipped to achieve deterministic latency.
`rx_std_wa_patternalign[<n>-1:0]`	Input	Asynchronous SSR(17)	This port is enabled when you place the word aligner in manual mode. In manual mode, you align words by asserting `rx_std_wa_patternalign`. When the PCS-PMA Interface width is 10 bits, `rx_std_wa_patternalign` is level sensitive. For all the other PCS-PMA Interface widths, `rx_std_wa_patternalign` is positive edge sensitive. You can use this port only when the word aligner is configured in manual or deterministic latency mode. When the word aligner is in manual mode, and the PCS-PMA interface width is 10 bits, this is a level sensitive signal. In this case, the word aligner monitors the input data for the word alignment pattern, and updates the word boundary when it finds the alignment pattern. For all other PCS-PMA interface widths, this signal is edge sensitive.This signal is internally synchronized inside the PCS using the PCS parallel clock and should be asserted for at least 2 clock cycles to allow synchronization.
`rx_std_wa_a1a2size[<n>-1:0]`	Input	Asynchronous SSR(17)	Used for the SONET protocol. Assert when the A1 and A2 framing bytes must be detected. A1 and A2 are SONET framing alignment overhead bytes and are only used when the PMA data width is 8 or 16 bits. The 2 alignment markers valid status is captured in the 2 bit of `rx_std_wa_ala2size` signal. When both the markers are matched, then the value of the signal is 2'b11. If simplified data interface is disabled, `rx_rmfifostatus` is a part of `rx_parallel_data`. For most configurations, `rx_rmfifostatus` corresponds to `rx_parallel_data[14:13]`. Refer to section *Transceiver*

continued...

Name	Direction	Clock Domain	Description
			PHY PCS-to-Core Interface Reference Port Mapping to identify the port mappings to `rx_parallel_data` for your specific transceiver configurations.
`rx_bitslip[<n>-1:0]`	Input	Asynchronous SSR[17]	Used when word aligner mode is bitslip mode. When the Word Aligner is in either Manual (FPGA Fabric width controlled), Synchronous State Machine or Deterministic Latency ,the `rx_bitslip signal` is not valid and should be tied to 0. For every rising edge of the `rx_std_bitslip` signal, the word boundary is shifted by 1 bit. Each bitslip removes the earliest received bit from the received data.

Table 75. Bit Reversal and Polarity Inversion

Name	Direction	Clock Domain	Description
`rx_std_byterev_ena[<n>-1:0]`	Input	Asynchronous SSR[17]	This control signal is available when the PMA width is 16 or 20 bits. When asserted, enables byte reversal on the RX interface. Use this when the MSB and LSB byte order of data packet from transmitter is inverted order than receiver.
`rx_std_bitrev_ena[<n>-1:0]`	Input	Asynchronous SSR[17]	When asserted, enables bit reversal on the RX interface. Bit order may be reversed if external transmission circuitry transmits the most significant bit first. When enabled, the receiver receives all words in the reverse order. The bit reversal circuitry operates on the output of the word aligner.
`tx_polinv[<n>-1:0]`	Input	Asynchronous SSR[17]	When asserted, the TX polarity bit is inverted. Only active when TX bit polarity inversion is enabled.
`rx_polinv[<n>-1:0]`	Input	Asynchronous SSR[17]	When asserted, the RX polarity bit is inverted. Only active when RX bit polarity inversion is enabled.

Related Information

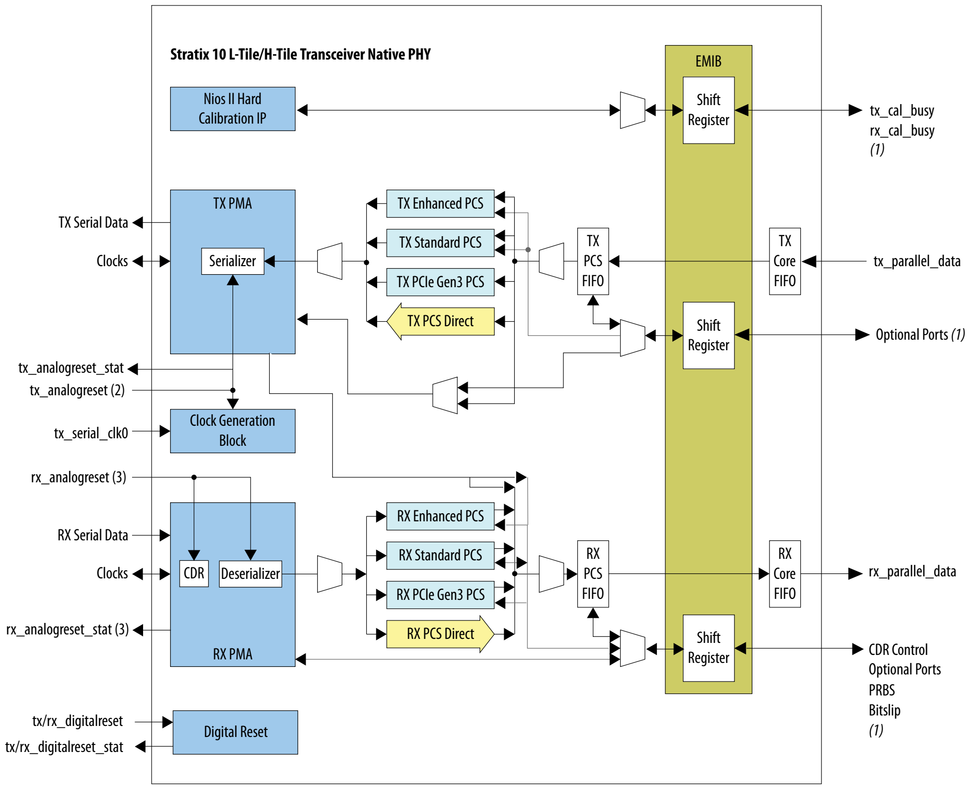
- ATX PLL IP Core - Parameters, Settings, and Ports on page 260
- fPLL IP Core - Parameters, Settings, and Ports on page 269
- CMU PLL IP Core - Parameters, Settings, and Ports on page 275
- Ports and Parameters on page 419
- Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP Interfaces on page 335
- Transceiver PHY PCS-to-Core Interface Reference Port Mapping on page 86
- Asynchronous Data Transfer on page 144
- Word Alignment Using the Standard PCS on page 111
- Intel Stratix 10 (L/H-Tile) Word Aligner Bitslip Calculator
  Use this tool to calculate the number of slips you require to achieve alignment based on the word alignment pattern and length.

2.3.16. Transceiver PHY PCS-to-Core Interface Reference Port Mapping

This section lists the following tables for the PCS-to-Core port interface mappings of all the supported configurations for the Enhanced PCS, Standard PCS, and PCS-Direct configurations when Simplified Data Interface is disabled or unavailable. For the port interface mappings for PCIe Gen1-Gen3, refer to the *PCIe Express* chapter. Refer to these tables when mapping certain port functions to `tx_parallel_data` and `rx_parallel_data`. The Intel Stratix 10L-/ H-Tile Transceiver PHY PCS-to-Core interface has a maximum 80-bit width parallel data bus per channel which includes data, control, word marker, PIPE, and PMA and PCS status ports depending on the PCS/datapath enabled and transceiver configurations.

Note: When **Simplified Data Interface** is enabled, some ports go through the slow shift registers (SSR) or fast shift registers (FSR). Refer to the *Asynchronous Data Transfer* section for more details about FSR and SSR.

Figure 28. PCS-Core Port Interface



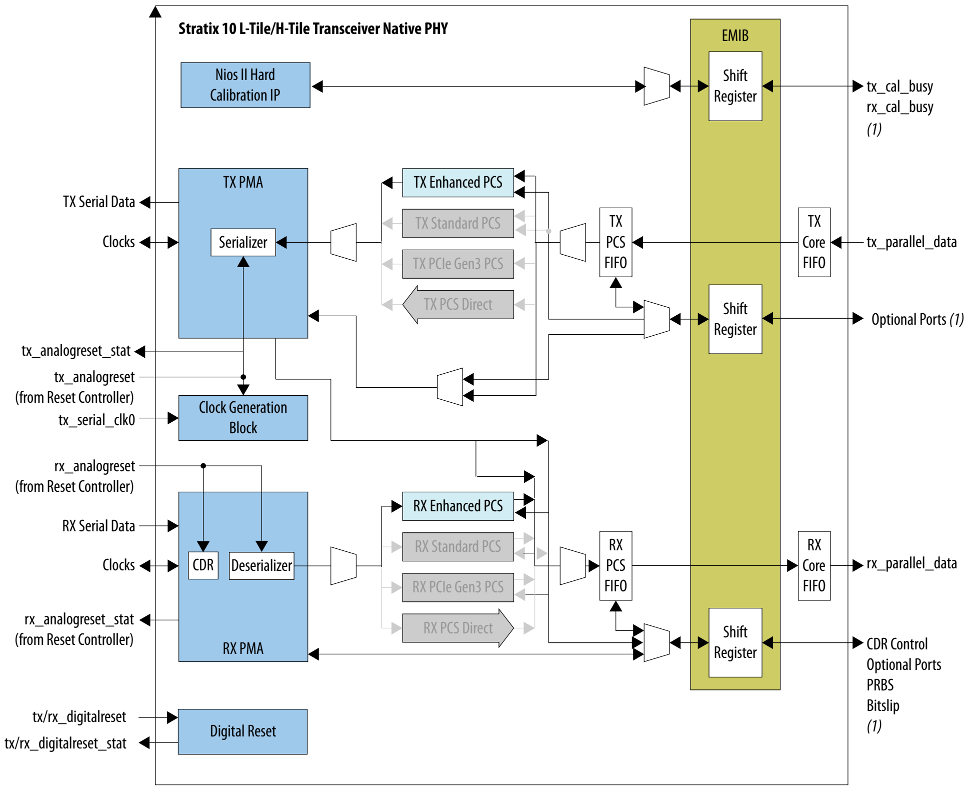
Note:
1. The number of ports reflected may be greater than shown. Refer to the port description table for enhanced PCS, standard PCS, PCS direct, and PCI Express PIPE interface.
2. This signal goes into the Reset Controller.
3. This signal comes from the Reset Controller.

Related Information

- Standard PCS Ports on page 80
- PCS-Core Interface Ports on page 66
- PCI Express (PIPE) on page 164
- PMA, Calibration, and Reset Ports on page 63
- Enhanced PCS Ports on page 73
- Asynchronous Data Transfer on page 144

2.3.16.1. PCS-Core Interface Ports: Enhanced PCS

Figure 29. PCS-Core Interface Port: Enhanced PCS



Note:
1. The number of ports reflected may be greater than shown. Refer to the port description table for enhanced PCS, standard PCS, PCS direct, and PCI Express PIPE interface.

Note: In the following table, the `tx_parallel_data` and `rx_parallel_data` mappings shown are for a single channel. To determine the mappings for multi-channel designs, the user must scale the single channel mappings with the appropriate channel multipliers. For example, `data[31:0]` maps to `tx_parallel_data[31:0]` and `rx_parallel_data[31:0]` for single channel designs. For multi-channel designs, `data[31:0]` for every channel would map to `tx_parallel_data[<n-1>80+31:<n-1>80]` and `rx_parallel_data[<n-1>80+31:<n-1>80]`, where <n> is the channel number.

Table 76. Simplified Data Interface=Disabled, Double-Rate Transfer=Disabled

TX Port Function	TX Port	RX Port Function	RX Port
Configuration-1, PMA Width-32, FPGA Fabric width-32			
`data[31:0]`	`tx_parallel_data[31:0]`	`data[31:0]`	`rx_parallel_data[31:0]`
`tx_fifo_wr_en`	`tx_parallel_data[79]`	`rx_prbs_err`	`rx_parallel_data[35]`
		`rx_prbs_done`	`rx_parallel_data[36]`
		`rx_data_valid`	`rx_parallel_data[79]`

continued...

TX Port Function	TX Port	RX Port Function	RX Port
Configuration-2, PMA Width-40, FPGA Fabric width-40			
data[39:0]	tx_parallel_data[39:0]	data[39:0]	rx_parallel_data[39:0]
tx_fifo_wr_en	tx_parallel_data[79]	rx_data_valid	rx_parallel_data[79]
Configuration-3, PMA Width-32/40/64, FPGA Fabric width-64/66/67			
data[31:0]	tx_parallel_data[31:0]	data[31:0]	rx_parallel_data[31:0]
data[63:32]	tx_parallel_data[71:40]	data[63:32]	rx_parallel_data[71:40]
tx_control[3:0]	tx_parallel_data[35:32]	rx_control[3:0]	rx_parallel_data[35:32]
tx_control[8:4]	tx_parallel_data[76:72]	rx_control[9:4]	rx_parallel_data[77:72]
tx_enh_data_valid	tx_parallel_data[36]	rx_enh_data_valid	rx_parallel_data[36]
tx_fifo_wr_en	tx_parallel_data[79]	rx_data_valid	rx_parallel_data[79]

Note: In the following table, the tx_parallel_data and rx_parallel_data mappings shown are for a single channel. To determine the mappings for multi-channel designs, the user must scale the single channel mappings with the appropriate channel multipliers. For example, data[31:0] maps to tx_parallel_data[31:0] and rx_parallel_data[31:0] for single channel designs. For multi-channel designs, data[31:0] for every channel would map to tx_parallel_data[<n-1>80+31:<n-1>80] and rx_parallel_data[<n-1>80+31:<n-1>80], where <n> is the channel number.

Table 77. Simplified Data Interface=Disabled, Double-Rate Transfer=Enabled

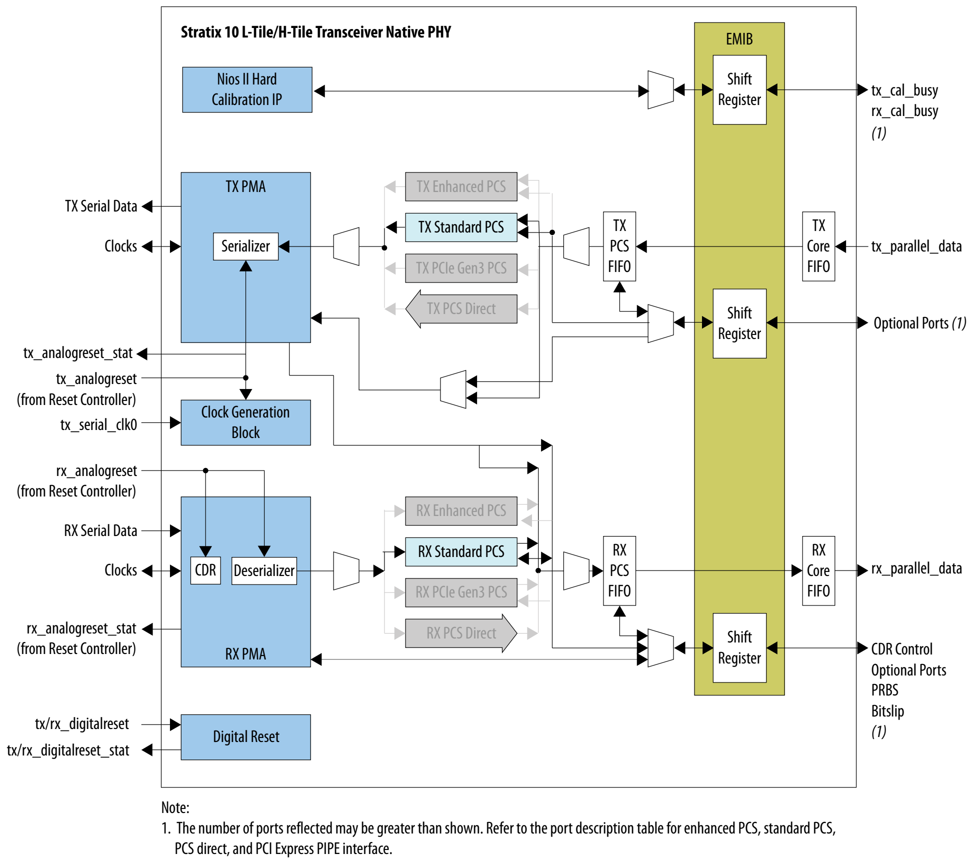
TX Port Function	TX Port	RX Port Function	RX Port
Configuration-4, PMA Width-32, FPGA Fabric width-16			
data[15:0]	tx_parallel_data[15:0] (lower word)	data[15:0]	rx_parallel_data[15:0] (lower word)
data[31:16]	tx_parallel_data[15:0] (upper word)	data[31:16]	rx_parallel_data[15:0] (upper word)
tx_word_marking_bit=0	tx_parallel_data[19] (lower word)	rx_word_marking_bit=0	rx_parallel_data[39] (upper word)
tx_word_marking_bit=1	tx_parallel_data[19] (upper word)	rx_word_marking_bit=1	rx_parallel_data[39] (lower word)
tx_fifo_wr_en	tx_parallel_data[79] (lower and upper word)	rx_data_valid	rx_parallel_data[79] (lower and upper word)
Configuration-5, PMA Width-40, FPGA Fabric width-20			
data[19:0]	tx_parallel_data[19:0] (lower word)	data[19:0]	rx_parallel_data[19:0] (lower word)
data[39:20]	tx_parallel_data[19:0] (upper word)	data[39:20]	rx_parallel_data[19:0] (upper word)
tx_word_marking_bit=0	tx_parallel_data[39] (lower word)	rx_word_marking_bit=0	rx_parallel_data[39] (lower word)
tx_word_marking_bit=1	tx_parallel_data[39] (upper word)	rx_word_marking_bit=1	rx_parallel_data[39] (upper word)

continued...

TX Port Function	TX Port	RX Port Function	RX Port
`tx_fifo_wr_en`	`tx_parallel_data[79]` (lower and upper word)	`rx_data_valid`	`rx_parallel_data[79]` (lower and upper word)
Configuration-6, PMA Width-32/40/64, FPGA Fabric width-32			
`data[31:0]`	`tx_parallel_data[31:0]` (lower word)	`data[31:0]`	`rx_parallel_data[31:0]` (lower word)
`data[63:32]`	`tx_parallel_data[31:0]` (upper word)	`data[63:32]`	`rx_parallel_data[31:0]` (upper word)
`tx_control[3:0]`	`tx_parallel_data[35:32]` (lower word)	`rx_control[3:0]`	`rx_parallel_data[35:32]` (lower word)
`tx_control[8:4]`	`tx_parallel_data[36:32]` (upper word)	`rx_control[9:4]`	`rx_parallel_data[37:32]` (upper word)
`tx_word_marking_bit=0`	`tx_parallel_data[39]` (lower word)	`rx_word_marking_bit=0`	`rx_parallel_data[39]` (lower word)
`tx_word_marking_bit=1`	`tx_parallel_data[39]` (upper word)	`rx_word_marking_bit=1`	`rx_parallel_data[39]` (upper word)
`tx_enh_data_valid`	`tx_parallel_data[36]` (lower and upper word)	`rx_enh_data_valid`	`rx_parallel_data[36]` (lower and upper word)
`tx_fifo_wr_en`	`tx_parallel_data[79]` (lower and upper word)	`rx_data_valid`	`rx_parallel_data[79]` (lower and upper word)

2.3.16.2. PCS-Core Interface Ports: Standard PCS

Figure 30. PCS-Core Interface Ports: Standard PCS



Note:
1. The number of ports reflected may be greater than shown. Refer to the port description table for enhanced PCS, standard PCS, PCS direct, and PCI Express PIPE interface.

Note: In the following table, the `tx_parallel_data` and `rx_parallel_data` mappings shown are for a single channel. To determine the mappings for multi-channel designs, the user must scale the single channel mappings with the appropriate channel multipliers. For example, `data[31:0]` maps to `tx_parallel_data[31:0]` and `rx_parallel_data[31:0]` for single channel designs. For multi-channel designs, `data[31:0]` for every channel would map to `tx_parallel_data[<n-1>80+31:<n-1>80]` and `rx_parallel_data[<n-1>80+31:<n-1>80]`, where <n> is the channel number.

Table 78. Simplified Data Interface=Disabled, Double-Rate Transfer=Disabled

TX Port Function	TX Port	RX Port Function	RX Port
Configuration-7, PMA Width-8, 8B10B-NA, Byte Serializer-Disabled			
data[7:0]	tx_parallel_data[7:0]	data[7:0]	rx_parallel_data[7:0]
		rx_std_wa_a1a2size	rx_parallel_data[8]
		rx_syncstatus	rx_parallel_data[10]
		rx_patterndetect	rx_parallel_data[12]

continued...

TX Port Function	TX Port	RX Port Function	RX Port
		rx_data_valid	rx_parallel_data[79]
Configuration-8, PMA Width-8, 8B10B-NA, Byte Serializer-Enabled			
data[7:0]	tx_parallel_data[7:0]	data[7:0]	rx_parallel_data[7:0]
data[15:8]	tx_parallel_data[18:11]	data[15:8]	rx_parallel_data[23:16]
		rx_std_wa_a1a2size	rx_parallel_data[8], [24]
		rx_syncstatus	rx_parallel_data[10], [26]
		rx_patterndetect	rx_parallel_data[12], [28]
		rx_data_valid	rx_parallel_data[79]
Configuration-9, PMA Width-10, 8B10B-Disabled, Byte Serializer-Disabled			
data[9:0]	tx_parallel_data[9:0]	data[9:0]	rx_parallel_data[9:0]
		rx_syncstatus	rx_parallel_data[10]
		rx_disperr	rx_parallel_data[11]
		rx_patterndetect	rx_parallel_data[12]
		rx_rmfifostatus[0]	rx_parallel_data[13]
		rx_rmfifostatus[1]	rx_parallel_data[14]
		rx_runningdisp	rx_parallel_data[15]
		rx_data_valid	rx_parallel_data[79]
Configuration-10, PMA Width-10, 8B10B-Disabled, Byte Serializer-Enabled			
data[9:0]	tx_parallel_data[9:0]	data[9:0]	rx_parallel_data[9:0]
data[19:10]	tx_parallel_data[20:11]	data[25:16]	rx_parallel_data[25:16]
		rx_syncstatus	rx_parallel_data[10], [26]
		rx_disperr	rx_parallel_data[11], [27]
		rx_patterndetect	rx_parallel_data[12], [28]
		rx_rmfifostatus[0]	rx_parallel_data[13], [29]
		rx_rmfifostatus[1]	rx_parallel_data[14], [30]
		rx_runningdisp	rx_parallel_data[15], [31]
		rx_data_valid	rx_parallel_data[79]
Configuration-11, PMA Width-10, 8B10B-Enabled, Byte Serializer-Disabled			
data[7:0]	tx_parallel_data[7:0]	data[7:0]	rx_parallel_data[7:0]
tx_datak	tx_parallel_data[8]	rx_datak	rx_parallel_data[8]
tx_forcedisp	tx_parallel_data[9]	rx_errdetect	rx_parallel_data[9]
tx_dispval	tx_parallel_data[10]	rx_syncstatus	rx_parallel_data[10]
		rx_disperr	rx_parallel_data[11]
		rx_patterndetect	rx_parallel_data[12]

continued...

TX Port Function	TX Port	RX Port Function	RX Port
		rx_rmfifostatus[0]	rx_parallel_data[13]
		rx_rmfifostatus[1]	rx_parallel_data[14]
		rx_runningdisp	rx_parallel_data[15]
		rx_data_valid	rx_parallel_data[79]
Configuration-12, PMA Width-10, 8B10B-Enabled, Byte Serializer-Enabled			
data[7:0]	tx_parallel_data[7:0]	data[7:0]	rx_parallel_data[7:0]
data[15:8]	tx_parallel_data[18:11]	data[15:8]	rx_parallel_data[23:16]
tx_datak	tx_parallel_data[8], [19]	rx_datak	rx_parallel_data[8], [24]
tx_forcedisp	tx_parallel_data[9], [20]	rx_errdetect	rx_parallel_data[9], [25]
tx_dispval	tx_parallel_data[10], [21]	rx_syncstatus	rx_parallel_data[10], [26]
		rx_disperr	rx_parallel_data[11], [27]
		rx_patterndetect	rx_parallel_data[12], [28]
		rx_rmfifostatus[0]	rx_parallel_data[13], [29]
		rx_rmfifostatus[1]	rx_parallel_data[14], [30]
		rx_runningdisp	rx_parallel_data[15], [31]
		rx_data_valid	rx_parallel_data[79]
Configuration-13, PMA Width-16, 8B10B-NA, Byte Serializer-Disabled			
data[7:0]	tx_parallel_data[7:0]	data[7:0]	rx_parallel_data[7:0]
data[15:8]	tx_parallel_data[18:11]	data[15:8]	rx_parallel_data[23:16]
		rx_std_wa_a1a2size	rx_parallel_data[8], [24]
		rx_syncstatus	rx_parallel_data[10], [26]
		rx_patterndetect	rx_parallel_data[12], [28]
		rx_data_valid	rx_parallel_data[79]
Configuration-14, PMA Width-16, 8B10B-NA, Byte Serializer-Enabled			
data[7:0]	tx_parallel_data[7:0]	data[7:0]	rx_parallel_data[7:0]
data[15:8]	tx_parallel_data[18:11]	data[15:8]	rx_parallel_data[23:16]
data[23:16]	tx_parallel_data[47:40]	data[23:16]	rx_parallel_data[47:40]
data[31:24]	tx_parallel_data[58:51]	data[31:24]	rx_parallel_data[63:56]
		rx_std_wa_a1a2size	rx_parallel_data[8], [24], [48], [64]
		rx_syncstatus	rx_parallel_data[10], [26], [50], [66]
		rx_patterndetect	rx_parallel_data[12], [28], [52], [68]
		rx_data_valid	rx_parallel_data[79]
Configuration-15, PMA Width-20, 8B10B-Disabled, Byte Serializer-Disabled			

continued...

TX Port Function	TX Port	RX Port Function	RX Port
data[9:0]	tx_parallel_data[9:0]	data[9:0]	rx_parallel_data[9:0]
data[19:10]	tx_parallel_data[20:11]	data[19:10]	rx_parallel_data[25:16]
		rx_syncstatus	rx_parallel_data[10], [26]
		rx_disperr	rx_parallel_data[11], [27]
		rx_patterndetect	rx_parallel_data[12], [28]
		rx_rmfifostatus[0]	rx_parallel_data[13], [29]
		rx_rmfifostatus[1]	rx_parallel_data[14], [30]
		rx_runningdisp	rx_parallel_data[15], [31]
		rx_data_valid	rx_parallel_data[79]
Configuration-16, PMA Width-20, 8B10B-Disabled, Byte Serializer-Enabled			
data[9:0]	tx_parallel_data[9:0]	data[9:0]	rx_parallel_data[9:0]
data[20:11]	tx_parallel_data[20:11]	data[19:10]	rx_parallel_data[25:16]
data[49:40]	tx_parallel_data[49:40]	data[29:20]	rx_parallel_data[49:40]
data[60:51]	tx_parallel_data[60:51]	data[39:30]	rx_parallel_data[65:56]
		rx_syncstatus	rx_parallel_data[10], [26], [50], [66]
		rx_disperr	rx_parallel_data[11], [27], [51], [67]
		rx_patterndetect	rx_parallel_data[12], [28], [52], [68]
		rx_rmfifostatus[0]	rx_parallel_data[13], [29], [53], [69]
		rx_rmfifostatus[1]	rx_parallel_data[14], [30], [54], [70]
		rx_runningdisp	rx_parallel_data[15], [31], [55], [71]
		rx_data_valid	rx_parallel_data[79]
Configuration-17, PMA Width-20, 8B10B-Enabled, Byte Serializer-Disabled			
data[7:0]	tx_parallel_data[7:0]	data[7:0]	rx_parallel_data[7:0]
data[15:8]	tx_parallel_data[18:11]	data[15:8]	rx_parallel_data[23:16]
tx_datak	tx_parallel_data[8], [19]	rx_datak	rx_parallel_data[8], [24]
tx_forcedisp	tx_parallel_data[9], [20]	rx_errdetect	rx_parallel_data[9], [25]
tx_dispval	tx_parallel_data[10], [21]	rx_syncstatus	rx_parallel_data[10], [26]
		rx_disperr	rx_parallel_data[11], [27]
		rx_patterndetect	rx_parallel_data[12], [28]
		rx_rmfifostatus[0]	rx_parallel_data[13], [29]
		rx_rmfifostatus[1]	rx_parallel_data[14], [30]

continued...

TX Port Function	TX Port	RX Port Function	RX Port
		rx_runningdisp	rx_parallel_data[15], [31]
		rx_data_valid	rx_parallel_data[79]
Configuration-18, PMA Width-20, 8B10B-Enabled, Byte Serializer-Enabled			
data[7:0]	tx_parallel_data[7:0]	data[7:0]	rx_parallel_data[7:0]
data[18:11]	tx_parallel_data[18:11]	data[15:8]	rx_parallel_data[23:16]
data[23:16]	tx_parallel_data[47:40]	data[23:16]	rx_parallel_data[47:40]
data[31:24]	tx_parallel_data[58:51]	data[31:24]	rx_parallel_data[63:56]
tx_datak	tx_parallel_data[8], [19], [48], [59]	rx_datak	rx_parallel_data[8], [24], [48], [64]
tx_forcedisp	tx_parallel_data[9], [20], [49], [60]	rx_errdetect	rx_parallel_data[9], [25], [49], [65]
tx_dispval	tx_parallel_data[10], [21], [50], [61]	rx_syncstatus	rx_parallel_data[10], [26], [50], [66]
		rx_disperr	rx_parallel_data[11], [27], [51], [67]
		rx_patterndetect	rx_parallel_data[12], [28], [52], [68]
		rx_rmfifostatus[0]	rx_parallel_data[13], [29], [53], [69]
		rx_rmfifostatus[1]	rx_parallel_data[14], [30], [54], [70]
		rx_runningdisp	rx_parallel_data[15], [31], [55], [71]
		rx_data_valid	rx_parallel_data[79]

Note: In the following table, the tx_parallel_data and rx_parallel_data mappings shown are for a single channel. To determine the mappings for multi-channel designs, the user must scale the single channel mappings with the appropriate channel multipliers. For example, data[31:0] maps to tx_parallel_data[31:0] and rx_parallel_data[31:0] for single channel designs. For multi-channel designs, data[31:0] for every channel would map to tx_parallel_data[<n-1>80+31:<n-1>80] and rx_parallel_data[<n-1>80+31:<n-1>80], where <n> is the channel number.

Table 79. Simplified Data Interface=Disabled, Double-Rate Transfer=Enabled

TX Port Function	TX Port	RX Port Function	RX Port
Configuration-19, PMA Width-8, 8B10B-NA, Byte Serializer-Enabled			
data[7:0]	tx_parallel_data[7:0] (lower word)	data[7:0]	rx_parallel_data[7:0] (lower word)
data[15:8]	tx_parallel_data[7:0] (upper word)	data[15:8]	rx_parallel_data[7:0] (upper word)
tx_word_marking_bit=0	tx_parallel_data[39] (lower word)		

continued...

TX Port Function	TX Port	RX Port Function	RX Port
tx_word_marking_bit=1	tx_parallel_data[39] (upper word)	rx_syncstatus	rx_parallel_data[10] (lower and upper word)
		rx_patterndetect	rx_parallel_data[12] (lower and upper word)
		rx_word_marking_bit=0	rx_parallel_data[39] (lower word)
		rx_word_marking_bit=1	rx_parallel_data[39] (upper word)
		rx_data_valid	rx_parallel_data[79] (lower and upper word)
Configuration-20, PMA Width-10, 8B10B-Disabled, Byte Serializer-Enabled			
data[9:0]	tx_parallel_data[9:0] (lower word)	data[9:0]	rx_parallel_data[9:0] (lower word)
data[19:10]	tx_parallel_data[9:0] (upper word)	data[19:10]	rx_parallel_data[9:0] (upper word)
tx_word_marking_bit=0	tx_parallel_data[39] (lower word)	rx_syncstatus	rx_parallel_data[10] (lower and upper word)
tx_word_marking_bit=1	tx_parallel_data[39] (upper word)	rx_disperr	rx_parallel_data[11] (lower and upper word)
		rx_patterndetect	rx_parallel_data[12] (lower and upper word)
		rx_rmfifostatus[0]	rx_parallel_data[13] (lower and upper word)
		rx_rmfifostatus[1]	rx_parallel_data[14] (lower and upper word)
		rx_runningdisp	rx_parallel_data[15] (lower and upper word)
		rx_word_marking_bit=0	rx_parallel_data[39] (lower word)
		rx_word_marking_bit=1	rx_parallel_data[39] (upper word)
		rx_data_valid	rx_parallel_data[79] (lower and upper word)
Configuration-21, PMA Width-10, 8B10B-Enabled, Byte Serializer-Enabled			
data[7:0]	tx_parallel_data[7:0] (lower word)	data[7:0]	rx_parallel_data[7:0] (lower word)
data[15:8]	tx_parallel_data[7:0] (upper word)	data[15:8]	rx_parallel_data[7:0] (upper word)
tx_datak	tx_parallel_data[8] (lower and upper word)	rx_datak	rx_parallel_data[8] (lower and upper word)
tx_forcedisp	tx_parallel_data[9] (lower and upper word)	code_violation_status (18)	rx_parallel_data[9] (lower and upper word)
tx_dispval	tx_parallel_data[10] (lower and upper word)	rx_syncstatus	rx_parallel_data[10] (lower and upper word)

continued...

TX Port Function	TX Port	RX Port Function	RX Port
tx_word_marking_bit=0	tx_parallel_data[39] (lower word)	rx_disperr	rx_parallel_data[11] (lower and upper word)
tx_word_marking_bit=1	tx_parallel_data[39] (upper word)	rx_patterndetect	rx_parallel_data[12] (lower and upper word)
		rx_rmfifostatus[0]	rx_parallel_data[13] (lower and upper word)
		rx_rmfifostatus[1]	rx_parallel_data[14] (lower and upper word)
		rx_runningdisp	rx_parallel_data[15] (lower and upper word)
		rx_word_marking_bit=0	rx_parallel_data[39] (lower word)
		rx_word_marking_bit=1	rx_parallel_data[39] (upper word)
		rx_data_valid	rx_parallel_data[79] (lower and upper word)
Configuration-22, PMA Width-16, 8B10B-NA, Byte Serializer-Disabled			
data[7:0]	tx_parallel_data[7:0] (lower word)	data[7:0]	rx_parallel_data[7:0] (lower word)
data[15:8]	tx_parallel_data[7:0] (upper word)	data[15:8]	rx_parallel_data[7:0] (upper word)
tx_word_marking_bit=0	tx_parallel_data[39] (lower word)		
tx_word_marking_bit=1	tx_parallel_data[39] (upper word)	rx_syncstatus	rx_parallel_data[10] (lower and upper word)
		rx_patterndetect	rx_parallel_data[12] (lower and upper word)
		rx_word_marking_bit=0	rx_parallel_data[39] (lower word)
		rx_word_marking_bit=1	rx_parallel_data[39] (upper word)
		rx_data_valid	rx_parallel_data[79] (lower and upper word)
Configuration-23, PMA Width-16, 8B10B-NA, Byte Serializer-Enabled			
data[7:0]	tx_parallel_data[7:0] (lower word)	data[7:0]	rx_parallel_data[7:0] (lower word)
data[15:8]	tx_parallel_data[18:11] (lower word)	data[15:8]	rx_parallel_data[23:16] (lower word)
data[23:16]	tx_parallel_data[7:0] (upper word)	data[23:16]	rx_parallel_data[7:0] (upper word)

continued...

(18) Asserts when the 8b10b decoder detects a code error. Deasserts when the 8b10b decoder does not detect a code error.

TX Port Function	TX Port	RX Port Function	RX Port
data[31:24]	tx_parallel_data[18:11] (upper word)	data[31:24]	rx_parallel_data[23:16] (upper word)
tx_word_marking_bit=0	tx_parallel_data[39] (lower word)		
tx_word_marking_bit=1	tx_parallel_data[39] (upper word)	rx_word_marking_bit=0	rx_parallel_data[39] (lower word)
		rx_word_marking_bit=1	rx_parallel_data[39] (upper word)
		rx_data_valid	rx_parallel_data[79] (lower and upper word)
Configuration-24, PMA Width-20, 8B10B-Disabled, Byte Serializer-Disabled			
data[9:0]	tx_parallel_data[9:0] (lower word)	data[9:0]	rx_parallel_data[9:0] (lower word)
data[19:10]	tx_parallel_data[9:0] (upper word)	data[19:10]	rx_parallel_data[9:0] (upper word)
tx_word_marking_bit=0	tx_parallel_data[39] (lower word)	rx_syncstatus	rx_parallel_data[10] (lower and upper word)
tx_word_marking_bit=1	tx_parallel_data[39] (upper word)	rx_disperr	rx_parallel_data[11] (lower and upper word)
		rx_patterndetect	rx_parallel_data[12] (lower and upper word)
		rx_rmfifostatus[0]	rx_parallel_data[13] (lower and upper word)
		rx_rmfifostatus[1]	rx_parallel_data[14] (lower and upper word)
		rx_runningdisp	rx_parallel_data[15] (lower and upper word)
		rx_word_marking_bit=0	rx_parallel_data[39] (lower word)
		rx_word_marking_bit=1	rx_parallel_data[39] (upper word)
		rx_data_valid	rx_parallel_data[79] (lower and upper word)
Configuration-25, PMA Width-20, 8B10B-Disabled, Byte Serializer-Enabled			
data[19:0]	tx_parallel_data[9:0], [20:11] (lower word)	data[19:0]	rx_parallel_data[9:0], [25:16] (lower word)
data[39:20]	tx_parallel_data[9:0], [20:11] (upper word)	data[39:20]	rx_parallel_data[9:0], [25:16] (upper word)
tx_word_marking_bit=0	tx_parallel_data[39] (lower word)	rx_syncstatus	rx_parallel_data[10], [26] (lower and upper word)
tx_word_marking_bit=1	tx_parallel_data[39] (upper word)	rx_disperr	rx_parallel_data[11], [27] (lower and upper word)

continued...

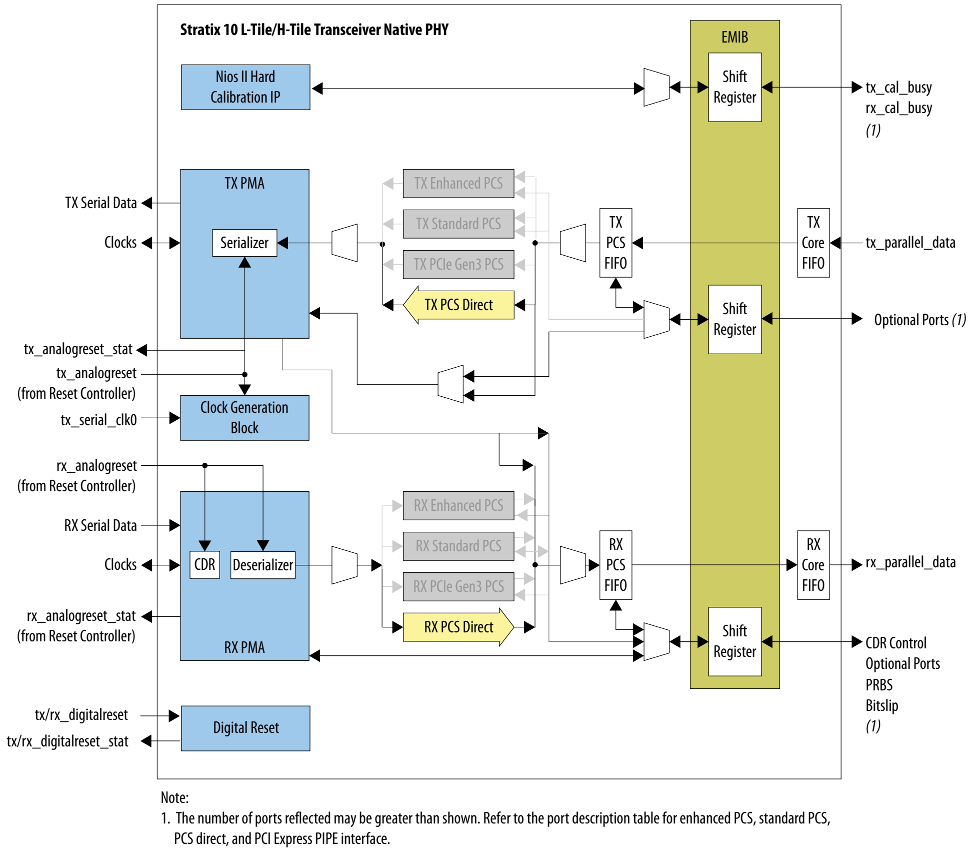
TX Port Function	TX Port	RX Port Function	RX Port
		rx_patterndetect	rx_parallel_data[12], [28] (lower and upper word)
		rx_rmfifostatus[0]	rx_parallel_data[13], [29] (lower and upper word)
		rx_rmfifostatus[1]	rx_parallel_data[14], [30] (lower and upper word)
		rx_runningdisp	rx_parallel_data[15], [31] (lower and upper word)
		rx_word_marking_bit=0	rx_parallel_data[39] (lower word)
		rx_word_marking_bit=1	rx_parallel_data[39] (upper word)
		rx_data_valid	rx_parallel_data[79] (lower and upper word)
Configuration-26, PMA Width-20, 8B10B-Enabled, Byte Serializer-Disabled			
data[7:0]	tx_parallel_data[7:0] (lower word)	data[7:0]	rx_parallel_data[7:0] (lower word)
data[15:8]	tx_parallel_data[7:0] (upper word)	data[15:8]	rx_parallel_data[7:0] (upper word)
tx_datak	tx_parallel_data[8] (lower and upper word)	rx_datak	rx_parallel_data[8] (lower and upper word)
tx_forcedisp	tx_parallel_data[9] (lower and upper word)	code_violation_status (18)	rx_parallel_data[9] (lower and upper word)
tx_dispval	tx_parallel_data[10] (lower and upper word)	rx_syncstatus	rx_parallel_data[10] (lower and upper word)
tx_word_marking_bit=0	tx_parallel_data[39] (lower word)	rx_disperr	rx_parallel_data[11] (lower and upper word)
tx_word_marking_bit=1	tx_parallel_data[39] (upper word)	rx_patterndetect	rx_parallel_data[12] (lower and upper word)
		rx_rmfifostatus[0]	rx_parallel_data[13] (lower and upper word)
		rx_rmfifostatus[1]	rx_parallel_data[14] (lower and upper word)
		rx_runningdisp	rx_parallel_data[15] (lower and upper word)
		rx_word_marking_bit=0	rx_parallel_data[39] (lower word)
		rx_word_marking_bit=1	rx_parallel_data[39] (upper word)
		rx_data_valid	rx_parallel_data[79] (lower and upper word)

continued...

TX Port Function	TX Port	RX Port Function	RX Port
Configuration-27, PMA Width-20, 8B10B-Enabled, Byte Serializer-Enabled			
data[7:0]	tx_parallel_data[7:0] (lower word)	data[7:0]	rx_parallel_data[7:0] (lower word)
data[15:8]	tx_parallel_data[18:11] (lower word)	data[15:8]	rx_parallel_data[23:16] (lower word)
data[23:16]	tx_parallel_data[7:0] (upper word)	data[23:16]	rx_parallel_data[7:0] (upper word)
data[31:24]	tx_parallel_data[18:11] (upper word)	data[31:24]	rx_parallel_data[23:16] (upper word)
tx_datak	tx_parallel_data[8], [19] (lower and upper word)	rx_datak	rx_parallel_data[8], [24] (lower and upper word)
tx_forcedisp	tx_parallel_data[9], [20] (lower and upper word)	code_violation_status [18]	rx_parallel_data[9], [25] (lower and upper word)
tx_dispval	tx_parallel_data[10], [21] (lower and upper word)	rx_syncstatus	rx_parallel_data[10], [26] (lower and upper word)
tx_word_marking_bit=0	tx_parallel_data[39] (lower word)	rx_disperr	rx_parallel_data[11], [27] (lower and upper word)
tx_word_marking_bit=1	tx_parallel_data[39] (upper word)	rx_patterndetect	rx_parallel_data[12], [28] (lower and upper word)
		rx_rmfifostatus[0]	rx_parallel_data[13], [29] (lower and upper word)
		rx_rmfifostatus[1]	rx_parallel_data[14], [30] (lower and upper word)
		rx_runningdisp	rx_parallel_data[15], [31] (lower and upper word)
		rx_word_marking_bit=0	rx_parallel_data[39] (lower word)
		rx_word_marking_bit=1	rx_parallel_data[39] (upper word)
		rx_data_valid	rx_parallel_data[79] (lower and upper word)

2.3.16.3. PCS-Core Interface Ports: PCS-Direct

Figure 31. PCS-Core Interface Ports: PCS-Direct



Note:
1. The number of ports reflected may be greater than shown. Refer to the port description table for enhanced PCS, standard PCS, PCS direct, and PCI Express PIPE interface.

Note: In the following table, the `tx_parallel_data` and `rx_parallel_data` mappings shown are for a single channel. To determine the mappings for multi-channel designs, the user must scale the single channel mappings with the appropriate channel multipliers. For example, `data[31:0]` maps to `tx_parallel_data[31:0]` and `rx_parallel_data[31:0]` for single channel designs. For multi-channel designs, `data[31:0]` for every channel would map to `tx_parallel_data[<n-1>80+31:<n-1>80]` and `rx_parallel_data[<n-1>80+31:<n-1>80]`, where <n> is the channel number.

Table 80. Simplified Data Interface=Disabled, Double-Rate Transfer=Disabled

TX Port Function	TX Port	RX Port Function	RX Port
Configuration-28, PMA Width-8, FPGA Fabric width-8			
`data[7:0]`	`tx_parallel_data[7:0]`	`data[7:0]`	`rx_parallel_data[7:0]`
`tx_fifo_wr_en`	`tx_parallel_data[79]`	`rx_data_valid`	`rx_parallel_data[79]`
Configuration-29, PMA Width-10, FPGA Fabric width-10			
`data[9:0]`	`tx_parallel_data[9:0]`	`data[9:0]`	`rx_parallel_data[9:0]`

continued...

TX Port Function	TX Port	RX Port Function	RX Port
tx_fifo_wr_en	tx_parallel_data[79]	rx_data_valid	rx_parallel_data[79]
Configuration-30, PMA Width-16, FPGA Fabric width-16			
data[15:0]	tx_parallel_data[15:0]	data[15:0]	rx_parallel_data[15:0]
tx_fifo_wr_en	tx_parallel_data[79]	rx_data_valid	rx_parallel_data[79]
Configuration-31, PMA Width-20, FPGA Fabric width-20			
data[19:0]	tx_parallel_data[19:0]	data[19:0]	rx_parallel_data[19:0]
tx_fifo_wr_en	tx_parallel_data[79]	rx_data_valid	rx_parallel_data[79]
Configuration-32, PMA Width-32, FPGA Fabric width-32			
data[31:0]	tx_parallel_data[31:0]	data[31:0]	rx_parallel_data[31:0]
tx_fifo_wr_en	tx_parallel_data[79]	rx_prbs_err	rx_parallel_data[35]
		rx_prbs_done	rx_parallel_data[36]
		rx_data_valid	rx_parallel_data[79]
Configuration-33, PMA Width-40, FPGA Fabric width-40			
data[39:0]	tx_parallel_data[39:0]	data[39:0]	rx_parallel_data[39:0]
tx_fifo_wr_en	tx_parallel_data[79]	rx_data_valid	rx_parallel_data[79]
Configuration-34, PMA Width-64, FPGA Fabric width-64			
data[31:0]	tx_parallel_data[31:0]	data[31:0]	rx_parallel_data[31:0]
data[63:32]	tx_parallel_data[71:40]	data[63:32]	rx_parallel_data[71:40]
tx_fifo_wr_en	tx_parallel_data[79]	rx_data_valid	rx_parallel_data[79]

Note: In the following table, the tx_parallel_data and rx_parallel_data mappings shown are for a single channel. To determine the mappings for multi-channel designs, the user must scale the single channel mappings with the appropriate channel multipliers. For example, data[31:0] maps to tx_parallel_data[31:0] and rx_parallel_data[31:0] for single channel designs. For multi-channel designs, data[31:0] for every channel would map to tx_parallel_data[<n-1>80+31:<n-1>80] and rx_parallel_data[<n-1>80+31:<n-1>80], where <n> is the channel number.

Table 81. Simplified Data Interface=Disabled, Double-Rate Transfer=Enabled

TX Port Function	TX Port	RX Port Function	RX Port
Configuration-35, PMA Width-16, FPGA Fabric width-8			
data[7:0]	tx_parallel_data[7:0] (lower word)	data[7:0]	rx_parallel_data[7:0] (lower word)
data[15:8]	tx_parallel_data[7:0] (upper word)	data[15:8]	rx_parallel_data[7:0] (upper word)
tx_word_marking_bit=0	tx_parallel_data[19] (lower word)	rx_word_marking_bit=0	rx_parallel_data[39] (upper word)
tx_word_marking_bit=1	tx_parallel_data[19] (upper word)	rx_word_marking_bit=1	rx_parallel_data[39] (lower word)

continued...

Send Feedback

TX Port Function	TX Port	RX Port Function	RX Port
tx_fifo_wr_en	tx_parallel_data[79] (lower and upper word)	rx_data_valid	rx_parallel_data[79]
Configuration-36, PMA Width-20, FPGA Fabric width-10			
data[9:0]	tx_parallel_data[9:0] (lower word)	data[9:0]	rx_parallel_data[9:0] (lower word)
data[19:10]	tx_parallel_data[9:0] (upper word)	data[19:10]	rx_parallel_data[9:0] (upper word)
tx_word_marking_bit=0	tx_parallel_data[19] (lower word)	rx_word_marking_bit=0	rx_parallel_data[39] (upper word)
tx_word_marking_bit=1	tx_parallel_data[19] (upper word)	rx_word_marking_bit=1	rx_parallel_data[39] (lower word)
tx_fifo_wr_en	tx_parallel_data[79] (lower and upper word)	rx_data_valid	rx_parallel_data[79]
Configuration-37, PMA Width-32, FPGA Fabric width-16			
data[15:0]	tx_parallel_data[15:0] (lower word)	data[15:0]	rx_parallel_data[15:0] (lower word)
data[31:16]	tx_parallel_data[15:0] (upper word)	data[31:16]	rx_parallel_data[15:0] (upper word)
tx_word_marking_bit=0	tx_parallel_data[19] (lower word)	rx_word_marking_bit=0	rx_parallel_data[39] (upper word)
tx_word_marking_bit=1	tx_parallel_data[19] (upper word)	rx_word_marking_bit=1	rx_parallel_data[39] (lower word)
tx_fifo_wr_en	tx_parallel_data[79] (lower and upper word)	rx_data_valid	rx_parallel_data[79]
Configuration-38, PMA Width-40, FPGA Fabric width-20			
data[19:0]	tx_parallel_data[19:0] (lower word)	data[19:0]	rx_parallel_data[19:0] (lower word)
data[39:20]	tx_parallel_data[19:0] (upper word)	data[39:20]	rx_parallel_data[19:0] (upper word)
tx_word_marking_bit=0	tx_parallel_data[39] (lower word)	rx_word_marking_bit=0	rx_parallel_data[39] (lower word)
tx_word_marking_bit=1	tx_parallel_data[39] (upper word)	rx_word_marking_bit=1	rx_parallel_data[39] (upper word)
tx_fifo_wr_en	tx_parallel_data[79] (lower and upper word)	rx_data_valid	rx_parallel_data[79] (lower and upper word)
Configuration-39, PMA Width-64, FPGA Fabric width-32			
data[31:0]	tx_parallel_data[31:0] (lower word)	data[31:0]	rx_parallel_data[31:0] (lower word)
data[63:32]	tx_parallel_data[31:0] (upper word)	data[63:32]	rx_parallel_data[31:0] (upper word)

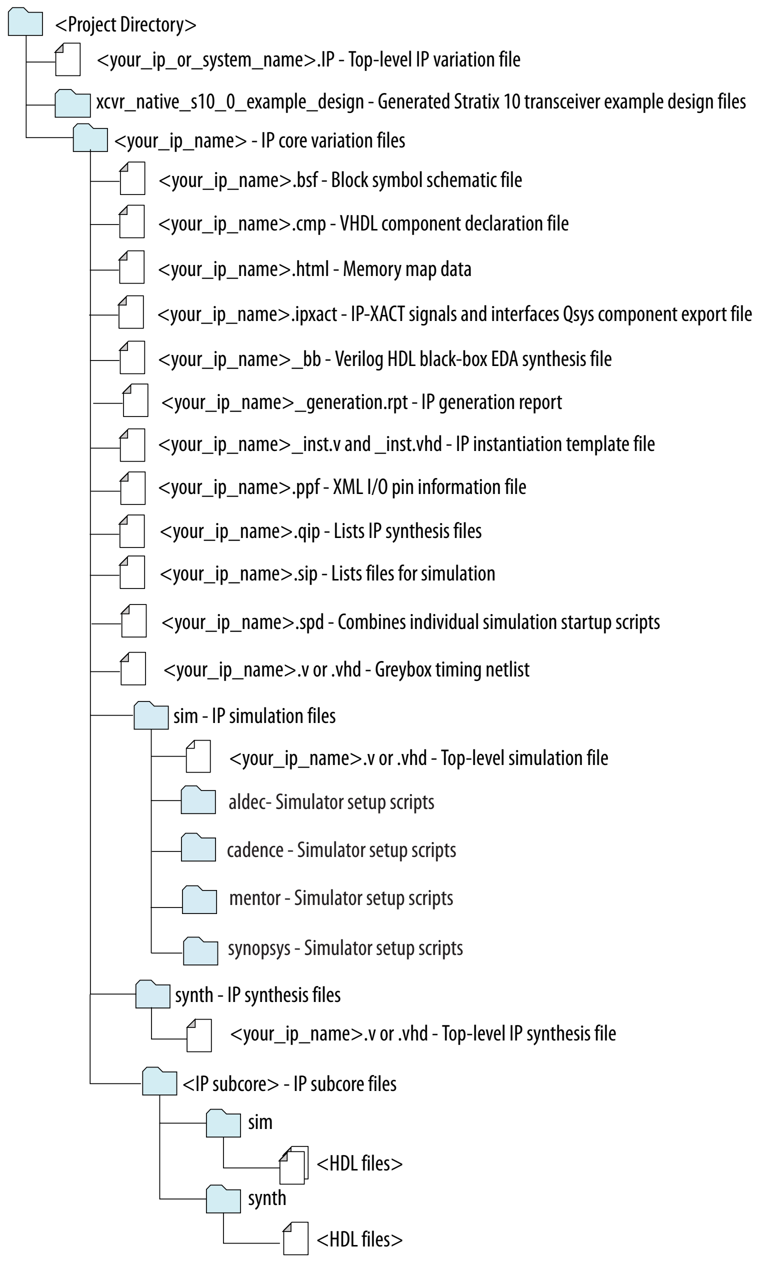
continued...

TX Port Function	TX Port	RX Port Function	RX Port
`tx_word_marking_bit=0`	`tx_parallel_data[39]` (lower word)	`rx_word_marking_bit=0`	`rx_parallel_data[39]` (lower word)
`tx_word_marking_bit=1`	`tx_parallel_data[39]` (upper word)	`rx_word_marking_bit=1`	`rx_parallel_data[39]` (upper word)
`tx_fifo_wr_en`	`tx_parallel_data[79]` (lower and upper word)	`rx_data_valid`	`rx_parallel_data[79]` (lower and upper word)

2.3.17. IP Core File Locations

When you generate your Transceiver Native PHY IP, the Intel Quartus Prime Pro Edition software generates the HDL files that define your instance of the IP. In addition, the Intel Quartus Prime Pro Edition software generates an example Tcl script to compile and simulate your design in the ModelSim simulator. It also generates simulation scripts for Synopsys VCS, Aldec Active-HDL, Aldec Riviera-Pro, and Cadence Incisive Enterprise.

Figure 32. Directory Structure for Generated Files


<Project Directory>
<your_ip_or_system_name>.IP - Top-level IP variation file
xcvr_native_s10_0_example_design - Generated Stratix 10 transceiver example design files
<your_ip_name> - IP core variation files
<your_ip_name>.bsf - Block symbol schematic file
<your_ip_name>.cmp - VHDL component declaration file
<your_ip_name>.html - Memory map data
<your_ip_name>.ipxact - IP-XACT signals and interfaces Qsys component export file
<your_ip_name>_bb - Verilog HDL black-box EDA synthesis file
<your_ip_name>_generation.rpt - IP generation report
<your_ip_name>_inst.v and _inst.vhd - IP instantiation template file
<your_ip_name>.ppf - XML I/O pin information file
<your_ip_name>.qip - Lists IP synthesis files
<your_ip_name>.sip - Lists files for simulation
<your_ip_name>.spd - Combines individual simulation startup scripts
<your_ip_name>.v or .vhd - Greybox timing netlist
sim - IP simulation files
<your_ip_name>.v or .vhd - Top-level simulation file
aldec- Simulator setup scripts
cadence - Simulator setup scripts
mentor - Simulator setup scripts
synopsys - Simulator setup scripts
synth - IP synthesis files
<your_ip_name>.v or .vhd - Top-level IP synthesis file
<IP subcore> - IP subcore files
sim
<HDL files>
synth
<HDL files>

The following table describes the directories and the most important files for the parameterized Transceiver Native PHY IP core and the simulation environment. These files are in clear text.

Table 82. Transceiver Native PHY Files and Directories

File Name	Description
<project_dir>	The top-level project directory.
<your_ip_name> **.v** or **.vhd**	The top-level design file.
<your_ip_name> **.qip**	A list of all files necessary for Intel Quartus Prime compilation.
<your_ip_name> **.bsf**	A Block Symbol File (.bsf) for your Transceiver Native PHY instance.
<project_dir>/<your_ip_name>/	The directory that stores the HDL files that define the Transceiver Native PHY IP.
<project_dir>/sim	The simulation directory.
*<project_dir>/sim/**aldec***	Simulation files for Riviera-PRO simulation tools.
*<project_dir>/sim/**cadence***	Simulation files for Cadence simulation tools.
*<project_dir>/sim/**mentor***	Simulation files for Mentor simulation tools.
*<project_dir>/sim/**synopsys***	Simulation files for Synopsys simulation tools.
<project_dir>/synth	The directory that stores files used for synthesis.

The Verilog and VHDL Transceiver Native PHY IP cores have been tested with the following simulators:

- ModelSim SE
- Synopsys VCS MX
- Cadence NCSim

If you select VHDL for your transceiver PHY, only the wrapper generated by the Intel Quartus Prime Pro Edition software is in VHDL. All the underlying files are written in Verilog or SystemVerilog. To enable simulation using a VHDL-only ModelSim license, the underlying Verilog and SystemVerilog files for the Transceiver Native PHY IP are encrypted so that they can be used with the top-level VHDL wrapper without using a mixed-language simulator.

For more information about simulating with ModelSim, refer to the *Mentor Graphics ModelSim Support* chapter in volume 3 of the *Intel Quartus Prime Handbook*.

The Transceiver Native PHY IP cores do not support the NativeLink feature in the Intel Quartus Prime Pro Edition software.

Related Information

- Simulating the Native PHY IP Core on page 236
- Mentor Graphics ModelSim Support

2.4. Using the Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY Intel Stratix 10 FPGA IP Core

This section describes the use of the Intel-provided Transceiver Native PHY IP core. This Native PHY IP core is the primary design entry tool and provides direct access to Intel Stratix 10 transceiver PHY features.

Use the Native PHY IP core to configure the transceiver PHY for your protocol implementation. To instantiate the IP, select the Intel Stratix 10 device family, click **Tools ➤ IP Catalog** to select your IP core variation. Use the **Parameter Editor** to specify the IP parameters and configure the PHY IP for your protocol implementation. To quickly configure the PHY IP, select a preset that matches your protocol configuration as a starting point. Presets are PHY IP configuration settings for various protocols that are stored in the IP **Parameter Editor**. Presets are explained in detail in the *Presets* section below.

You can also configure the PHY IP by selecting an appropriate **Transceiver Configuration Rule**. The transceiver configuration rules check the valid combinations of the PCS and PMA blocks in the transceiver PHY layer, and report errors or warnings for any invalid settings.

Use the Native PHY IP core to instantiate one of the following PCS options:

- Standard PCS
- Enhanced PCS
- PCIe Gen3 PCS
- PCS Direct

Based on the Transceiver Configuration Rule that you select, the PHY IP core selects the appropriate PCS. Refer to the *How to Place Channels for PIPE Configuration* section or the PCIe solutions guides on restrictions on placement of transceiver channels next to active banks with PCI Express interfaces that are Gen3 capable.

After you configure the PHY IP core in the **Parameter Editor**, click **Generate HDL** to generate the IP instance. The top level file generated with the IP instance includes all the available ports for your configuration. Use these ports to connect the PHY IP core to the PLL IP core, the reset controller IP core, and to other IP cores in your design.

Figure 33. Native PHY IP Core Ports and Functional Blocks

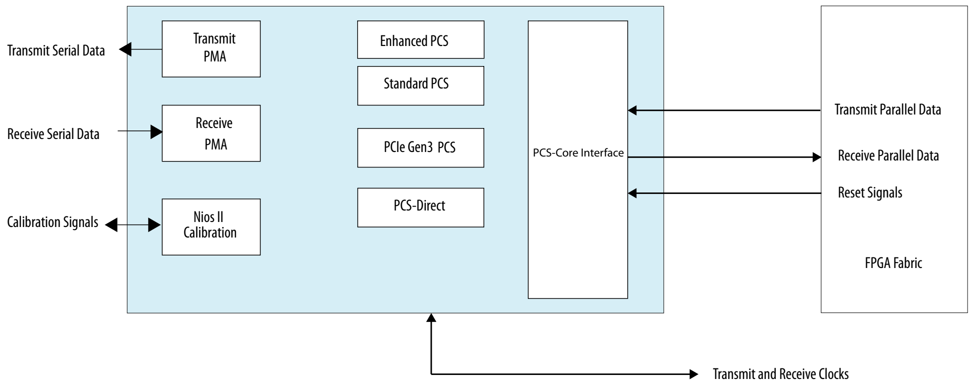
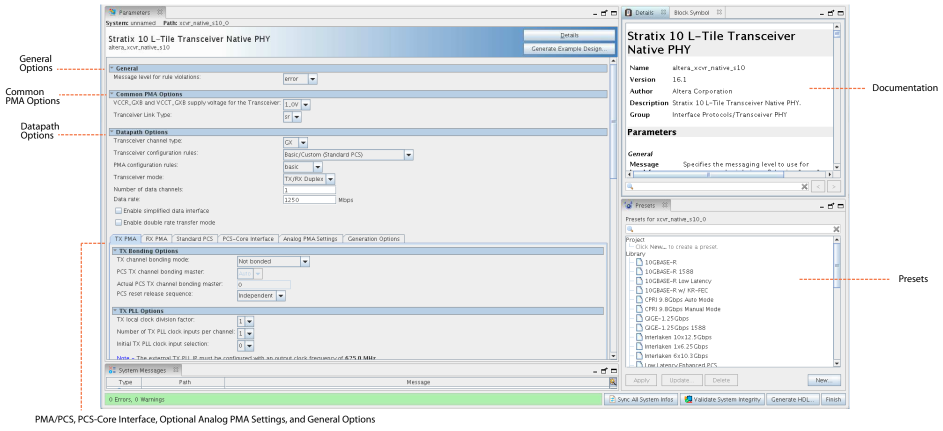


Figure 34. Native PHY IP Core Parameter Editor



System: unnamed Path: xcvr_native_s10_0

Stratix 10 L-Tile Transceiver Native PHY
altera_xcvr_native_s10

General
Message level for rule violations: error

Common PMA Options
VCCR_GXB and VCCT_GXB supply voltage for the Transceiver: 1_0V
Tranceiver Link Type: sr

Datapath Options
Transceiver channel type: GX
Transceiver configuration rules: Basic/Custom (Standard PCS)
PMA configuration rules: basic
Transceiver mode: TX/RX Duplex
Number of data channels: 1
Data rate: 1250 Mbps
☐ Enable simplified data interface
☐ Enable double rate transfer mode

TX Bonding Options
TX channel bonding mode: Not bonded
PCS TX channel bonding master: Auto
Actual PCS TX channel bonding master: 0
PCS reset release sequence: Independent

TX PLL Options
TX local clock division factor: 1
Number of TX PLL clock inputs per channel: 1
Initial TX PLL clock input selection: 0
Note – The external TX PLL IP must be configured with an output clock frequency of 625.0 MHz

0 Errors, 0 Warnings

Stratix 10 L-Tile Transceiver Native PHY

Name	altera_xcvr_native_s10
Version	16.1
Author	Altera Corporation
Description	Stratix 10 L-Tile Transceiver Native PHY.
Group	Interface Protocols/Transceiver PHY

Parameters

General
Message Specifies the messaging level to use for

Presets for xcvr_native_s10_0
Project
Click New... to create a preset.
Library
10GBASE-R
10GBASE-R 1588
10GBASE-R Low Latency
10GBASE-R w/ KR-FEC
CPRI 9.8Gbps Auto Mode
CPRI 9.8Gbps Manual Mode
GIGE-1.25Gbps
GIGE-1.25Gbps 1588
Interlaken 10x12.5Gbps
Interlaken 1x6.25Gbps
Interlaken 6x10.3Gbps
Low Latency Enhanced PCS

Note: Although the Intel Quartus Prime Pro Edition software provides legality checks, the supported FPGA fabric to PCS interface widths and the supported datarates are pending characterization.

Related Information

- General and Datapath Parameters on page 36
- How to Place Channels for PIPE Configurations on page 207

2.4.1. PMA Functions

The Native PHY IP core allows you to set the TX PMA and RX PMA through the **Analog PMA Settings** tab.

Note: These PMA settings apply across all channels for that PHY instance. If you want to have different PMA settings for different channels in one PHY instance, you must use the **Sample QSF Assignments** option to overwrite these settings on a per-channel basis. You can also find a list of sample transmitter QSF assignments in the *TX PMA Use Model* section.

Related Information

TX PMA Use Model on page 108

2.4.1.1. TX PMA Use Model

You can configure all TX PMA settings using the **Analog PMA Settings** tab, or if needed, through the corresponding QSF assignments provided through the tab. You can also reconfigure them through the registers outlined in the *Logical View of the L-Tile/H-Tile Transceiver Registers*.

Transmitter QSF Assignments

These are the sample QSF assignments for the transmitter PMA. The syntax is as shown, with an example below.

Syntax:

```
set_instance_assignment -name HSSI_PARAMETER "$
{full_attribute_name}={chosen_attribute_value} " -to ${tx_pin}
```

Example:

```
set_instance_assignment -name HSSI_PARAMETER ""pma_tx_buf_vod_output_swing_ctrl
=31 " -to tx_serial_pin[0]
```

Note: Attribute names are longer than the ones you find on the Register Map. Refer to the following table for a list of attribute names and values.

Table 83. Transmitter QSF Assignment Attributes

Attribute	Full Attribute Name	Attribute Values
TX Output Swing Level (VOD)	`pma_tx_buf_vod_output_swing_ctrl`	17 (600 mV) until 31 (V_{CCT} or Transmitter Power Supply Voltage)
Pre-emphasis 1st post-tap magnitude	`pma_tx_buf_pre_emp_switching_ctrl_1st_post_tap`	0-24
Pre-emphasis 1st post-tap polarity	`pma_tx_buf_pre_emp_sign_1st_post_tap`	`fir_post_1t_neg` (negative) OR `fir_post_1t_pos` (positive)
Pre-emphasis 1st pre-tap magnitude	`pma_tx_buf_pre_emp_switching_ctrl_pre_tap_1t`	0-15
Pre-emphasis 1st pre-tap polarity	`pma_tx_buf_pre_emp_sign_pre_tap_1t`	`fir_pre_1t_neg` (negative) OR `fir_pre_1t_pos` (positive)
Slew Rate	`pma_tx_buf_slew_rate_ctrl`	0 (slowest) to 5 (fastest)

Related Information

Logical View of the L-Tile/H-Tile Transceiver Registers on page 450

2.4.1.2. RX PMA Use Model

The Adaptive Parametric Tuning (ADAPT) engine allows the continuous time linear equalization (CTLE) and decision feedback equalization (DFE) blocks of the RX PMA to adapt to an optimal value.

Adaptation is available for CTLE, VGA, and DFE. The RX PMA Adaptation Modes include:

- **Manual CTLE**, **Manual VGA**, **DFE Off**
- **Adaptive CTLE**, **Adaptive VGA**, **DFE Off**
- **Adaptive CTLE**, **Adaptive VGA**, **All-Tap Adaptive DFE**
- **Adaptive CTLE**, **Adaptive VGA**, **1-Tap Adaptive DFE**
- **ctle_dfe_mode_2 (Adaptive Mode for PCIe Gen3)** — This mode fixes the AC gain and leaves everything else adaptive.

Table 84. RX PMA Adaptation Range of Values

Analog PMA Setting	Range of Values
CTLE	AC gain: 0-15 EQ gain: 0-47
VGA	0-31
DFE (Adaptive only)	Tap1: -63 to 63 Taps 2-3: -31 to 31 Taps 4-7: -15 to 15 Taps 8-15: -7 to 7

2.4.1.2.1. Using RX in Manual Mode

When CTLE or VGA is in manual mode, you can use the Transceiver Native PHY IP Core or QSF assignments to select your CTLE AC gain, CTLE EQ gain, and VGA gain settings.

Contact Intel Support to obtain the recommended values for your system loss profile.

These values can always be overwritten through the Avalon memory-mapped interface. The new values take precedence over values defined statically in the PHY IP core. Use this method to dynamically set values and avoid re-compilation.

Note: Intel does not recommend that you use manual mode due to PVT variations.

2.4.1.2.2. Using RX in Adaptive Mode

When CTLE, VGA or DFE is in adaptive mode, you must use reconfiguration to access the Avalon memory-mapped interface and reset and start the adaptation. Ensure that `rx_ready` is asserted before starting adaptation. The adaptation engine converges the values within 50 ms.

Refer to the RX PMA sections of the *Logical View of the L-Tile/H-Tile Transceiver Registers* for more details on the register addresses, and the *Intel Stratix 10 Dynamic Transceiver Reconfiguration* chapter for details on the Avalon memory-mapped interface and how to perform dynamic reads and writes to the analog PMA settings.

Related Information

- Reconfiguration Interface and Dynamic Reconfiguration on page 394
- Logical View of the L-Tile/H-Tile Transceiver Registers on page 450

2.4.1.2.3. Setting RX PMA Adaptation Modes

There are a number of attributes that are configured differently across the RX adaptation modes. Refer to the Setting RX PMA Adaptation Modes resource for more details.

Related Information

- Logical View of the L-Tile/H-Tile Transceiver Registers on page 450
- Setting RX PMA Adaptation Modes

2.4.1.2.4. Register Sequences

When using the logical view register map to reconfigure the PMA, you must use the register addresses in the following sections:

- Pre-Emphasis: To change pre-emphasis manually
- VOD: to manually change VOD
- Manual CTLE: To change CTLE manually
- Manual VGA: To change VGA manually
- Adaptation Control - Start and Adaptation Control - Stop: To reset and start adaptation
- Adaptation Initial Values
- Adapted Value Readout: To read out the adapted CTLE, VGA, and DFE values

2.4.2. PCS Functions

2.4.2.1. Receiver Word Alignment

2.4.2.1.1. Word Alignment Using the Standard PCS

To achieve receiver word alignment, use the word aligner of the Standard PCS in one of the following modes:

- RX data bitslip
- Manual mode
- Synchronous State Machine
- Deterministic Latency Mode
- Word alignment in GbE Mode

RX Bitslip

To use the RX bitslip, select **Enable rx_bitslip port** and set the word aligner mode to **bitslip**. This adds `rx_bitslip` as an input control port. An active high edge on `rx_bitslip` slips one bit at a time. When `rx_bitslip` is toggled, the word aligner slips one bit at a time on every active high edge. Assert the `rx_bitslip` signal for at least 200 ns to ensure it passes through the slow shift register. You can verify this feature by monitoring `rx_parallel_data`.

The RX bitslip feature is optional and may or may not be enabled.

Figure 35. RX Bitslip in 8-bit Mode

`tx_parallel_data` = 8'hbc

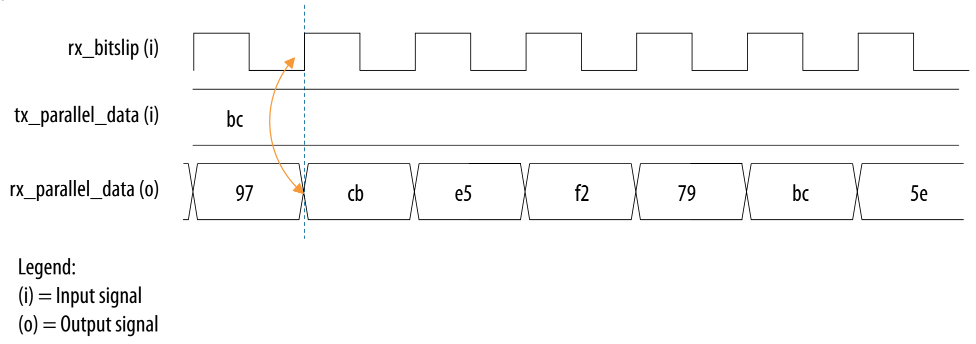


Figure 36. RX Bitslip in 10-bit Mode

`tx_parallel_data` = 10'h3bc

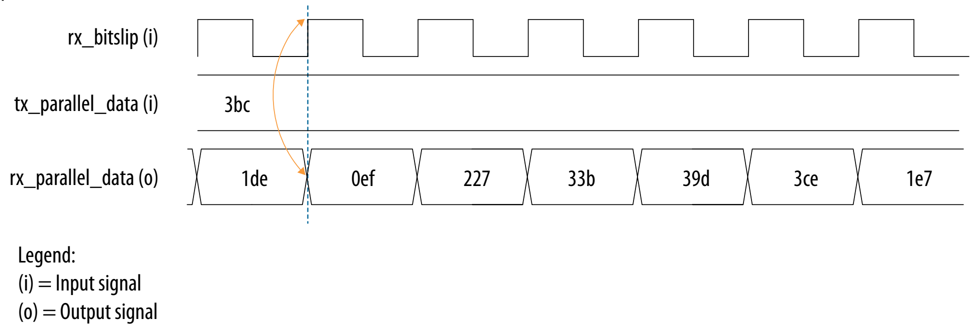
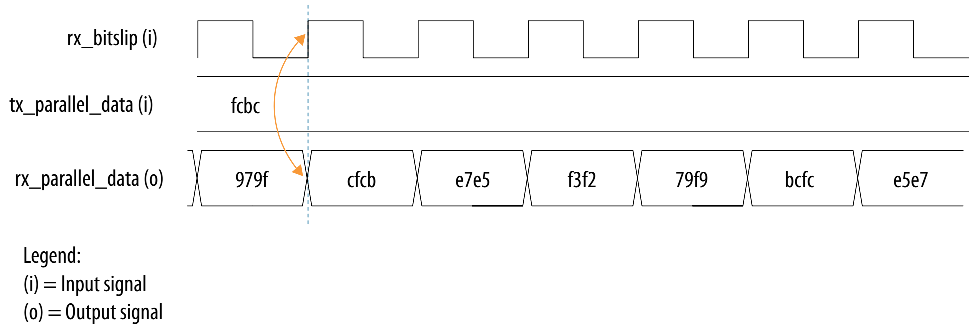


Figure 37. RX Bitslip in 16-bit Mode

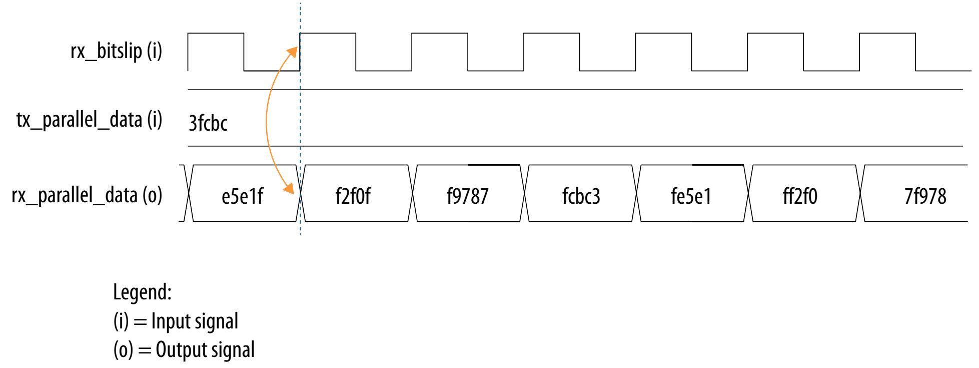
`tx_parallel_data` = 16'hfcbc



Send Feedback

Figure 38. RX Bitslip in 20-bit Mode

`tx_parallel_data` = 20'h3fcbc



Refer to the *Word Aligner bitslip Mode* section for more information.

Related Information

- Word Aligner Bitslip Mode on page 372
- Intel Stratix 10 L-/H-Tile Word Aligner Bitslip Calculator

Word Aligner Manual Mode

Refer to the Intel Stratix 10 (L/H-Tile) Word Aligner Bitslip Calculator to calculate the number of slips you require to achieve alignment based on the word alignment pattern and length. To use this mode:

1. Set the **RX word aligner mode** to **Manual (FPGA Fabric controlled)**.
2. Set the **RX word aligner pattern length** option according to the PCS-PMA interface width.
3. Enter a hexadecimal value in the **RX word aligner pattern (hex)** field.

This mode adds `rx_patterndetect` and `rx_syncstatus`. You can select the **Enable rx_std_wa_patternalign port** option to enable `rx_std_wa_patternalign`.

Note:

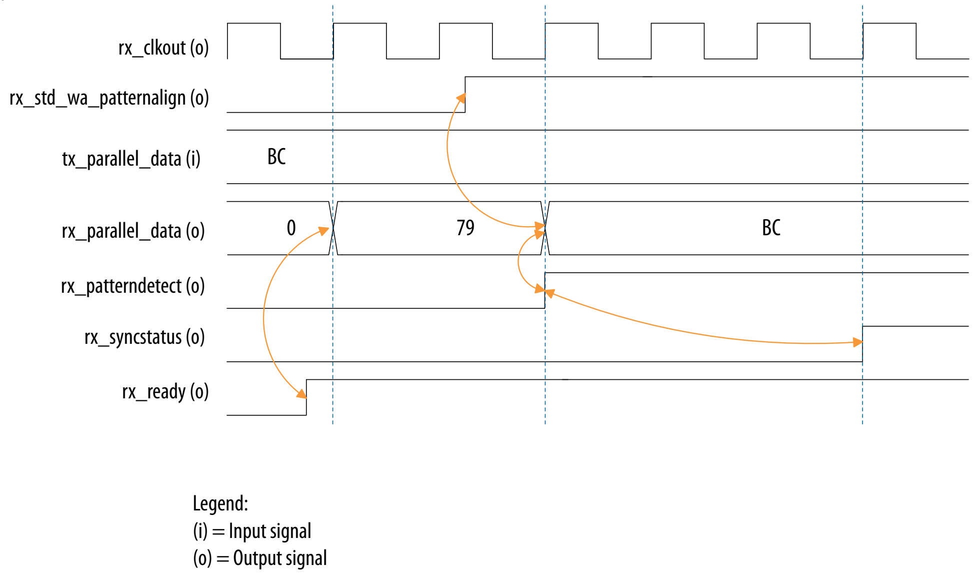
- `rx_patterndetect` is asserted whenever there is a pattern match.
- `rx_syncstatus` is asserted after the word aligner achieves synchronization, 3 `clkout` cycles after `rx_patterndetect` goes high.
- `rx_std_wa_patternalign` is asserted to re-align and resynchronize.
- If there is more than one channel in the design, `rx_patterndetect`, `rx_syncstatus` and `rx_std_wa_patternalign` become buses in which each bit corresponds to one channel.

You can verify this feature by monitoring `rx_parallel_data`.

The following timing diagrams demonstrate how to use the ports and show the relationship between the various control and status signals. In the top waveform, `rx_parallel_data` is initially misaligned. After asserting the rx_std_wa_patternalign signal, it becomes aligned. The bottom waveform shows the behavior of the rx_syncstatus signal when rx_parallel_data is already aligned.

Figure 39. Manual Mode when the PCS-PMA Interface Width is 8 Bits

tx_parallel_data = 8'hBC and the word aligner pattern = 8'hBC



In manual alignment mode, the word alignment operation is manually controlled with the rx_std_wa_patternalign input signal or the rx_enapatternalign register. The word aligner operation is level-sensitive to rx_enapatternalign. The word aligner asserts the rx_syncstatus signal for one parallel clock cycle whenever it re-aligns to the new word boundary.

Note: Only the 10-bit mode is level sensitive. The 8-, 16-, and 20-bit modes are edge sensitive.

Refer to the *Word Aligner Manual Mode* section for more information.

Related Information

Word Aligner Manual Mode on page 373

Word Aligner Synchronous State Machine Mode

To use this mode:

- Select the **Enable TX 8B/10B encoder** option.
- Select the **Enable RX 8B/10B decoder** option.

The 8B/10B encoder and decoder add the following additional ports:

- `tx_datak`
- `rx_datak`
- `rx_errdetect`
- `rx_disperr`
- `rx_runningdisp`

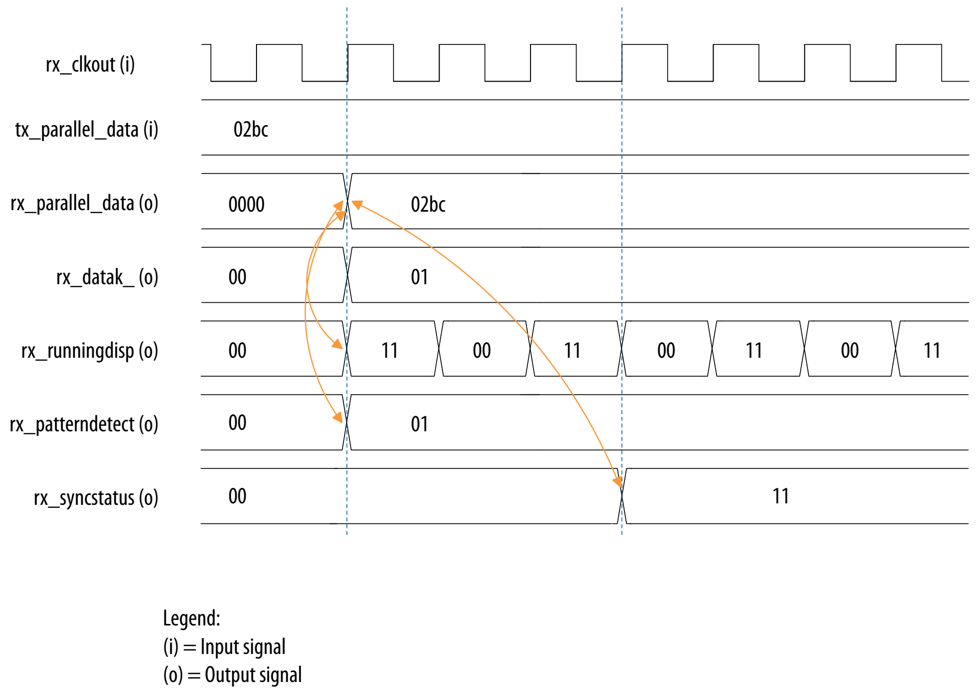
1. Set the **RX word aligner mode** to **synchronous state machine**.
2. Set the **RX word aligner pattern length** option according to the PCS-PMA interface width.
3. Enter a hexadecimal value in the **RX word aligner pattern (hex)** field.

The RX word aligner pattern is the 8B/10B encoded version of the data pattern. You can also specify the number of word alignment patterns (LSB first) to achieve synchronization, the number of invalid data words to lose synchronization, and the number of valid data words to decrement error count. This mode adds two additional ports: `rx_patterndetect` and `rx_syncstatus`.

Note:
- `rx_patterndetect` is asserted whenever there is a pattern match.
- `rx_syncstatus` is asserted after the word aligner achieves synchronization, 3 `clkout` cycles after `rx_patterndetect` goes high.
- If there is more than one channel in the design, `tx_datak`, `rx_datak`, `rx_errdetect`, `rx_disperr`, `rx_runningdisp`, `rx_patterndetect`, and `rx_syncstatus` become buses in which each bit corresponds to one channel.

You can verify this feature by monitoring `rx_parallel_data`.

Figure 40. Synchronization State Machine Mode when the PCS-PMA Interface Width is 16 Bits



Refer to the *Word Aligner Synchronous State Machine Mode* section for more information.

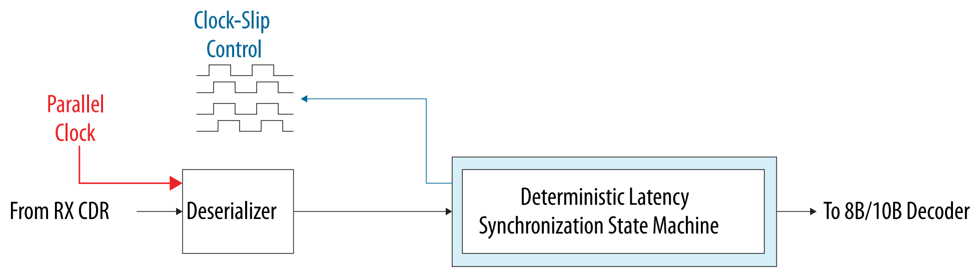
Related Information

Word Aligner Synchronous State Machine Mode on page 373

Word Aligner in Deterministic Latency Mode for CPRI

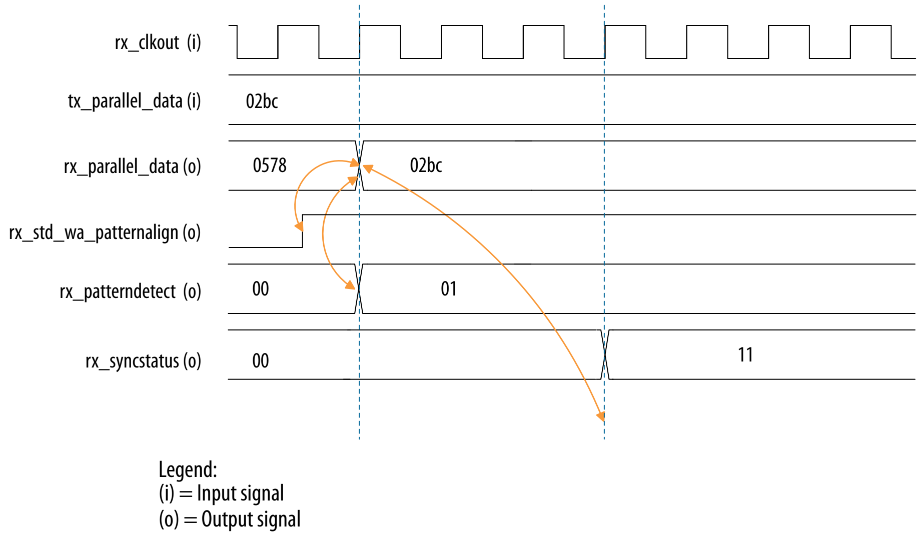
The deterministic latency state machine in the word aligner reduces the known delay variation from the word alignment process. It automatically synchronizes and aligns the word boundary by slipping one half of a serial clock cycle (1UI) in the deserializer. Incoming data to the word aligner is aligned to the boundary of the word alignment pattern (K28.5).

Figure 41. Deterministic Latency State Machine in the Word Aligner



When using deterministic latency state machine mode, assert `rx_std_wa_patternalign` to initiate the pattern alignment after the reset sequence is complete. This is an edge-triggered signal in all cases except one: when the word aligner is in manual mode and the PMA width is 10 bits, in which case `rx_std_wa_patternalign` is level sensitive.

Figure 42. Word Aligner in Deterministic Latency Mode 16 Bits Waveform



Calculating Latency through the Word Aligner

You can use the word aligner in either of the following modes to achieve deterministic latency:

- Deterministic Latency State Machine (DLSM)
- Synchronous State Machine (SSM)
- Manual mode
- RX Bitslip mode

Table 85. Word Aligner Latency in DLSM and RX Bitslip Modes

Condition	Latency
If `rx_std_bitslipboundarysel` [19] is EVEN	Constant
If `rx_std_bitslipboundarysel` [19] is ODD	Constant + 1 UI

Table 86. Word Aligner Latency in SSM and Manual Modes

Condition	Latency
10-bit PMA	Constant (nsec) + (`rx_std_bitslipboundarysel`) * UI[20] (nsec)
20-bit PMA	Constant (nsec) + (19 - `rx_std_bitslipboundarysel`) * UI[20] (psec)

(19) If you do not account for `rx_std_bitslipboundarysel`, there will be nondeterminism of 1 UI in these word aligner modes.

Related Information

- Deterministic Latency Use Model on page 146
- MySupport

Word Alignment in GbE Mode

The word aligner for the GbE and GbE with IEEE 1588v2 protocols is configured in automatic synchronization state machine mode.

The Intel Quartus Prime Pro Edition software automatically configures the synchronization state machine to indicate synchronization when the receiver receives three consecutive synchronization ordered sets. A synchronization ordered set is a /K28.5/ code group followed by an odd number of valid /Dx.y/ code groups. The fastest way for the receiver to achieve synchronization is to receive three continuous {/K28.5/, /Dx.y/} ordered sets.

The Native PHY IP core signals receiver synchronization status on the `rx_syncstatus` port of each channel. A high on the `rx_syncstatus` port indicates that the lane is synchronized; a low on the `rx_syncstatus` port indicates that the lane has fallen out of synchronization. The receiver loses synchronization when it detects three invalid code groups separated by less than three valid code groups or when it is reset.

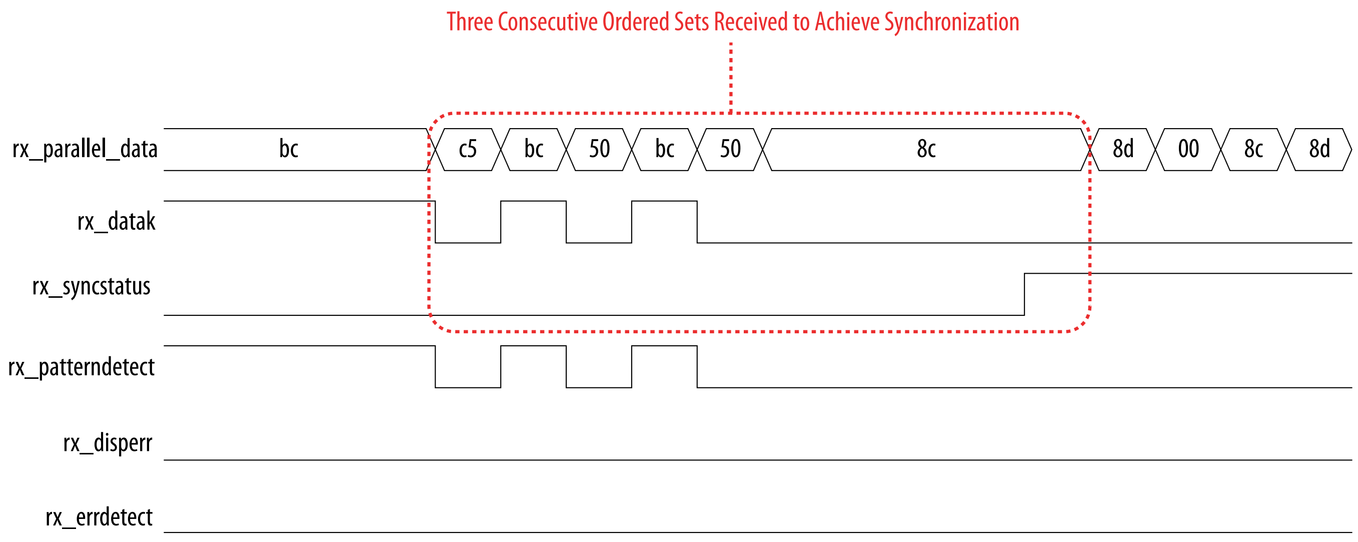
Table 87. Synchronization State Machine Parameter Settings for GbE

Synchronization State Machine Parameter	Setting
Number of word alignment patterns to achieve sync	3
Number of invalid data words to lose sync	3
Number of valid data words to decrement error count	3

The following figure shows `rx_syncstatus` high when three consecutive ordered sets are sent through `rx_parallel_data`.

(20) UI is the inverse of the serial datarate. These constants have different values based on the PHY configuration. Contact MySupport for further details on the expected values. The `rx_std_bitslipboundarysel` output status port is available to you at the PCS-Core interface.

Figure 43. rx_syncstatus High



2.4.2.1.2. Word Alignment Using the Enhanced PCS

Use the block synchronizer of the Enhanced PCS to achieve word boundary for protocols such as 10GBase-R, 40G/100G Ethernet, and so on.

Refer to the *Block Synchronizer* section for more information.

Use the gearbox in the PCIe Gen3 PCS for 128B/130B as required by the PIPE Gen3 applications. Refer to the *Gearbox* section for more information.

Related Information

- Block Synchronizer on page 361
- Gearbox on page 176

2.4.2.2. Receiver Clock Compensation

2.4.2.2.1. Clock Compensation Using the Standard PCS

Use the Rate Match FIFO of the Standard PCS in one of the following modes, to compensate for clock differences between the RX PCS and the FPGA fabric:

- Rate Match FIFO in Basic 10-bit PMA mode
- Rate Match FIFO in Basic 20-bit PMA mode
- Rate Match FIFO in GbE mode
- Rate Match FIFO in PIPE mode
- Rate Match FIFO in PIPE 0 ppm mode

Rate Match FIFO in Basic (Single Width) Mode

Only the rate match FIFO operation is covered in these steps.

1. Select **basic (single width)** in the **RX rate match FIFO mode** list.
2. Enter values for the following parameters.

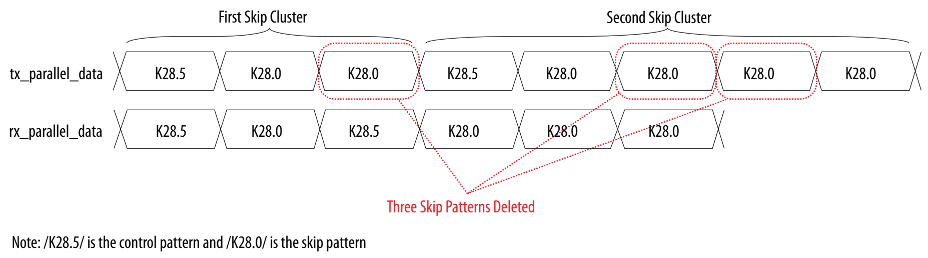
Parameter	Value	Description
RX rate match insert/delete +ve pattern (hex)	20 bits of data specified as a hexadecimal string	The first 10 bits correspond to the skip pattern and the last 10 bits correspond to the control pattern. The skip pattern must have neutral disparity.
RX rate match insert/delete –ve pattern (hex)	20 bits of data specified as a hexadecimal string	The first 10 bits correspond to the skip pattern and the last 10 bits correspond to the control pattern. The skip pattern must have neutral disparity.

ve (volt encodes) are NRZ_L conditions where +ve encodes 0 and –ve encodes 1. ve is a running disparity (+/–RD) specifically used with the rate matcher. Depending on the ppm difference (which is defined by protocol) between the recovered clock and the local clock, the rate matcher adds or deletes a maximum of four skip patterns (neutral disparity). The net neutrality is conserved even after the skip word insertion or deletion because the control words alternate between positive and negative disparity.

In the following figure, the first skip cluster has a /K28.5/ control pattern followed by two /K28.0/ skip patterns. The second skip cluster has a /K28.5/ control pattern followed by four /K28.0/ skip patterns. The rate match FIFO deletes only one /K28.0/ skip pattern from the first skip cluster to maintain at least one skip pattern in the cluster after deletion. Two /K28.0/ skip patterns are deleted from the second cluster for a total of three skip patterns deletion requirement.

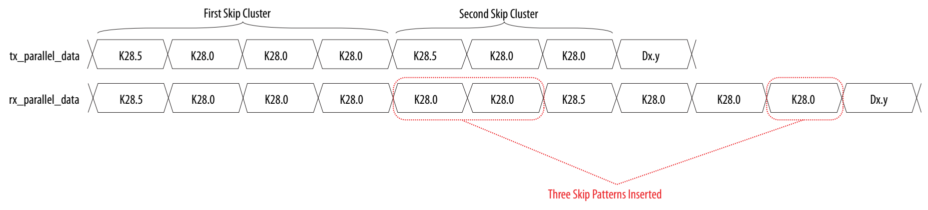
The rate match FIFO can insert a maximum of four skip patterns in a cluster, if there are no more than five skip patterns in the cluster after insertion.

Figure 44. Rate Match FIFO Deletion with Three Skip Patterns Required for Deletion



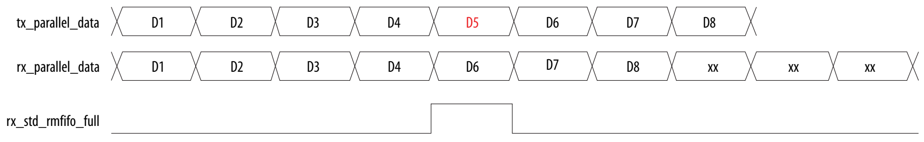
In the following figure, /K28.5/ is the control pattern and neutral disparity /K28.0/ is the skip pattern. The first skip cluster has a /K28.5/ control pattern followed by three /K28.0/ skip patterns. The second skip cluster has a /K28.5/ control pattern followed by two /K28.0/ skip patterns. The rate match FIFO inserts only two / K28.0/ skip patterns into the first skip cluster to maintain a maximum of five skip patterns in the cluster after insertion. One /K28.0/ skip pattern is inserted into the second cluster for a total of three skip patterns to meet the insertion requirement.

Figure 45. Rate Match FIFO Insertion with Three Skip Patterns Required for Insertion



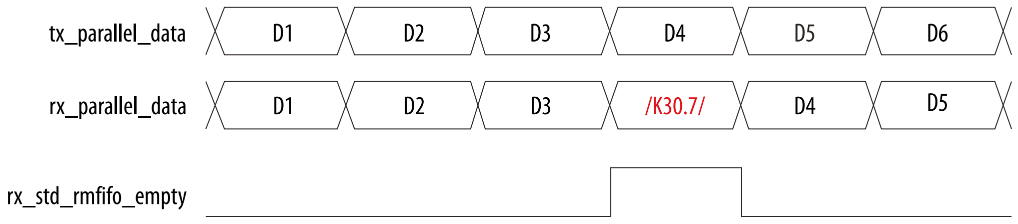
The following figure shows the deletion of **D5** when the upstream transmitter reference clock frequency is greater than the local receiver reference clock frequency. It asserts `rx_std_rmfifo_full` for one parallel clock cycle while the deletion takes place.

Figure 46. Rate Match FIFO Becoming Full After Receiving D5



The following figure shows the insertion of skip symbols when the local receiver reference clock frequency is greater than the upstream transmitter reference clock frequency. It asserts `rx_std_rmfifo_empty` for one parallel clock cycle while the insertion takes place.

Figure 47. Rate Match FIFO Becoming Empty After Receiving D3



Rate Match FIFO Basic (Double Width) Mode

1. Select **basic (double width)** in the **RX rate match FIFO mode** list.
2. Enter values for the following parameters.

Parameter	Value	Description
RX rate match insert/delete +ve pattern (hex)	20 bits of data specified as a hexadecimal string	The first 10 bits correspond to the skip pattern and the last 10 bits correspond to the control pattern. The skip pattern must have neutral disparity.
RX rate match insert/delete -ve pattern (hex)	20 bits of data specified as a hexadecimal string	The first 10 bits correspond to the skip pattern and the last 10 bits correspond to the control pattern. The skip pattern must have neutral disparity.

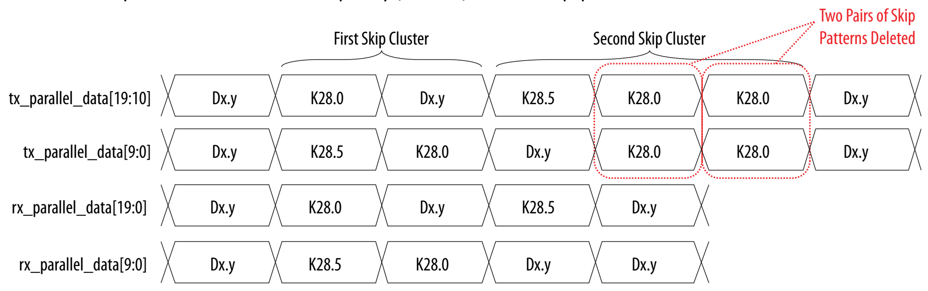
The rate match FIFO can delete as many pairs of skip patterns from a cluster as necessary to avoid the rate match FIFO from overflowing. The rate match FIFO can delete a pair of skip patterns only if the two 10-bit skip patterns appear in the same clock cycle on the LSByte and MSByte of the 20-bit word. If the two skip patterns appear straddled on the MSByte of a clock cycle and the LSByte of the next clock cycle, the rate match FIFO cannot delete the pair of skip patterns.

In the following figure, the first skip cluster has a /K28.5/ control pattern in the LSByte and /K28.0/ skip pattern in the MSByte of a clock cycle followed by one /K28.0/ skip pattern in the LSByte of the next clock cycle. The rate match FIFO cannot delete the two skip patterns in this skip cluster because they do not appear in the same clock cycle. The second skip cluster has a /K28.5/ control pattern in the MSByte of a clock cycle followed by two pairs of /K28.0/ skip patterns in the next two cycles. The rate match FIFO deletes both pairs of /K28.0/ skip patterns (for a total of four skip patterns deleted) from the second skip cluster to meet the three skip pattern deletion requirement.

The rate match FIFO can insert as many pairs of skip patterns into a cluster necessary to avoid the rate match FIFO from under running. The 10-bit skip pattern can appear on the MSByte, the LSByte, or both, of the 20-bit word.

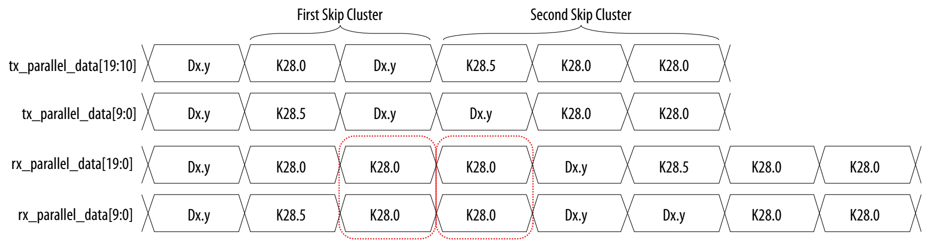
Figure 48. Rate Match FIFO Deletion with Four Skip Patterns Required for Deletion

/K28.5/ is the control pattern and neutral disparity /K28.0/ is the skip pattern.



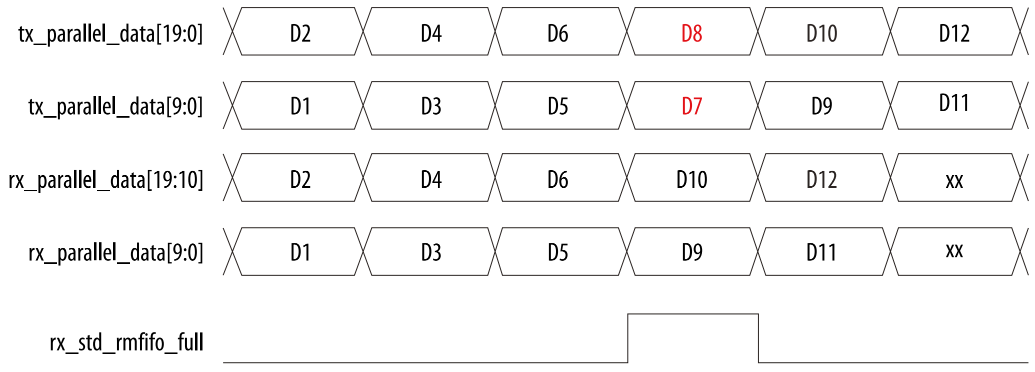
In the following figure, /K28.5/ is the control pattern and neutral disparity /K28.0/ is the skip pattern. The first skip cluster has a /K28.5/ control pattern in the LSByte and /K28.0/ skip pattern in the MSByte of a clock cycle. The rate match FIFO inserts pairs of skip patterns in this skip cluster to meet the three skip pattern insertion requirement.

Figure 49. Rate Match FIFO Insertion with Four Skip Patterns Required for Insertion



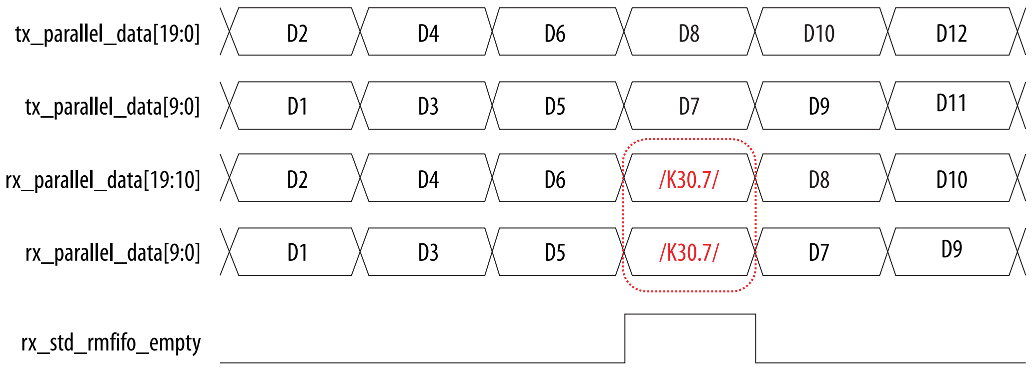
The following figure shows the deletion of the 20-bit word D7D8.

Figure 50. Rate Match FIFO Becoming Full After Receiving the 20-Bit Word D5D6



The following figure shows the insertion of two skip symbols.

Figure 51. Rate Match FIFO Becoming Empty After Reading out the 20-Bit Word D5D6



Rate Match FIFO for GbE

The rate match FIFO compensates frequency Part-Per-Million (ppm) differences between the upstream transmitter and the local receiver reference clock up to 125 MHz ± 100 ppm difference.

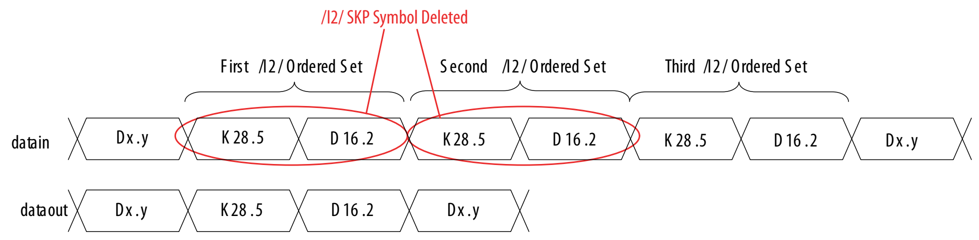
Note: 200 ppm total is only true if calculated as (125 MHz + 100 ppm) - (125 MHz - 100 ppm) = 200 ppm. By contrast, (125 MHz + 0 ppm) - (125 MHz - 200 ppm) is out of specification.

The GbE protocol requires the transmitter to send Idle ordered sets /I1/ (/K28.5/ D5.6/) and /I2/ (/K28.5/D16.2/) during inter-packet gaps (IPG) adhering to the rules listed in the IEEE 802.3-2008 specification.

The rate match operation begins after the synchronization state machine in the word aligner indicates synchronization is acquired by driving the `rx_syncstatus` signal high. The rate matcher deletes or inserts both symbols /K28.5/ and /D16.2/ of the /I2/ ordered sets as a pair in the operation to prevent the rate match FIFO from overflowing or underflowing. The rate match operation can insert or delete as many /I2/ ordered sets as necessary.

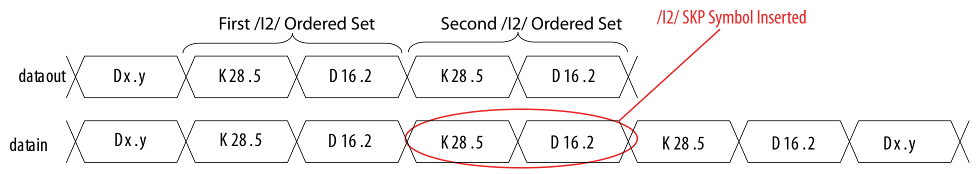
The following figure shows a rate match deletion operation example where three symbols must be deleted. Because the rate match FIFO can only delete /I2/ ordered sets, it deletes two /I2/ ordered sets (four symbols deleted).

Figure 52. Rate Match FIFO Deletion



The following figure shows an example of rate match FIFO insertion in the case where one symbol must be inserted. Because the rate match FIFO can only insert /I2/ ordered sets, it inserts one /I2/ ordered set (two symbols inserted).

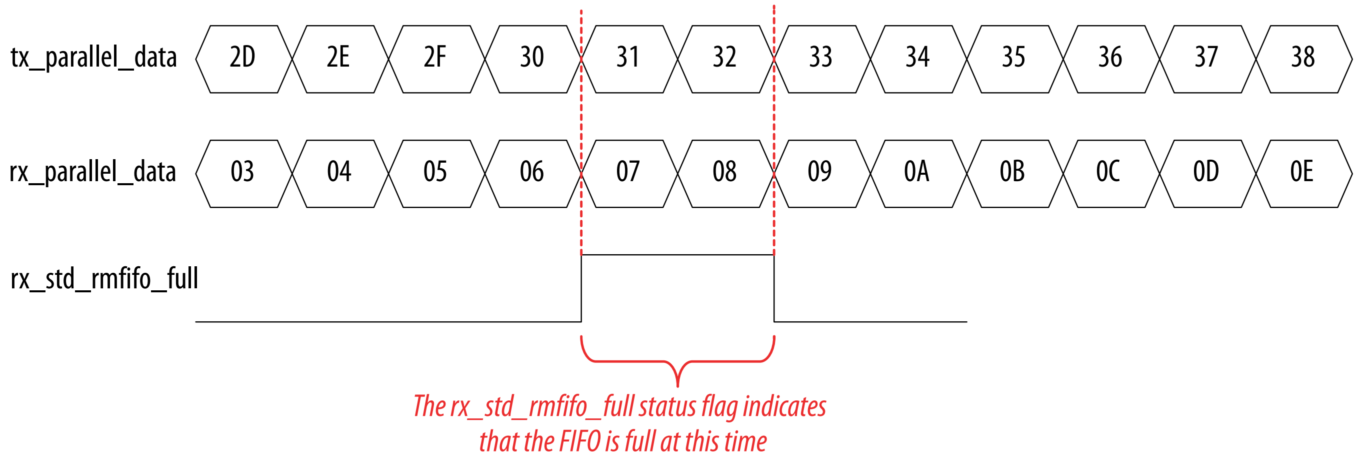
Figure 53. Rate Match FIFO Insertion



`rx_std_rmfifo_full` and `rx_std_rmfifo_empty` are forwarded to the FPGA fabric to indicate rate match FIFO full and empty conditions.

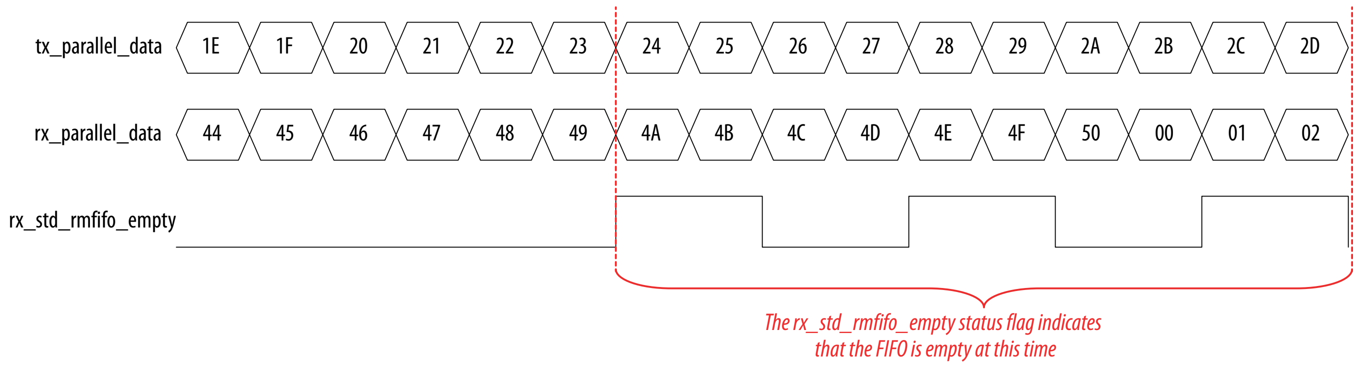
The rate match FIFO does not delete code groups to overcome a FIFO full condition. It asserts the `rx_std_rmfifo_full` flag for at least two recovered clock cycles to indicate rate match FIFO full. The following figure shows the rate match FIFO full condition when the write pointer is faster than the read pointer.

Figure 54. Rate Match FIFO Full Condition



The rate match FIFO does not insert code groups to overcome the FIFO empty condition. It asserts the `rx_std_rmfifo_empty` flag for at least two recovered clock cycles to indicate that the rate match FIFO is empty. The following figure shows the rate match FIFO empty condition when the read pointer is faster than the write pointer.

Figure 55. Rate Match FIFO Empty Condition



In the case of rate match FIFO full and empty conditions, you must assert the `rx_digitalreset` signal to reset the receiver PCS blocks.

Clock Compensation for PIPE

Refer to the *Gen1 and Gen2 Clock Compensation* and *Gen3 Clock Compensation* sections for more information.

Related Information

- Gen1 and Gen2 Clock Compensation on page 170
- Gen3 Clock Compensation on page 175

2.4.2.2.2. Clock Compensation Using the Enhanced PCS

For protocols such as 10GBASE-R, 40G/100G Ethernet, and so on, use the RX Core FIFO of the Enhanced PCS in 10GBASE-R Mode.

Figure 56. IDLE Word Insertion

This figure shows the insertion of IDLE words in the receiver data stream.

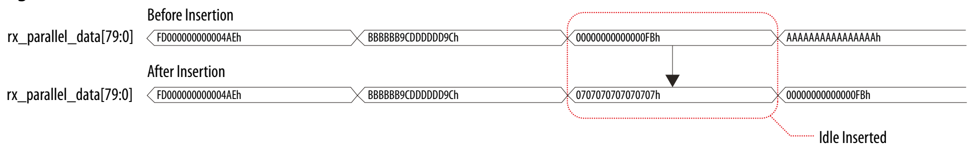


Figure 57. IDLE Word Deletion

This figure shows the deletion of IDLE words from the receiver data stream.

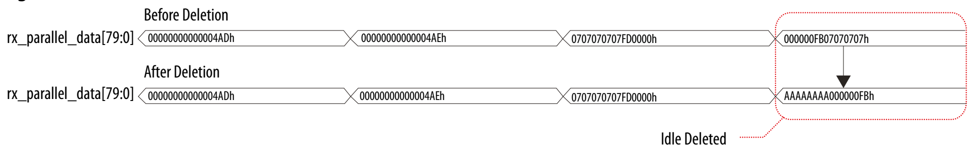
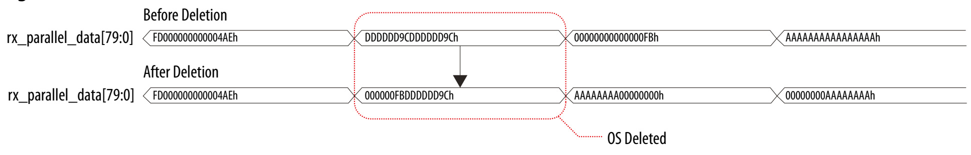


Figure 58. OS Word Deletion

This figure shows the deletion of Ordered set word in the receiver data stream.



Refer to the *10GBASE-R Mode* section for more information.

Related Information

10GBASE-R Mode on page 364

2.4.2.3. Encoding/Decoding

Use the 8B/10B encoder/decoder of the Standard PCS for protocols that require this encoding.

Similarly, use the 64B/66B encoder/decoder of the Enhanced PCS for protocols such as 10GBASE-R, 40G/100G Ethernet, and so on.

Refer to the *8B/10B Encoder Control Code Encoding*, *8B/10B Encoder Reset Condition*, and *8B/10B Encoder Idle Character Replacement Feature* sections for more information about the 8B/10B encoder.

Refer to the *8B/10B Decoder Control Code Encoding* section for more information about about the 8B/10B decoder.

Related Information

- 8B/10B Encoder Control Code Encoding on page 370
- 8B/10B Encoder Reset Condition on page 370
- 8B/10B Encoder Idle Character Replacement Feature on page 370
- 8B/10B Decoder Control Code Encoding on page 379

2.4.2.3.1. 8B/10B Encoder and Decoder

To enable the 8B/10B Encoder and the 8B/10B Decoder, select the **Enable TX 8B/10B Encoder** and **Enable RX 8B/10B Decoder** options on the **Standard PCS** tab in the IP Editor. Platform Designer allows implementing the 8B/10B decoder in **RX-only** mode.

The following ports are added:

- `tx_datak`
- `rx_datak`
- `rx_runningdisp`
- `rx_disperr`
- `rx_errdetect`

`rx_datak` and `tx_datak` indicate whether the parallel data is a control word or a data word. The incoming 8-bit data (`tx_parallel_data`) and the control identifier (`tx_datak`) are converted into a 10-bit data. After a power on reset, the 8B/10B encoder takes the 10-bit data from the RD- column. Next, the encoder chooses the 10-bit data from the RD+ column to maintain neutral disparity. The running disparity is shown by `rx_runningdisp`.

2.4.2.3.2. 8B/10B Encoding for GbE, GbE with IEEE 1588v2

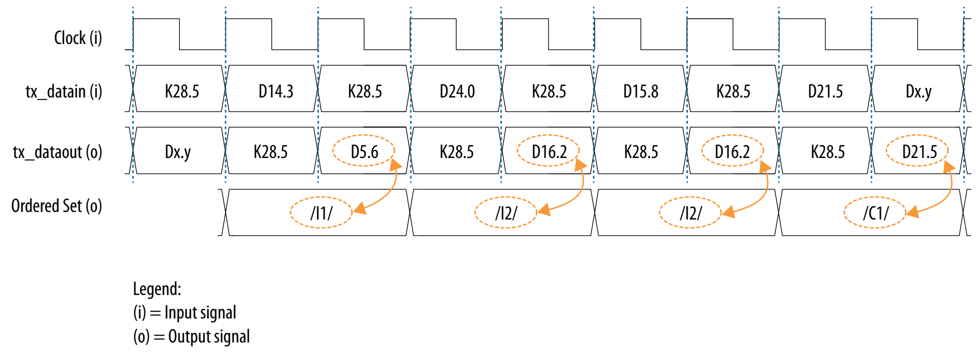
The 8B/10B encoder clocks 8-bit data and 1-bit control identifiers from the transmitter phase compensation FIFO and generates 10-bit encoded data. The 10-bit encoded data is sent to the PMA.

The IEEE 802.3 specification requires GbE to transmit Idle ordered sets (/I/) continuously and repetitively whenever the gigabit media-independent interface (GMII) is Idle. This transmission ensures that the receiver maintains bit and word synchronization whenever there is no active data to be transmitted.

For the GbE protocol, the transmitter replaces any /Dx.y/ following a /K28.5/ comma with either a /D5.6/ (/I1/ ordered set) or a /D16.2/ (/I2/ ordered set), depending on the current running disparity. The exception is when the data following the /K28.5/ is /D21.5/ (/C1/ ordered set) or /D2.2/ (/C2/) ordered set. If the running disparity before the /K28.5/ is positive, an /I1/ ordered set is generated. If the running disparity is negative, a /I2/ ordered set is generated. The disparity at the end of a /I1/ is the opposite of that at the beginning of the /I1/. The disparity at the end of a /I2/ is the same as the beginning running disparity immediately preceding transmission of the Idle code. This sequence ensures a negative running disparity at the end of an Idle ordered set. A /Kx.y/ following a /K28.5/ does not get replaced.

Note: /D14.3/, /D24.0/, and /D15.8/ are replaced by /D5.6/ or /D16.2/ (for I1 and I2 ordered sets). D21.5 (/C1/) is not replaced.

Figure 59. Idle Ordered-Set Generation Example

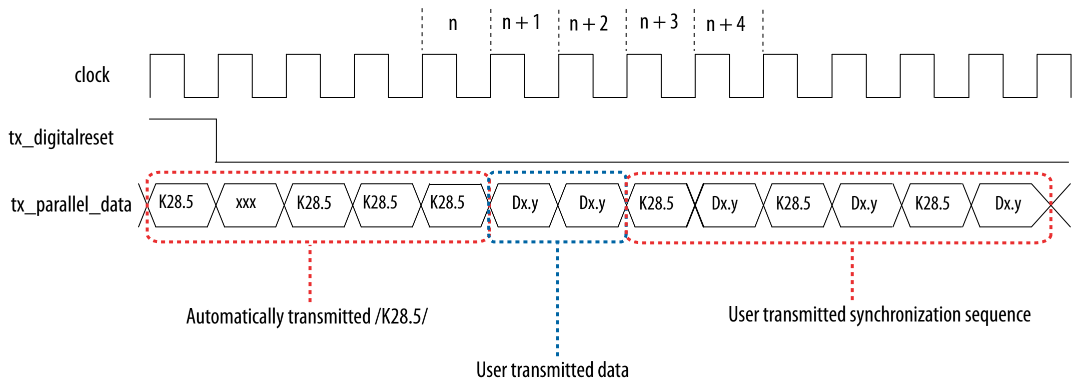


Reset Condition for 8B/10B Encoder in GbE, GbE with IEEE 1588v2

After deassertion of `tx_digitalreset`, the transmitters automatically transmit at least three /K28.5/ comma code groups before transmitting user data on the `tx_parallel_data` port. This transmission could affect the synchronization state machine behavior at the receiver.

Depending on when you start transmitting the synchronization sequence, there could be an even or odd number of /Dx.y/ code groups transmitted between the last of the three automatically sent /K28.5/ code groups and the first /K28.5/ code group of the synchronization sequence. If there is an even number of /Dx.y/code groups received between these two /K28.5/ code groups, the first /K28.5/ code group of the synchronization sequence begins at an odd code group boundary. The synchronization state machine treats this as an error condition and goes into the loss of synchronization state.

Figure 60. Reset Condition



2.4.2.3.3. KR-FEC Functionality for 64B/66B Based Protocols

You can use the KR-FEC block in the Enhanced PCS for both 10GBASE-KR/Ethernet and custom protocol implementation, provided that the protocol is 64B/66B based. This block is designed according to *IEEE802.3 Clause 74*, and can be used up to the maximum datarate of the transceiver channel.

For example, you can implement the Superlite II V2 protocol running four bonded lanes at 16 Gbps across a lossy backplane (close to 30 dB of IL at 8 GHz), and use the KR-FEC block in addition to RX equalization, to further reduce BER. Note that you incur additional latency that inherently occurs when using FEC. For the KR-FEC implementation mentioned in the example above, the latency is approximately an additional 40 parallel clock cycles for the full TX and RX path). The latency numbers depend on the actual line rate and other PCS blocks used for the protocol implementation. Refer to the Intel FPGA Wiki for more information about high speed transceiver demo designs.

Note: The material in the Intel FPGA Wiki is provided AS-IS and is not supported by Intel Corporation.

Refer to the *KR FEC Blocks* and *RX KR FEC Blocks* sections for more information about the KR-FEC blocks.

Refer to the *64B/66B Encoder and Transmitter State Machine (TX SM)* and *64B/66B Decoder and Receiver State Machine (RX SM)* sections for more information about the 64B/66B encoder and decoder.

Related Information

- KR FEC Blocks on page 359
- RX KR FEC Blocks on page 366
- 64B/66B Encoder and Transmitter State Machine (TX SM) on page 356
- 64B/66B Decoder and Receiver State Machine (RX SM) on page 361
- Intel FPGA Wiki

2.4.2.4. Running Disparity Control and Check

This is a function of the 8B/10B encoder block of the Standard PCS.

2.4.2.4.1. 8B/10B TX Disparity Control

The Disparity Control feature controls the running disparity of the output from the 8B/10B Decoder.

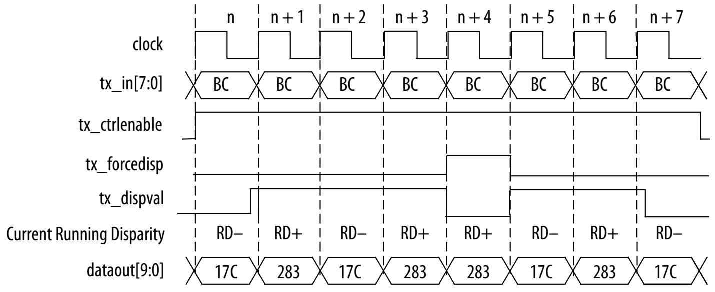
To enable TX Disparity Control, select the **Enable TX 8B/10B Disparity Control** option. The following ports are added:

- `tx_forcedisp`—a control signal that indicates whether a disparity value has to be forced or not
- `tx_dispval`—a signal that indicates the value of the running disparity that is being forced

When the number of data channels is more than 1, `tx_forcedisp` and `tx_dispval` are shown as buses in which each bit corresponds to one channel.

The following figure shows the current running disparity being altered in Basic single-width mode by forcing a positive disparity /K28.5/ when it was supposed to be a negative disparity /K28.5/. In this example, a series of /K28.5/ code groups are continuously being sent. The stream alternates between a positive running disparity (RD+) /K28.5/ and a negative running disparity (RD-) /K28.5/ to maintain a neutral overall disparity. The current running disparity at time n + 3 indicates that the /K28.5/ in time n + 4 should be encoded with a negative disparity. Because `tx_forcedisp` is high at time n + 4, and `tx_dispval` is low, the /K28.5/ at time n + 4 is encoded as a positive disparity code group.

Figure 61. 8B/10B TX Disparity Control



Refer to the *8B/10B Encoder Current Running Disparity Control Feature* section for more information about the 8B/10B data current running disparity control.

Refer to the *8B/10B Decoder Running Disparity Checker Feature* section for more information about the 8B/10B data current running disparity checker.

The Interlaken disparity generator and checker blocks of the Enhanced PCS support these functions. Refer to the *Interlaken Disparity Generator* and *Interlaken Disparity Checker* sections for more information about the Interlaken disparity generator and checker, respectively.

Related Information

- 8B/10B Encoder Control Code Encoding on page 370
- 8B/10B Decoder Running Disparity Checker Feature on page 379
- Interlaken Disparity Generator on page 358
- Interlaken Disparity Checker on page 361

2.4.2.5. FIFO Operation for the Enhanced PCS

2.4.2.5.1. Enhanced PCS FIFO Operation

Phase Compensation Mode

Phase compensation mode ensures correct data transfer between the core clock and parallel clock domains. The read and write sides of the TX Core or RX Core FIFO must be driven by the same clock frequency. The depth of the TX or RX FIFO is constant in this mode. Therefore, the TX Core or RX Core FIFO flag status can be ignored. You can tie `tx_fifo_wr_en` or `rx_data_valid` to logic level 1.

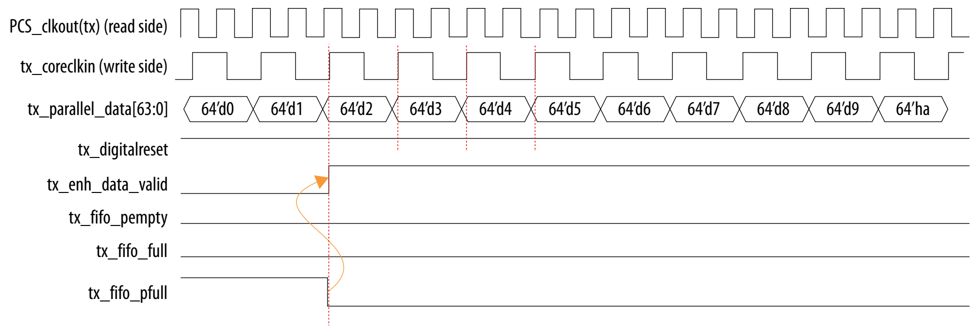
Basic Mode

Basic mode allows you to drive the write and read side of a FIFO with different clock frequencies. `tx_coreclkin` or `rx_coreclkin` must have a minimum frequency of the lane datarate divided by 66. The frequency range for `tx_coreclkin` or `rx_coreclkin` is (datarate/32) to (datarate/66). For best results, Intel recommends that `tx_coreclkin` or `rx_coreclkin` be set to (datarate/32). Monitor the FIFO flag to control write and read operations.

For TX FIFO, assert `tx_enh_data_valid` with the `tx_fifo_pfull` signal going low. This can be done with the following example assignment:

```
assign tx_enh_data_valid = ~tx_fifo_pfull;
```

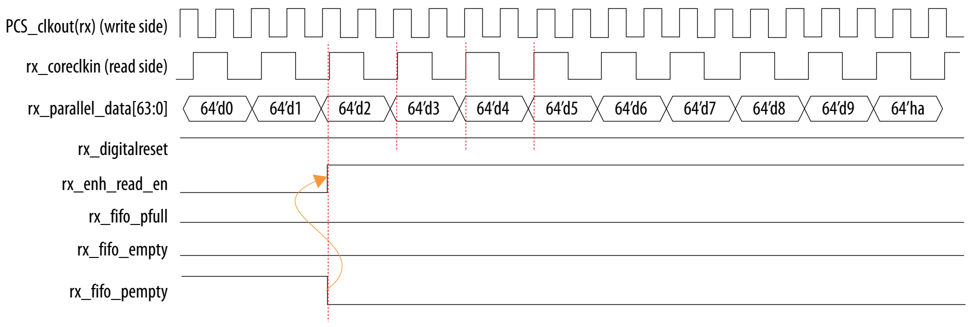
Figure 62. TX FIFO Basic Mode Operation



For RX FIFO, assert `rx_enh_read_en` with the `rx_fifo_pempty` signal going low. This can be done with the following example assignment:

```
assign rx_enh_read_en = ~rx_fifo_pempty;
```

Figure 63. RX FIFO Basic Mode Operation



If you are using even gear ratios, the `rx_enh_data_valid` signal is always high. For uneven gear ratios, `rx_enh_data_valid` switches between high and low. RX parallel data is valid when `rx_enh_data_valid` is high. Discard invalid RX parallel data when the `rx_enh_datavalid` signal is low.

Register Mode

In this mode, the FIFO is bypassed. The read and the write clock are the same.

10GBASE-R Configurations

In the 10GBASE-R configuration, the TX FIFO behaves as a phase compensation FIFO and the RX FIFO behaves as a clock compensation FIFO.

In the 10GBASE-R with 1588 configuration, both the TX FIFO and the RX FIFO are used in phase compensation mode. The TX and RX phase compensation FIFOs are constructed in the FPGA fabric by the PHY IP automatically.

In the 10GBASE-R with KR FEC configuration, use the TX FIFO in phase compensation mode and the RX FIFO behaves as a clock compensation FIFO.

2.4.2.6. Polarity Inversion

The positive and negative signals of a serial differential link might accidentally be swapped during board layout.

Solutions such as a board re-spin or major updates to the FPGA fabric logic can be costly. The TX and RX data polarity inversion features are available both in the Standard and Enhanced PCS to correct this situation.

The Standard PCS supports both the static and dynamic polarity inversion features. However, the Enhanced PCS only supports the static polarity inversion feature.

2.4.2.6.1. TX Data Polarity Inversion

Use the TX data polarity inversion feature to swap the positive and negative signals of a serial differential link if they were erroneously swapped during board layout.

To enable TX data polarity inversion when using the Enhanced PCS, select the **Enable TX data polarity inversion** option in the **Gearbox** section of the Native PHY IP core. Refer to the *TX Gearbox, TX Bitslip and Polarity Inversion* section for more information.

To enable transmitter polarity inversion in low latency, basic, and basic with rate match modes of the Standard PCS, perform the following actions in the Native PHY IP core.

- Select the **Enable TX polarity inversion** option
- Select the **Enable tx_polinv port** option

This mode adds `tx_polinv`. If there is more than one channel in the design, `tx_polinv` is a bus with each bit corresponding to a channel. Provided that `tx_polinv` is asserted, the TX data transmitted has a reverse polarity. Refer to the Polarity Inversion Feature section for more information.

Related Information

- TX Gearbox, TX Bitslip and Polarity Inversion on page 358
- Polarity Inversion Feature on page 371

2.4.2.6.2. RX Data Polarity Inversion

Use the RX data polarity inversion feature to swap the positive and negative signals of a serial differential link if they were erroneously swapped during board layout.

To enable RX data polarity inversion, when using the Enhanced PCS, select the **Enable RX data polarity inversion** option in the **Gearbox** section of the Native PHY IP core. Refer to the *RX Gearbox, RX Bitslip, and Polarity Inversion* section for more information.

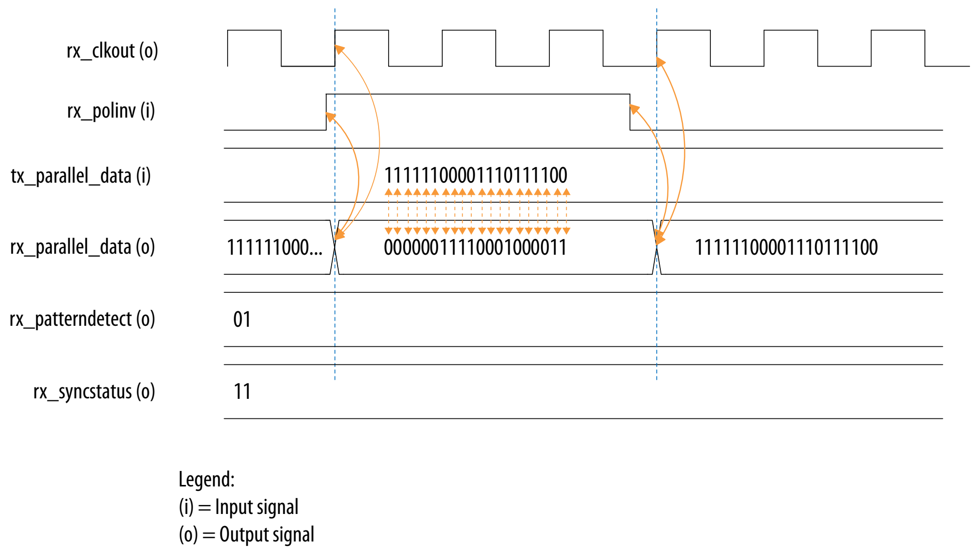
To enable receiver polarity inversion in Basic/Custom (Standard PCS), Basic/Customer w/ Rate Match (Standard PCS), and in Standard PCS low latency mode, perform the following actions in the Native PHY IP core:

- Select the **Enable RX polarity inversion** option
- Select the **Enable rx_polinv port** option

This mode adds `rx_polinv`. If there is more than one channel in the design,`rx_polinv` is a bus in which each bit corresponds to a channel. Provided that `rx_polinv` is asserted, the RX data received has a reverse polarity. You can verify this feature by monitoring `rx_parallel_data`.

Note: For PCS Direct, you may enable the static polarity inversion bits through register access. Refer to the *Logical View of the L-Tile/H-Tile Transceiver Registers* for details.

Figure 64. RX Polarity Inversion



Refer to the *RX Polarity Inversion Feature* section for more information.

Related Information

- RX Gearbox, RX Bitslip, and Polarity Inversion on page 360
- RX Polarity Inversion Feature on page 377
- Logical View of the L-Tile/H-Tile Transceiver Registers on page 450

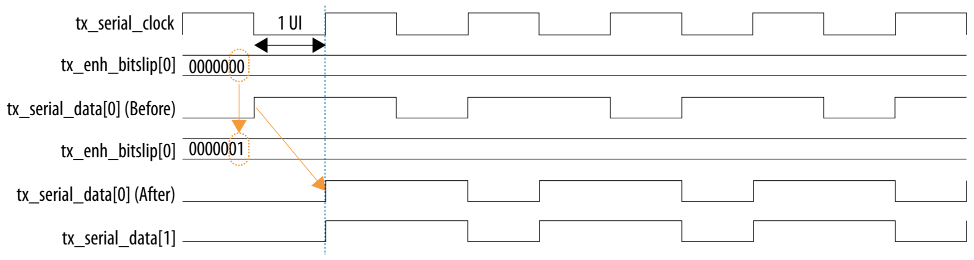
2.4.2.7. Data Bitslip

2.4.2.7.1. TX Data Bitslip

In the Enhanced PCS, the bit slip feature in the TX gearbox allows you to slip the transmitter bits before they are sent to the serializer.

The value specified on the TX bit slip bus indicates the number of bit slips. The minimum slip is one UI. The maximum number of bits slipped is equal to the FPGA fabric-to-transceiver interface width minus 1. For example, if the FPGA fabric-to transceiver interface width is 64 bits; the bit slip logic can slip a maximum of 63 bits. Each channel has six bits to determine the number of bits to slip. The TX bit slip bus is a level-sensitive port, so the TX serial data is bit slipped statically by TX bit slip port assignments. Each TX channel has its own TX bit slip assignment and the bit slip amount is relative to the other TX channels. You can improve lane-to-lane skew by assigning TX bit slip ports with proper values. The following figure shows the effect of slipping `tx_serial_data[0]` by one UI to reduce the skew with `tx_serial_data[1]`. After the bit slip `tx_serial_data[0]` and `tx_serial_data[1]` are aligned.

Figure 65. TX Bitslip



Refer to the *TX Gearbox, TX Bitslip and Polarity Inversion* section for more information.

When using the Standard PCS, select the **Enable TX bitslip** and **Enable tx_std_bitslipboundarysel port** options to use the TX bitslip feature. This adds the `tx_std_bitslipboundarysel` input port. The TX PCS automatically slips the number of bits specified by `tx_std_bitslipboundarysel`. There is no port for TX bit slip. If there is more than one channel in the design, `x_std_bitslipboundarysel` ports are multiplied by the number of channels. You can verify this feature by monitoring the `tx_parallel_data` port. Enabling the TX bit slip feature is optional.

Note: The `rx_parallel_data` values in the following figures are based on the TX and RX bit reversal features being disabled.

Figure 66. TX Bitslip in 8-bit Mode

`tx_parallel_data` = 8'hbc. `tx_std_bitslipboundarysel` = 5'b00001 (bit slip by 1 bit).

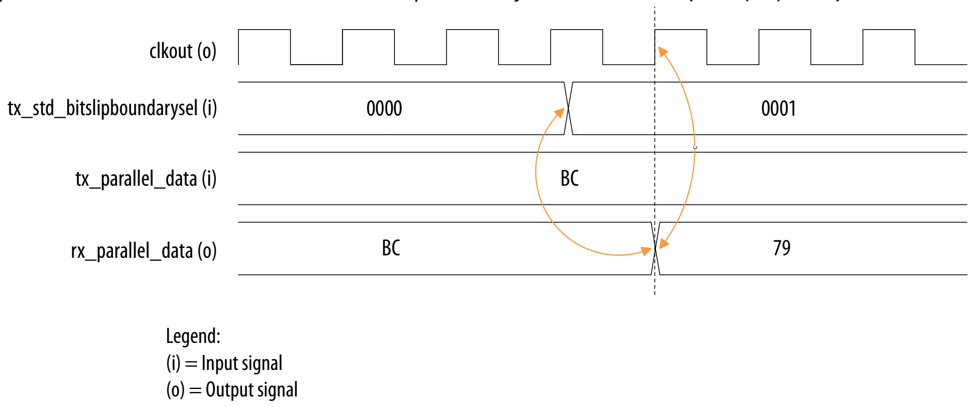


Figure 67. TX Bitslip in 10-bit Mode

`tx_parallel_data` = 10'h3bc. `tx_std_bitslipboundarysel` = 5'b00011 (bit slip by 3 bits).

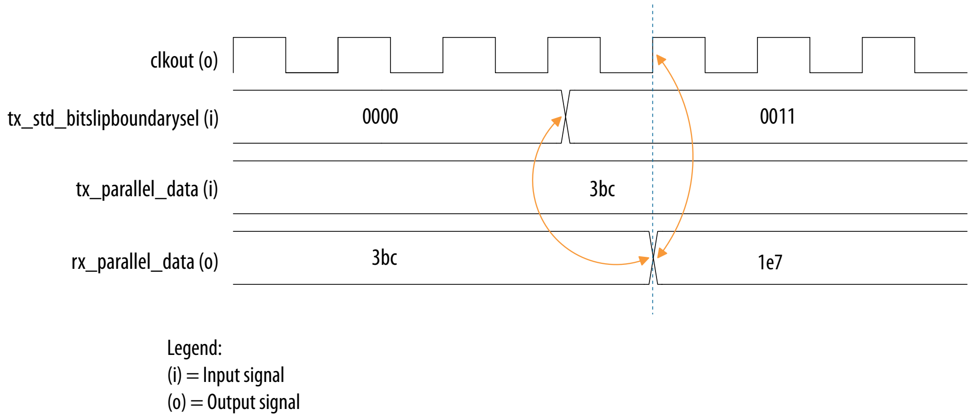


Figure 68. TX Bitslip in 16-bit Mode

`tx_parallel_data` = 16'hfcbc. `tx_std_bitslipboundarysel` =5'b00011 (bit slip by 3 bits).

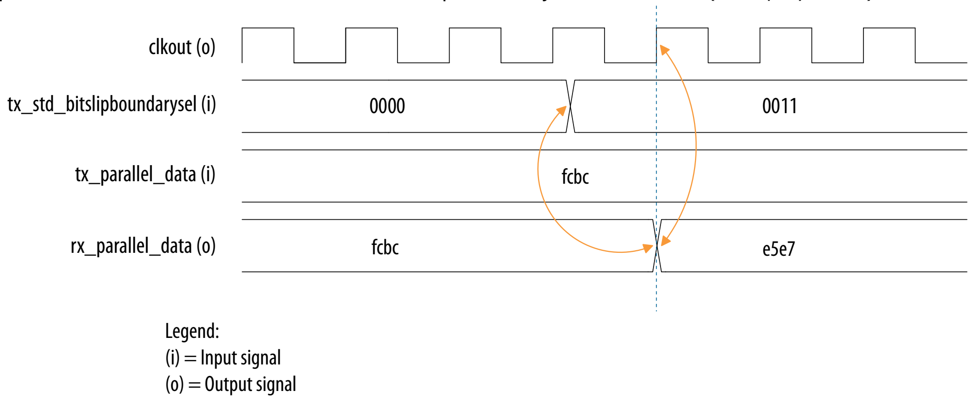
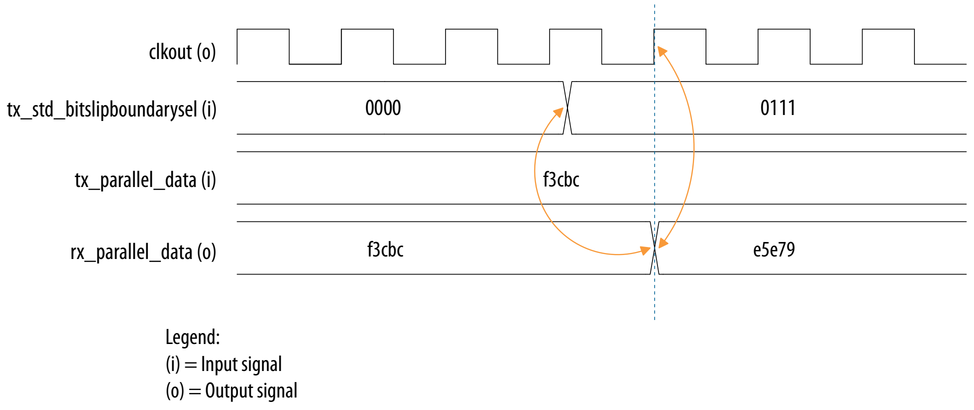


Figure 69. TX Bitslip in 20-bit Mode

`tx_parallel_data` = 20'hF3CBC. `tx_std_bitslipboundarysel` = 5'b00111 (bit slip by 7 bits).



Refer to the *TX Bit Slip* section for more information

Related Information

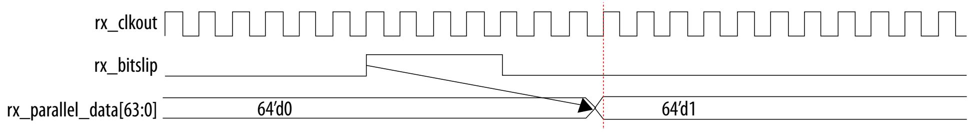
- TX Gearbox, TX Bitslip and Polarity Inversion on page 358
- TX Bit Slip on page 372

2.4.2.7.2. RX Data Bitslip

When using the Enhanced PCS, the RX data bitslip in the RX gearbox allows you to slip the recovered data.

An asynchronous active high edge on the `rx_bitslip` port changes the word boundary, shifting `rx_parallel_data` one bit at a time. Use the `rx_bitslip` port with its own word aligning logic. Assert the `rx_bitslip` signal for at least two parallel clock cycles to allow synchronization. You can verify the word alignment by monitoring `rx_parallel_data`. Using the RX data bitslip feature is optional.

Figure 70. RX Bitslip



Refer to the *RX Gearbox, RX Bitslip, and Polarity Inversion* section for more information.

To use the RX bitslip feature when using the Standard PCS, select **Enable rx_bitslip port** and set the word aligner mode to **bitslip**. This adds `rx_bitslip` as an input control port. An active high edge on `rx_bitslip` slips one bit at a time. When `rx_bitslip` is switched between high and low, the word aligner slips one bit at a time on every active high edge. Assert the `rx_bitslip` signal for at least two parallel clock cycles to allow synchronization. You can verify this feature by monitoring `rx_parallel_data`.

Figure 71. RX Bitslip in 8-bit Mode

`tx_parallel_data` = 8'hbc

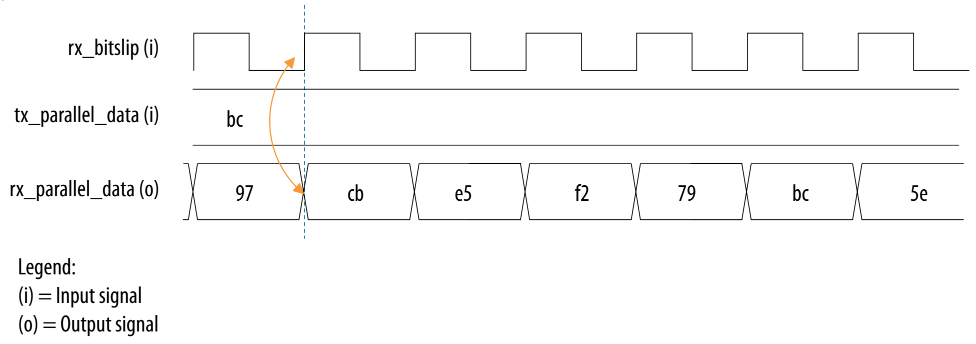


Figure 72. RX Bitslip in 10-bit Mode

`tx_parallel_data` = 10'h3bc

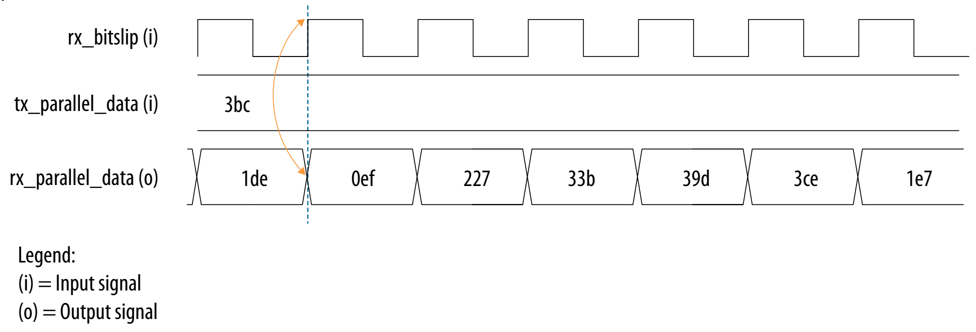


Figure 73. RX Bitslip in 16-bit Mode

`tx_parallel_data` = 16'hfcbc

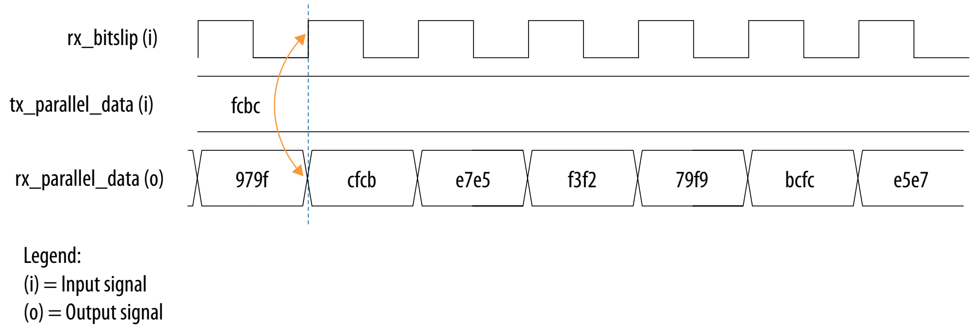
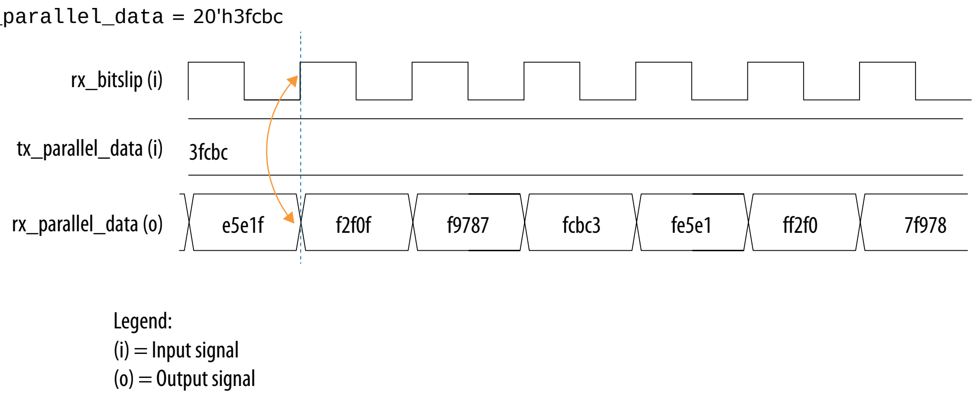


Figure 74. RX Bitslip in 20-bit Mode



Refer to the *Word Aligner Bitslip Mode* section for more information.

Related Information

- RX Gearbox, RX Bitslip, and Polarity Inversion on page 360
- Word Aligner Bitslip Mode on page 372

2.4.2.8. Bit Reversal

Refer to the Intel Stratix 10 (L/H-Tile) Word Aligner Bitslip Calculator to calculate the number of slips you require to achieve alignment based on the word alignment pattern and length.

Related Information

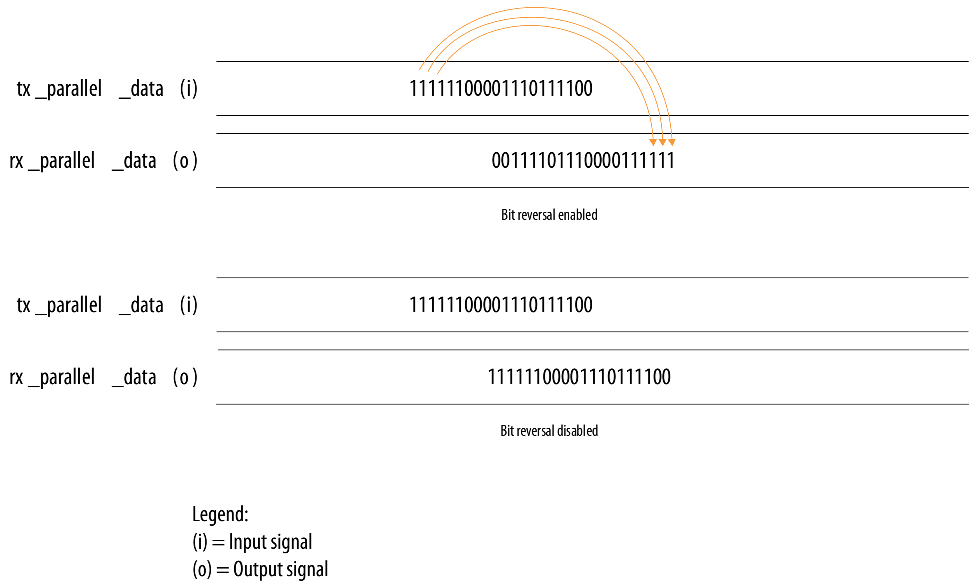
Intel Stratix 10 (L/H-Tile) Word Aligner Bitslip Calculator

2.4.2.8.1. Transmitter Bit Reversal

You can enable the TX bit reversal feature in Basic/Custom (Standard PCS), Basic/Custom w/ Rate Match (Standard PCS) and in Standard PCS low latency mode.

This feature is parameter-based, and creates no additional ports. If there is more than one channel in the design, all channels have TX bit reversal. To enable TX bit reversal, select the **Enable TX bit reversal** option in the Native PHY IP core.

Figure 75. TX Bit Reversal



You can implement soft logic to perform bit reversal for the Enhanced PCS.

Refer to the *8B/10B Encoder Bit Reversal Feature* section for more information.

Related Information

8B/10B Encoder Bit Reversal Feature on page 371

2.4.2.8.2. Receiver Bit Reversal

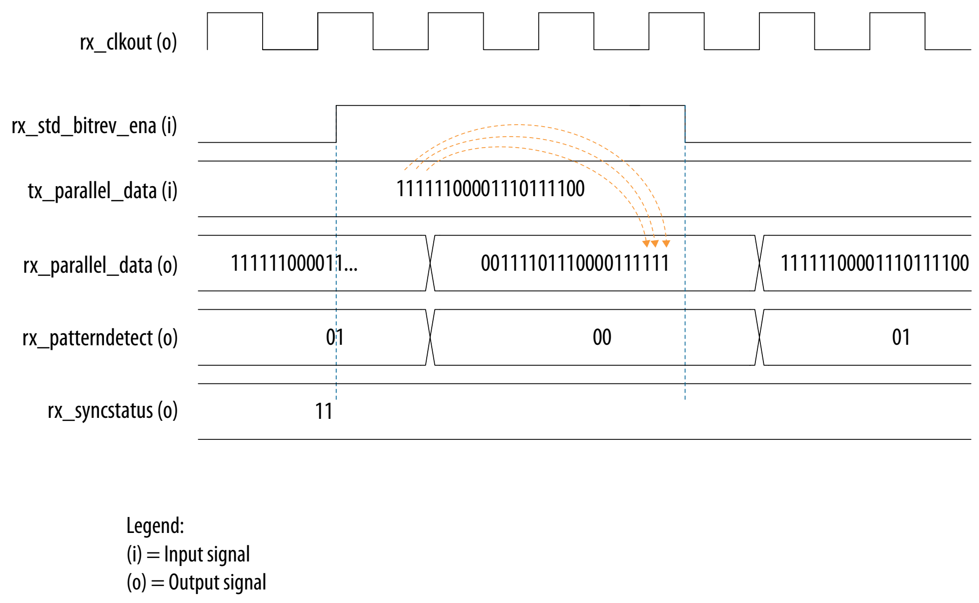
This is a function of the word alignment block available in the Standard PCS.

You can enable the RX bit reversal feature in Basic/Custom (Standard PCS), Basic/Custom w/ Rate Match (Standard PCS) and in Standard PCS low latency mode. The word aligner is available in any mode, bit slip, manual, or synchronous state machine.

To enable this feature, select the **Enable RX bit reversal** and **Enable rx_std_bitrev_ena port** options. This adds `rx_std_bitrev_ena`. If there is more than one channel in the design, `rx_std_bitrev_ena` becomes a bus in which each bit corresponds to a channel. Provided that `rx_std_bitrev_ena` is asserted, the RX data received by the core shows bit reversal.

You can verify this feature by monitoring `rx_parallel_data`.

Figure 76. RX Bit Reversal



You can implement soft logic to perform bit reversal for the Enhanced PCS.

Refer to the *Word Aligner RX Bit Reversal Feature* section for more information.

Related Information

Word Aligner RX Bit Reversal Feature on page 376

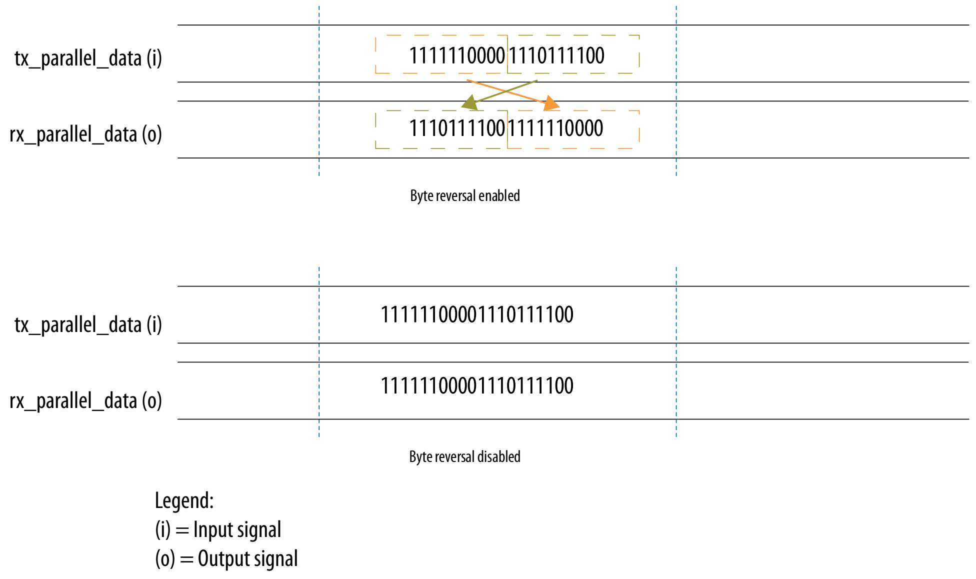
2.4.2.9. Byte Reversal

2.4.2.9.1. Transmitter Byte Reversal

You can enable the TX byte reversal feature in Basic/Custom (Standard PCS), Basic/ Custom w/ Rate Match (Standard PCS) and in Standard PCS low latency mode.

This feature is parameter-based, and creates no additional ports. If there is more than one channel in the design, all channels have TX byte reversal. To enable TX byte reversal, select the **Enable TX byte reversal** option in the Native PHY IP core.

Figure 77. TX Byte Reversal



You can implement soft logic to perform byte reversal for the Enhanced PCS.

Refer to the *8B/10B Encoder Byte Reversal Feature* section for more information.

Related Information

8B/10B Encoder Byte Reversal Feature on page 371

2.4.2.9.2. Receiver Byte Reversal

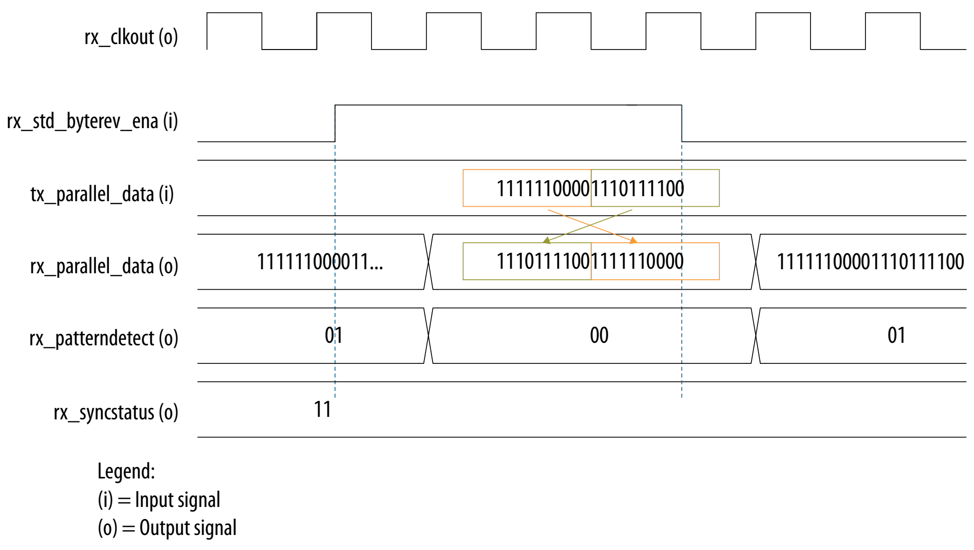
This is a function of the word alignment block available in the Standard PCS.

You can enable the RX byte reversal feature in Basic/Custom (Standard PCS), Basic/Custom w/ Rate Match (Standard PCS) and in Standard PCS low latency mode.

To enable this feature, select the **Enable RX byte reversal** and **Enable rx_std_byterev_ena port** options. This adds `rx_std_byterev_ena`. If there is more than one channel in the design, `rx_std_byterev_ena` becomes a bus in which each bit corresponds to a channel. As long as `rx_std_byterev_ena` is asserted, the RX data received by the core shows byte reversal.

You can verify this feature by monitoring `rx_parallel_data`.

Figure 78. RX Byte Reversal



You can implement soft logic to perform byte reversal for the Enhanced PCS.

Refer to the *Word Aligner RX Byte Reversal Feature* section for more information.

Related Information

Word Aligner RX Byte Reversal Feature on page 377

2.4.2.10. Double Rate Transfer Mode

Enable the double rate transfer mode option located in the **Datapath Options** tab in the Native PHY IP core to:

- Take advantage of the higher speeds of the Hyperflex architecture in the Intel Stratix 10 fabric core
- Achieve a comparative reduction of IP resource counts with similar IP cores

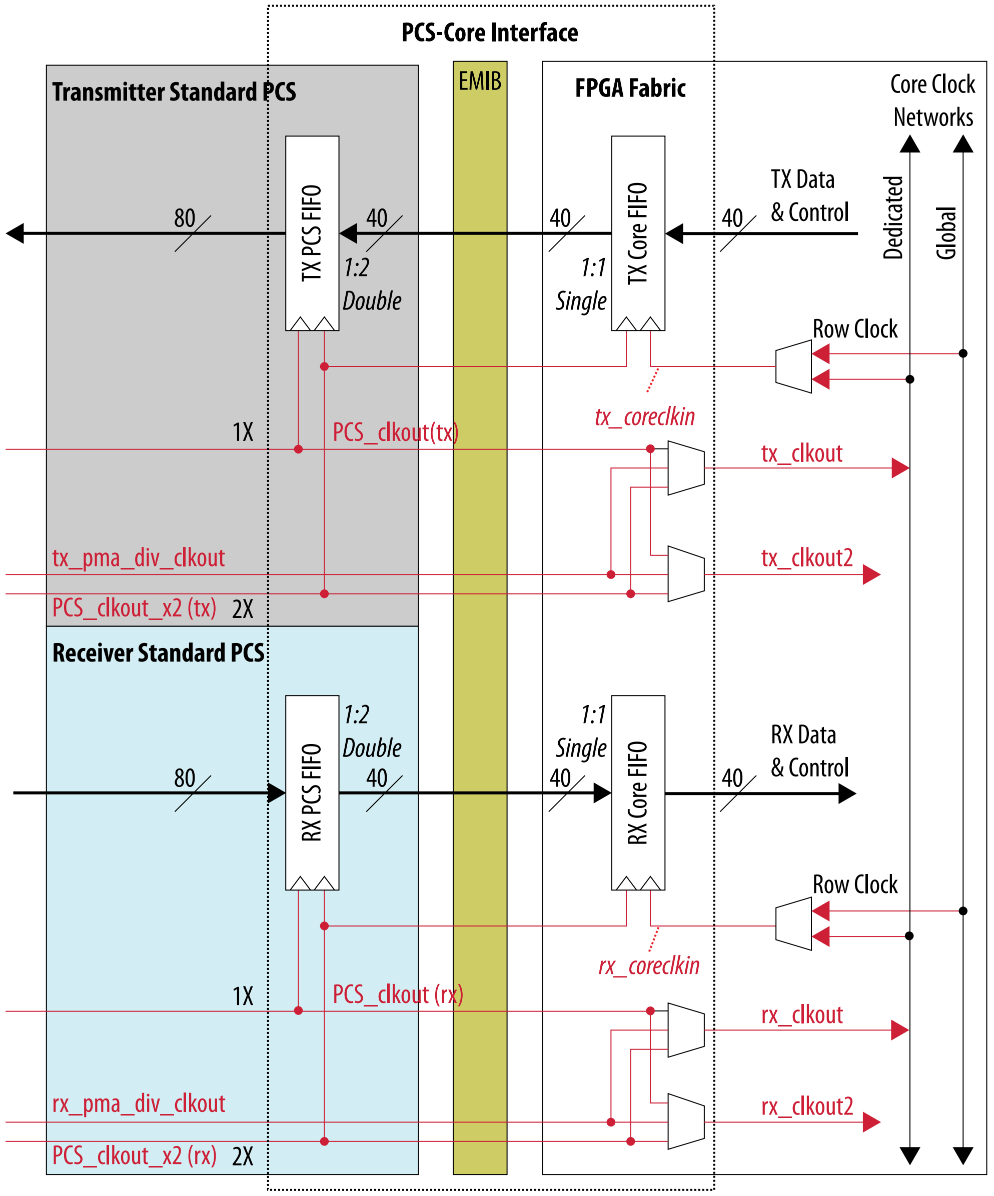
Double rate transfer means that the data width from the TX PCS FIFO to the PMA is double the data width coming from the FPGA fabric through the EMIB to the TX PCS FIFO. The write clock frequency is double the read clock of the TX PCS FIFO. Whereas the data width from FPGA Fabric to the TX Core FIFO is the same as the data width from the TX Core FIFO to the EMIB. The read and write clock frequencies of the TX Core FIFO are the same. At the RX side, the data width from the PMA to the RX PCS FIFO is double the data width coming from the RX PCS FIFO to the EMIB. The RX PCS FIFO read clock frequency is double the frequency of the write clock. Whereas the data width from the EMIB to the RX Core FIFO is the same as the data width from the RX Core FIFO to the FPGA Fabric. The read and write clock frequencies of the RX Core FIFO are the same.

When this mode is enabled, the PCS parallel data is split into two words. Each word is transferred to and from the transceiver at twice the parallel clock frequency. You can enable the double rate transfer mode for almost all configurations except for the following:

- PCS FIFO data width ≤ 10 bit
- Core FIFO data width ≤ 10 bit

When double rate transfer mode is enabled, select **PCS clkout x2** in the **TX Clock Options** and **RX Clock Options** in the **PCS-Core Interface** tab of the Native PHY IP Parameter Editor. There is one exception. When using the TX standard PCS with PMA or PCS data width = 20 and byte-serializer = OFF, set PCS_`clk_2x` = x1, and you must provide a x2 clock generated from the fPLL to drive `tx_coreclkin2` for double rate transfer. There is a checkbox you can select, which enables this port on the IP Parameter Editor.

Figure 79. Double Rate Transfer Mode Clocking and Datapath



Disabling or enabling double rate transfer mode changes the parallel data mapping. Refer to the *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* section for detailed data mapping information.

Related Information

Transceiver PHY PCS-to-Core Interface Reference Port Mapping on page 86

2.4.2.10.1. Word Marking Bits

A word marking bit is only required when using double rate transfer mode.

The maximum possible FIFO datapath width in the Intel Stratix 10 transceiver is 40 bits wide. To transfer 80 bits of `tx_parallel_data` or `rx_parallel_data` (includes data bus and control bits) across the datapath, the parallel data is divided into two data words of 40 bits each. A new marking bit is added to indicate the word boundary in `tx_parallel_data` or `rx_parallel_data`, respectively, to mark the lower 40-bit word and the upper 40-bit word.

When channels are configured in double rate transfer mode, you must set the word marking bit of `tx_parallel_data` to 0 or 1 to indicate the lower or upper 40-bit word on the transmit datapath. On the receive datapath, either the upper or lower word may be received first. You must use the marking bits to realign the data. On the receive datapath, the word marking bits also indicate the lower or the upper word. Usually it is the same as on the transmit datapath (where 0 is the lower word and 1 is the upper word). However, there are some exceptions. For the following configurations, the upper word is received at 0 and the lower word at 1:

- Enhanced PCS, with interface width of 32 bits
- PCS Direct, with interface width of 16, 20 and 32 bits

There is a special reset sequence involving the word marking bit that is required when using double rate transfer mode. Refer to the *Special TX PCS Reset Release Sequence* section for more information.

Refer to the *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* section for marking bit information.

Related Information

- Transceiver PHY PCS-to-Core Interface Reference Port Mapping on page 86
- Double Rate Transfer Mode enabled on page 328

2.4.2.10.2. How to Implement Double Rate Transfer Mode

You should be familiar with the *Standard PCS Architecture*, *Enhanced PCS Architecture*, *PLL Architecture*, and *Transceiver Native PHY PCS-to-Core Interface Reference Port Mapping* before implementing double rate transfer mode.

1. Instantiate the **Stratix 10 L-Tile/H-Tile Transceiver Native PHY IP** from the IP Catalog (**Installed IP > Library > Interface Protocols > Transceiver PHY > Stratix 10 L-Tile/H-Tile Transceiver Native PHY**).
2. Select **Enable double rate transfer mode** located under **Datapath Options**.

   Note: If you enable double rate transfer mode, you cannot enable the simplified data interface.
3. In the **TX Clock Options** area, select **PCS clkout x2** from the **Selected tx_clkout clock source** pull-down menu.
4. In the **RX Clock Options** area, select **PCS clkout x2** from the **Selected rx_clkout source** pull-down menu.

5. In the **TX Clock Options** area, select **Enable tx_clkout2 port** and use PCS clkout as the source. Having this second clock port gives you access to the full-rate clock for your full-width data.
6. In the **RX Clock Options** area, select **Enable rx_clkout2 port** and use PCS clkout as the source. Having this second clock port gives you access to the full-rate clock for your full-width data.
7. Configure the **TX FIFO partially full threshold** and **RX FIFO partially full threshold** values accordingly in **PCS-Core Interface** tab.
8. Configure the **TX FIFO partially empty threshold** and **RX FIFO partially empty threshold** values accordingly in **PCS-Core Interface** tab.
9. If you are using the Enhanced PCS, configure both the **Enhanced PCS / PMA interface width** and **FPGA fabric / Enhanced PCS interface width** settings accordingly.
10. If you are using the Standard PCS, configure the **Standard PCS / PMA interface width** accordingly.
11. Click **Generate HDL** to generate the Native PHY IP Core (this is your RTL file).
12. Implement the **Special TX PCS Reset Sequence** for your specific configuration. Refer to Resetting Transceiver Channels for more information.

Related Information

- Resetting Transceiver Channels on page 318
- Double Rate Transfer Mode enabled on page 328
- Transceiver PHY PCS-to-Core Interface Reference Port Mapping on page 86
- PLLs and Clock Networks on page 249
- Enhanced PCS Architecture on page 353
- Intel Stratix 10 Standard PCS Architecture on page 366

2.4.2.11. Asynchronous Data Transfer

In Intel Stratix 10 devices, a list of asynchronous sideband and control signals are transferred between the transceiver and the FPGA Fabric using shift register chains. There are two categories of shift registers

- Fast shift register (FSR)
- Slow shift register (SSR)

The FSR has a shorter register chain and is used to transfer signals that are more timing-critical. The SSR has a longer register chain and is used for signals that are less timing-critical.

Refer to the *Transceiver PHY PCS-to-Core Interface Reference Port Mapping* section for the list of FSR and SSR signals that are transferred using shift register chain

Related Information

Transceiver PHY PCS-to-Core Interface Reference Port Mapping on page 86

2.4.2.11.1. FPGA Fabric to Transceiver Transfer

For signals that are coming from FPGA Fabric into the transceiver, there is capture logic in the transceiver which captures the signals before sending them to the shift registers.

The sampling time of the capture logic is relative to the length of the shift register chain. To ensure the signals are successfully sampled and loaded into the register chain, you must hold those signals for a minimum period of time, depending on the type of shift register chain used to transfer the signals.

Table 88. Register Chain Minimum Hold Time Calculations

Register Chain	Minimum Hold Cycles	Minimum Hold Time [21]
FSR	10	1.667*10 = 16.67 ns
SSR	120	1.667*120 = 200.04 ns

Given that the internal oscillator clock frequency can vary between 600 MHz and 900 MHz in the hardware, Intel recommends that you hold the signals for the worst case scenario of 600 MHz as summarized in the table above.

2.4.2.11.2. Transceiver to FPGA Fabric Transfer

For signals that are going out from the transceiver to the FPGA Fabric, there is update logic in the transceiver that updates the signal level of the interface ports following the shift register update cycle.

There are minimum frequency requirements for FSR and SSR signals as listed below. You must capture the signals using a clock that is ≥ the frequency in the table below.

Table 89. Register Chain Minimum Sampling Frequency

Register Chain	Minimum Sampling Frequency (Without Using the Hard IP) [22]	Minimum Sampling Frequency (Using the Hard IP) [22]
FSR	225 MHz	113 MHz
SSR	10.98 MHz	10 MHz

Given that the internal oscillator clock frequency can vary between 600 MHz and 900 MHz in the hardware, use a sampling frequency for the worst case scenario of 900 MHz as summarized above.

2.4.2.12. Low Latency

Related Information

- PLLs on page 251
- Using PLLs and Clock Networks on page 304
- Resetting Transceiver Channels on page 318
- Intel Stratix 10 Standard PCS Architecture on page 366

[21] The calculation assumes an OSC divider factor of 1.

[22] The calculation assumes an OSC divider factor of 1.

2.4.2.12.1. How to Enable Low Latency in Basic (Standard PCS)

In the Native PHY IP Core, use the following settings to enable low latency:

1. Select the **Enable 'Standard PCS' low latency mode** option.
2. Select either **low_latency** or **register FIFO** in the **TX FIFO mode** list.
3. Select either **low_latency** or **register FIFO** in the **RX FIFO mode** list.
4. Select either **Disabled** or **Serialize x2** in the **TX byte serializer mode** list.
5. Select either **Disabled** or **Serialize x2** in the **RX byte deserializer mode** list.
6. Ensure that **RX rate match FIFO mode** is **disabled**.
7. Set the **RX word aligner mode** to **bitslip**.
8. Set the **RX word aligner pattern length** to **7** or **16**.

   Note: TX bitslip, RX bitslip, bit reversal, and polarity inversion modes are supported.

2.4.2.12.2. How to Enable Low Latency in Basic (Enhanced PCS)

In the Native PHY IP Core, use the following settings to enable low latency:

1. Select the **Enable 'Enhanced PCS' low latency mode** option.
2. Select one of the following gear ratios:

   32:32, **40:40**, **64:64**, **40:64**, **32:64**, **64:66**
3. Select **Phase compensation** from the **TX Core Interface FIFO mode** and **RX PCS-Core Interface FIFO mode (PCS FIFO-Core FIFO)** pull-down menus.
4. Enable the `tx_clkout2` and `rx_clkout2` ports.
5. Select **PCS clkout** in the **Selected tx_clkout source** and **Selected rx_clkout source** fields.
6. Select **PCS clkout** in the **Selected tx_clkout2 source** and **Selected rx_clkout2 source** fields.

Related Information

- Intel Stratix 10 Standard PCS Architecture on page 366
- PLLs on page 251
- Resetting Transceiver Channels on page 318
- Using PLLs and Clock Networks on page 304

2.4.3. Deterministic Latency Use Model

You can use the phase-measuring FIFO method to measure the deterministic latency for Intel Stratix 10 L-Tile and H-Tile transceivers. The phase-measuring FIFOs replace the traditional register mode, and are expected to be widely used.

2.4.3.1. Phase-measuring FIFOs

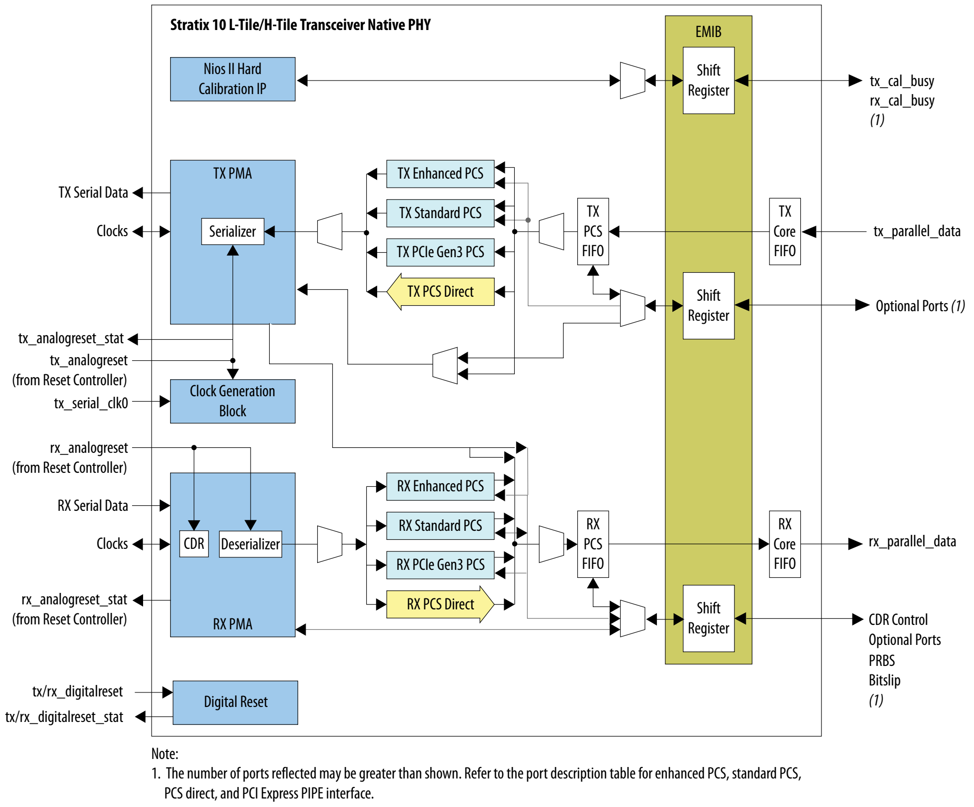
2.4.3.1.1. Primary Use Model

To support higher speed transfer rates and the new EMIB fabric introduced between the FPGA fabric and the L-Tile/H-Tile, Intel Stratix 10 devices replace the traditional register mode transfer used for deterministic latency (CPRI, IEEE 1588) with a set of phase compensation FIFOs that allow you to measure its delay. The phase compensation FIFOs include:

- TX PCS FIFO
- TX Core FIFO
- RX PCS FIFO
- RX Core FIFO

The TX PCS and Core FIFOs comprise the PCS-Core Interface on the TX side. Similarly, the RX PCS and Core FIFOs comprise the PCS-Core interface on the RX side.

Figure 80. PCS-Core Port Interface



Note:
1. The number of ports reflected may be greater than shown. Refer to the port description table for enhanced PCS, standard PCS, PCS direct, and PCI Express PIPE interface.

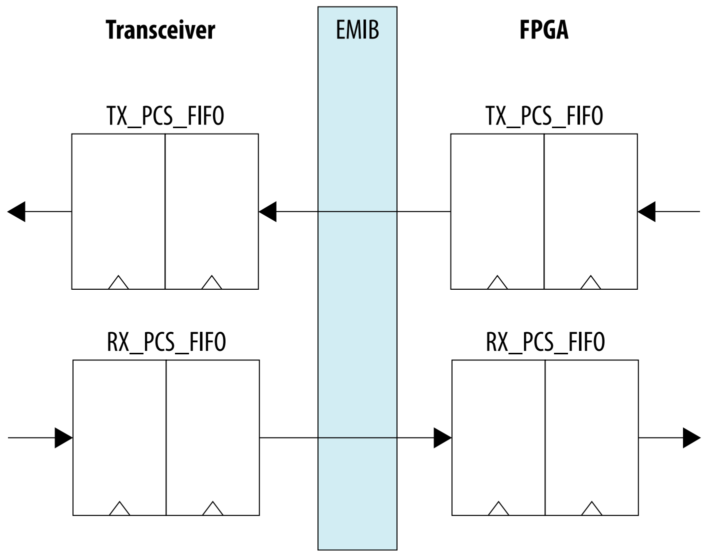
The Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY IP core allows you to configure all these FIFOs in phase compensation mode for deterministic latency applications (CPRI, IEEE1588 etc.,). It provides the following ports to measure latency through the TX PCS FIFO, TX Core FIFO, RX PCS FIFO, and RX Core FIFO, respectively:

- `tx_pcs_fifo_latency_pulse`
- `tx_fifo_latency_pulse`
- `rx_pcs_fifo_latency_pulse`
- `rx_fifo_latency_pulse`
- `latency_sclk`

You must select the **Enable latency measurement ports** option in the **Latency Measurement Ports** section of the **PCS-Core Interface** panel of the Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY IP core, to enable these ports. All ports with the exception of `latency_sclk` are output ports. The `latency_sclk` port is an input to the Intel Stratix 10 H-Tile Transceiver Native PHY IP core. Set the four FIFOs in phase compensation mode.

The four FIFOs associated with the PCS-Core interface allow for measurement of their latency to sub-cycle accuracy. Each FIFO can output a `latency_pulse` that is 1 or 0, proportional to how full the FIFO is. For example, if there is a FIFO that is 8 words deep and the FIFO is 4.5 words full, the `latency_pulse` is a 1 (4.5/8) = 56% of the time.

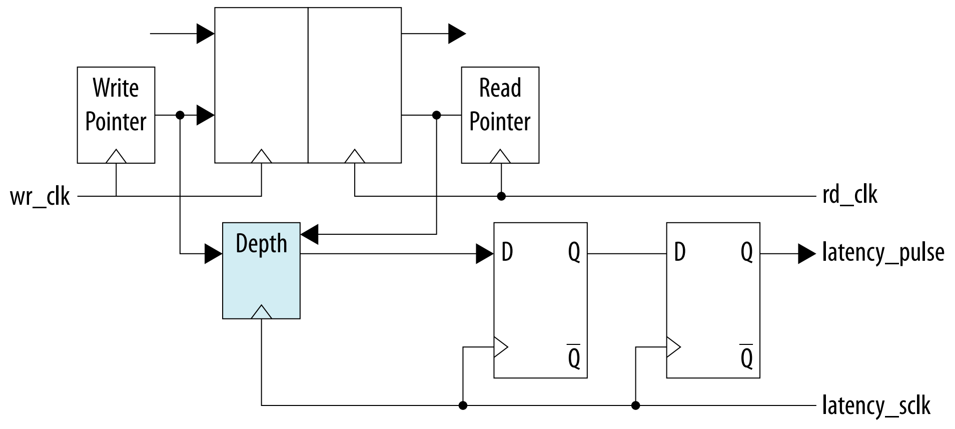
Figure 81. Phase-measuring FIFO



This measurement pulse is sampled via a `sample_clock` that can run up to 262 MHz. Meta-stable hardening to this clock is done within the hard logic.

Note: Refer to *Deterministic Latency* for more information about how to calculate latency across FIFOs.

Figure 82. Phase-measuring FIFO Block Diagram



To measure the fullness of the FIFO, you must run your sampling clock at a rate that is not equal to the parallel clock. For example, `parallel_clock` * (128/127) or `parallel_clock` * (64/127) so that the sampling clock sweeps various phase relationships relative to the parallel clock. You must determine via a simple counter how often the resulting pulse is a 1 versus a 0.

The phase measuring circuit is designed to work in the case of a phase compensation FIFO and in the case of a phase compensation FIFO where the read and write pointers may have an exact 2:1 ratio.

The Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY IP core shows you the default maximum depth of the FIFO for the mode chosen.

Table 90. Potential Depths for Each FIFO

These are the possible depths of each FIFO, and not the default spacing of the counters. The depth selected depends on the mode of the FIFO. When in a 1:2 or a 2:1 mode, depth should be defined as the maximum value that the counter may take for the 2x clock.

FIFO	Modes	Description
TX Core FIFO (FPGA Fabric side)	8 words deep 16 words deep 32 words deep	Refer to the **System Messages** of Native PHY IP core to determine the default FIFO depth based on the mode/ configuration selected in the IP GUI.
TX PCS FIFO (transceiver side)	8 words deep 16 words deep	Refer to the **System Messages** of Native PHY IP core to determine the default FIFO depth based on the mode/ configuration selected in the IP GUI.
RX PCS FIFO (transceiver side)	8 words deep 16 words deep	Refer to the **System Messages** of Native PHY IP core to determine the default FIFO depth based on the mode/ configuration selected in the IP GUI.
RX Core FIFO (FPGA Fabric side)	8 words deep 16 words deep 64 words deep	Refer to the **System Messages** of Native PHY IP core to determine the default FIFO depth based on the mode/ configuration selected in the IP GUI.

Refer to the *FIFO Latency Calculation* section for details about usage and examples of the deterministic latency port.

Related Information

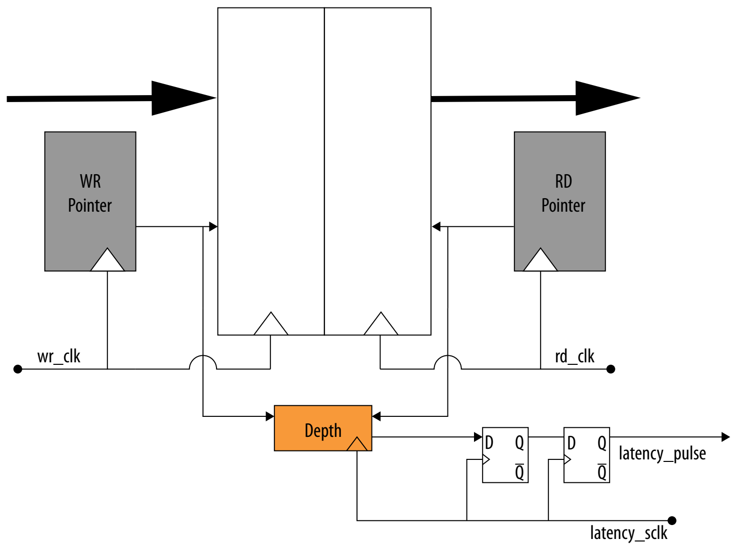
- FIFO Latency Calculation on page 150
- Deterministic Latency on page 233

2.4.3.1.2. FIFO Latency Calculation

Latency through FIFOs (in PC mode) may vary from reset to reset. You can determine that variation by using the appropriate FIFO latency calculation logic. The phase measuring technique measures the distance between read and write counters. The phase measuring circuit is designed to work in the case of a phase comp FIFO and in the case of a phase comp FIFO where the read and write pointers may have an exact 2:1 ratio.

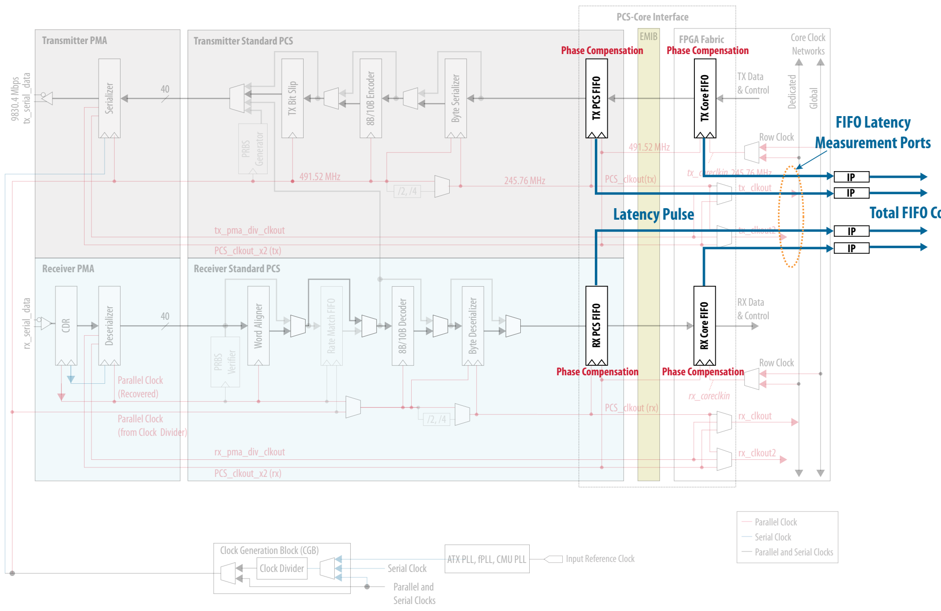
To measure the fullness of the FIFO, run your sampling clock at a rate that is not equal to the parallel clock. The guideline for calculating the sampling clock is that the sampling clock sweeps various phase relationships relative to the parallel clock. Using a simple counter, you should determine how often the resulting pulse is a 1 versus a 0. You can find this implementation in the FIFO latency calculation logic referenced below.

Figure 83. Phase Measuring FIFO



Below is an example implementation of the FIFO latency calculation logic. Each FIFO in phase compensation mode has a latency measurement port (read only) available at the PCS-Core interface. Each latency measurement port supplies the latency pulse whose duty cycle depends on the FIFO latency. This latency pulse is further processed in the FPGA fabric to calculate FIFO latency. There are four similar pulses for the four FIFOs, and you must perform latency calculations on all of them separately. The IP represented in the following figure represents the logic in the FPGA fabric that performs the latency calculation. This IP is not a part of the Native PHY IP core, so you must instantiate it exclusively. Output of the IP is the **Total FIFO count** bus. This bus produces the result of the FIFO latency calculation. This result can only be read when the calculation is complete, and is denoted by the `result_ready` signal coming out of the IP. Refer to Figure 87 on page 152.

Figure 84. FIFO Latency Calculation



Once `result_ready` is asserted, you can read data on the **Total FIFO count** bus and calculate latency time using the following equation.

Figure 85. FIFO Latency Formula

$$\text{FIFO Latency (nsec)} = \frac{(\text{Total FIFO count}) * (\mathit{LATENCY\_SCLK}\ \text{period})}{\text{Number of pulses}}$$

The latency pulse available at a latency measurement port of a FIFO appears as shown below. The duty cycle of the pulse depends on the difference between the read and write pointers at a particular instant. The duty cycle of a pulse varies with time as both read and write pointers move. To find the actual latency through the FIFO, the IP block calculates the average of latency pulse duty cycles for an appropriate duration.

As seen in the figure below, a sampling clock (LATENCY_SCLK) samples the pulse. After the sampling duration, all the samples of 1s are accumulated and appear at the **Total FIFO count** output bus, which is then used in Figure 85 on page 151 to calculate latency (in nsec). The **Number of pulses** parameter is the actual number of latency pulses for which the latency calculation logic executes. This parameter defines the execution time of the latency calculation logic. The design file referenced below defaults to 128 latency pulses to calculate FIFO latency. Due to hardware limitations, the sampling clock cannot be higher than 260 MHz.

Figure 86. FIFO Latency Measurement Port

Number of pulses and the LATENCY_SCLK period vary depending on your particular implementation.

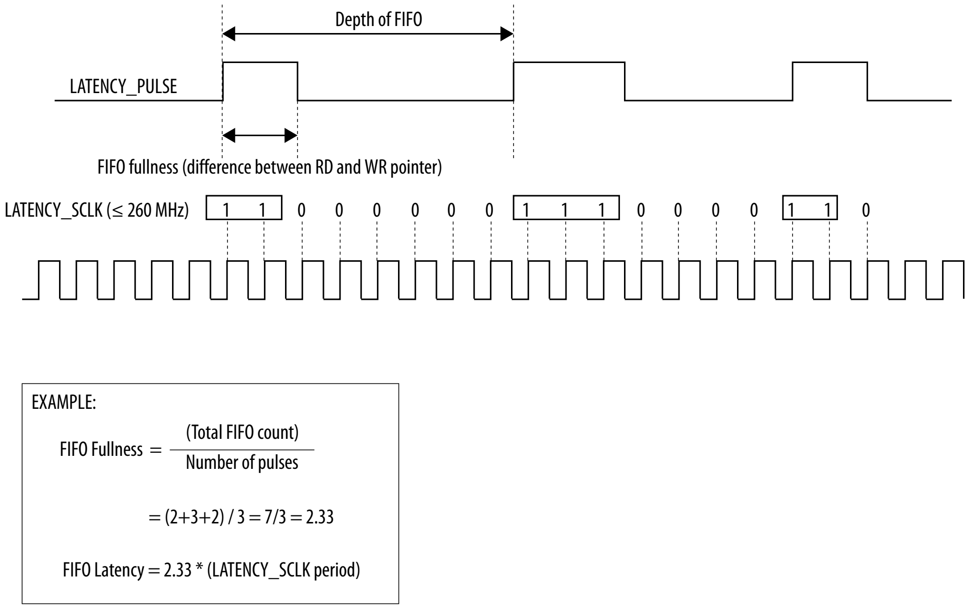

Depth of FIFO
LATENCY_PULSE
FIFO fullness (difference between RD and WR pointer)
LATENCY_SCLK (≤ 260 MHz)
1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 0
EXAMPLE:
FIFO Fullness = (Total FIFO count) / Number of pulses
= (2+3+2) / 3 = 7/3 = 2.33
FIFO Latency = 2.33 * (LATENCY_SCLK period)

Figure 87. FIFO Latency Calculation Logic

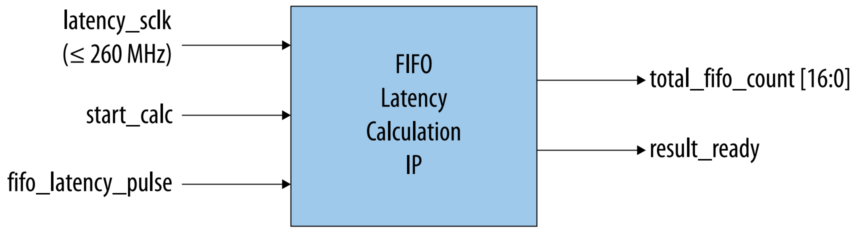


Table 91. FIFO Latency Calculation IP Signals

Symbol	Input/Output	Description
`latency_sclk`	Input	Sampling clock required by the latency calculation logic
`start_calc`	Input	Control signal to start the latency calculation
`fifo_latency_pulse`	Input	Latency pulse
`total_fifo_count`	Output	Result from the latency calculation logic
`result_ready`	Output	Status signal that signifies completion of the latency calculation logic

Figure 88. FIFO Latency Calculation Logic for Four FIFOs

Point A is the assertion of `start_calc` signals, and point B is an assertion of `result_ready` signals. The `total_fifo_count` signal is valid after `result_ready` is asserted.

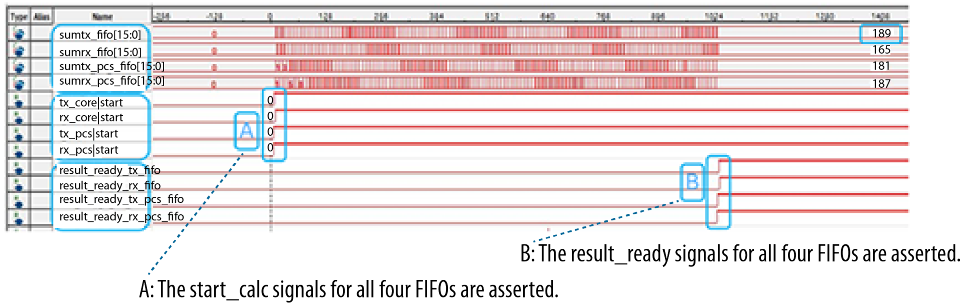


Figure 89. FIFO Latency Formula for all FIFOs

$$\text{TX Core FIFO (sumtx\_fifo) Latency} = \frac{\text{(Total FIFO count) * (LATENCY\_SCLK period)}}{\text{Number of pulses}}$$

$$= \frac{189}{128 * 247.69\text{ MHz}}$$

$$= 5.96\text{ nsec}$$

Follow these guidelines for the preferred frequency for `latency_sclk`:

- `latency_sclk` must be ≤ 260 MHz.
- The ratio of parallel clock to `latency_sclk`, when expressed in lowest terms, should have denominator of at least 50, but preferably, more than 100. The ideal resultant `latency_sclk` should be between 200 MHz to 260 MHz.
- Do not use a `latency_sclk` frequency that is a multiple of `tx_clkout`.
- Examples:

Table 92. Parallel Clock to `latency_sclk` Ratio Scenarios

`tx_clkout` Frequency	Correct `latency_sclk` Frequency	Incorrect `latency_sclk` Frequency
150 MHz	209 MHz	160 MHz
245 MHz	247 MHz	250 MHz
300 MHz	247 MHz	260 MHz

You can create `latency_sclk` using the IOPLL or the external reference clock.

Related Information

Design file for FIFO latency calculation logic

2.4.4. Debug Functions

Intel Stratix 10 transceivers contain features that are useful in debugging and characterizing high-speed serial links. They include hardened pattern generators and verifiers, loopback modes and On-Die Instrumentation (ODI) for monitoring height and eye width.

Note: ODI is fully supported for H-Tile devices. However, for L-Tile devices, ODI is a functional diagnostic utility for remote system debug and link tuning, in other words, up to 25.8 Gbps. The ODI feature does not support relative channel-to-channel comparisons on L-Tile devices.

2.4.4.1. Pattern Generators and Verifiers

Use the pattern generators and verifiers to simulate traffic and easily characterize high-speed links without fully implementing any upper protocol stack layer.

2.4.4.1.1. Pattern Generator and Verifier Use Model

The pattern generator and verifier are shared between the Standard and Enhanced datapaths through the PCS. Therefore, they have only one set of control signals and registers. Either the data from the PCS or the data generated from the pattern generator can be sent to the PMA at any time.

Because of this, you must save the settings of the registers corresponding to the PRBS generators and verifiers before enabling them if you want to disable them later. You can enable the pattern generators and verifier through registers outlined in the *Logical View of the L-Tile/H-Tile Transceiver Registers*. Note that you must reconfigure the EMIB to use the PRBS Verifier. To disable the pattern generators and verifiers, write the original values back relevant attribute addresses. The pattern generators and verifiers are supported only for non-bonded channels.

You have PRBS control and status signals available to the core. The Transceiver Toolkit also provides an easy way to use the PRBS generator and verifier along with the PRBS soft accumulators.

Related Information

Logical View of the L-Tile/H-Tile Transceiver Registers on page 450

2.4.4.1.2. Square Wave Pattern Generator

The square wave generator has programmable n-number of consecutive serial bit 1s and 0s. Refer to the *Intel Stratix 10 L-Tile/H-Tile Transceiver PHY Architecture* section for more details.

Related Information

Intel Stratix 10 L-Tile/H-Tile Transceiver PHY Architecture on page 343

2.4.4.1.3. PRBS Generator and Verifier

The PRBS generator generates a self-aligning pattern and covers a known number of unique sequences. Because the PRBS pattern is generated by a Linear Feedback Shift Register (LFSR), the next pattern can be determined from the previous pattern. When the PRBS verifier receives a portion of the received pattern, it can generate the next sequence of bits to verify whether the next data sequence received is correct.

The PRBS generator and verifier can only be used with either a 10-bit or 64-bit PCS-PMA interface. PRBS9 is available in 10-bit and 64-bit PCS-PMA widths. All other PRBS patterns are available in 64-bit PCS-PMA width only. Because the FPGA fabric-PCS interface must run within the recommended speed range of the FPGA core, ensure that you are using the correct PCS-PMA width for your corresponding datarate so as not to go above this range.

Table 93. PRBS Supported Polynomials and Data Widths

Use the 10-bit mode of PRBS9 when the datarate is lower than 3 Gbps.

Pattern	Polynomial	64-Bit	10-Bit	Best Use
PRBS7	$G(x) = 1 + x^6 + x^7$	Yes	—	Use PRBS7 and PRBS9 to test transceiver links with linear impairments, and with 8B/10B.
PRBS9	$G(x) = 1 + x^5 + x^9$	Yes	Yes	
PRBS15	$G(x) = 1 + x^{14} + x^{15}$	Yes	—	Use PRBS15 for jitter evaluation.
PRBS23	$G(x) = 1 + x^{18} + x^{23}$	Yes	—	Use PRBS23 or PRBS31 for jitter evaluation (data-dependent jitter) of non-8B/10B links, such as SDH/SONET/OTN jitter testers. Most 40G, 100G, and 10G applications use PRBS31 for link evaluation.
PRBS31	$G(x) = 1 + x^{28} + x^{31}$	Yes	—	

Related Information

- PRBS Pattern Generator on page 390
- PRBS Pattern Verifier on page 390

2.4.4.1.4. PRBS Control and Status Ports

Within the **RX PMA** tab of the Native PHY IP core, you can enable the following control and status ports to use the internal PRBS verifier:

- `rx_prbs_done`—Indicates the PRBS sequence has completed one full cycle. It stays high until you reset it with `rx_prbs_err_clr`.
- `rx_prbs_err`—Goes high if an error occurs. This signal is pulse-extended to allow you to capture it in the RX FPGA CLK domain.
- `rx_prbs_err_clr`—Used to reset the `rx_prbs_err` signal.

Refer to Table 50 on page 63 for more details on their functions. Refer to the *Transceiver PHY PCS-to-Core Interface Port Mapping* section for the exact bit location. Refer to the *Asynchronous Data Transfer* section for requirements on these signals when simplified interface is not enabled.

Related Information

- Configuring the Native PHY IP Core on page 34
- Transceiver PHY PCS-to-Core Interface Reference Port Mapping on page 86
- Asynchronous Data Transfer on page 144

2.4.4.1.5. Enabling and Disabling the PRBS Generator and PRBS Verifier

You can enable and disable the PRBS generator and verifier through registers outlined in the *Logical View of the L-Tile/H-Tile Transceiver Registers*.

Before enabling the PRBS generator and verifier, note the existing configuration of the attributes you are overwriting by reading out the register values beforehand.

Related Information

Logical View of the L-Tile/H-Tile Transceiver Registers on page 450

2.4.4.2. PRBS Soft Accumulators

This feature allows you to accumulate bits and measure a bit error rate (BER) when using the PRBS generator and verifier.

2.4.4.2.1. PRBS Soft Accumulators Use Model

You can select the Enable PRBS soft accumulators option by clicking **Dynamic Reconfiguration > Optional Reconfiguration Ports**. Refer to the *Logical View of the L-Tile/H-Tile Transceiver Registers* for the necessary register sequences to use the soft accumulators. The Transceiver Toolkit also provides an easy way to use the PRBS generator and verifier along with the PRBS soft accumulators.

PRBS soft accumulators are word-based counters. The value read from the PRBS soft accumulators represents the number of words counted. Therefore, to obtain the total accumulated bits, you must multiply the value read from the accumulated bits passed through the count [49:0] registers with the width of the PCS-PMA interface. For accumulated error count [49:0] registers, it counts 1 provided that there are bit errors in a word (either one bit in a word is erroneous or all the bits in a word are erroneous). Because of this, the accumulated error count [49:0] registers do not provide absolute bit errors counted. For each count, the absolute bit errors can range from one to the width of the PCS-PMA interface.

2.4.4.3. Loopback

Each channel has the following loopback modes:

- Serial loopback
- Pre-CDR reverse serial loopback
- Post-CDR reverse serial loopback

These modes help you test individual blocks by looping in your own TX data into the RX so you can test your RX data. Alternatively, you can loop your RX data back out of your TX so you can view it on a scope. Regardless of your datapath configuration, you can enable any one of these loopback modes to debug and test your system. Refer to the *Loopback Modes* section for details on these paths.

2.4.4.3.1. Enabling and Disabling Loopback

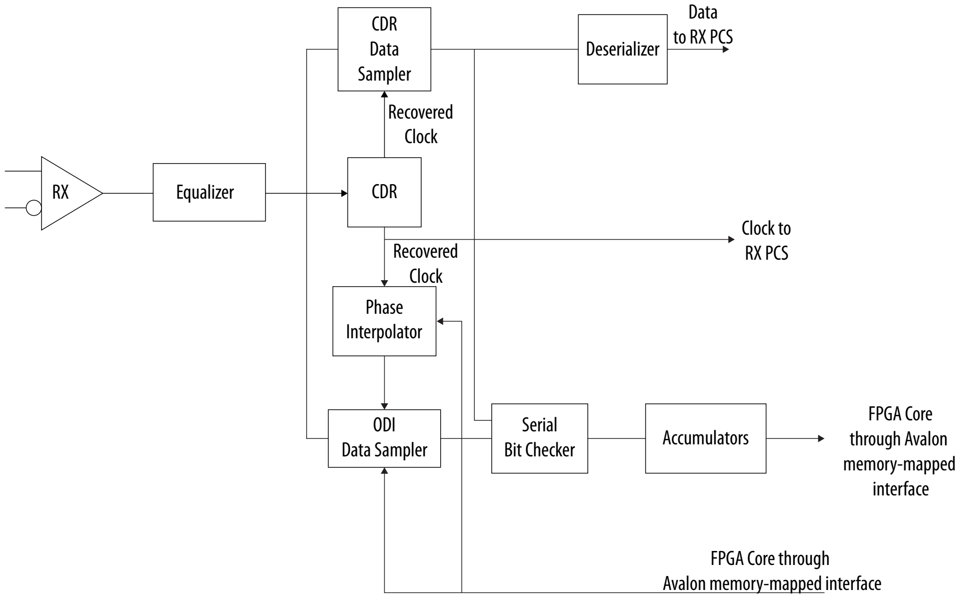
You can enable and disable serial loopback using the `rx_seriallpbken` input port that can be enabled using the **RX PMA** tab of the Native PHY IP Core.

Additionally, you can enable or disable serial loopback and the other loopback modes using the registers outlined in the *Logical View of the L-Tile/H-Tile Transceiver Registers* section.

2.4.4.4. On-die Instrumentation

You can use the Transceiver Toolkit to monitor the eye width and eye height of the signal, or you can also use the Avalon memory-mapped interface through sequences outlined in the *Logical View of the L-Tile/H-Tile Transceiver Registers*.

Figure 90. On-die Instrumentation Serial Bit Checker



Note: ODI is fully supported for H-Tile devices. However, for L-Tile devices, ODI is a functional diagnostic utility for remote system debug and link tuning, in other words, up to 25.8 Gbps. The ODI feature does not support relative channel-to-channel comparisons on L-Tile devices.

2.4.4.4.1. On-die Instrumentation Overview

The ODI block works by sweeping the horizontal phase values and vertical voltage values and comparing it against the recovered data to capture the eye opening.

The clock data recovery (CDR) unit's recovered clock is fed through a phase interpolator, which has 128 possible resolutions covering 2 UI. The phase interpolator's output clocks the ODI data sampler, which compares the receiver input after the RX equalizer's value with a voltage reference (64 levels for the top half of the eye and 64 levels for the bottom half of the eye). You can access both the phase interpolator and the ODI data sampler's voltage reference from the FPGA core through the Native PHY IP core's Avalon memory-mapped interface. The output of the ODI's data sampler is compared with the CDR data sampler through a serial bit checker. When the DFE is enabled, you must configure the serial bit checker to check for four different data patterns because the DFE is speculative. The number of bits tested and the number of error bits caught in the serial bit checker is summed in an accumulator. You can access the accumulator's output to the FPGA core through the Native PHY IP core's Avalon memory-mapped interface. The ODI implementation allows you to measure the bit error rate (BER) of live traffic.

2.4.4.4.2. How to Enable ODI

This procedure enables ODI through the Avalon memory-mapped interface, allowing you to view the eye.

1. If the device is H-tile production and background calibration is enabled, disable the background calibration:
   a. Set `0x542[0]` to `0x0`.

   b. Read `0x481[2]` until it becomes `0x0`.
2. If the RX is in adaptation mode[23], set `0x148[0]` to `0x1`. Otherwise, skip this step.
3. Set `0x169[6]` to `0x1` to enable the counter to detect error bits.
4. Set `0x168[0]` to `0x1` to enable the serial bit checker for ODI.
5. If the DFE is enabled[24]:
   a. Set `0x169[2]` to `0x1` to enable DFE speculation.
   b. Set `0x149[5:0]` to `0x07` to read the DFE tap signs.
   c. Read `0x17F[6]`[25], and store it as `DFE_tap1_sign`.
6. If the DFE is disabled[24]:
   a. Set `0x169[2]` to `0x0` to disable DFE speculation.
7. Trade off between the ODI runtime and BER resolution by setting the number of bits to count before stopping at each horizontal or vertical point combination. Set `{0x169[1:0], 0x168[5]}` to:
   a. Count up to 2^{16}: `0x0`.
   b. Count up to 10^6: `0x1`.
   c. Count up to 10^7: `0x2`.
   d. Count up to 10^8: `0x3`.
   e. Count up to 3×10^8: `0x4`.
   f. Count up to 10^9: `0x5`.
   g. Count up to 2^{32}: `0x6`.
8. Set `0x158[5]` to `0x1` to enable serial bit checker control.
9. Set `0x12D[4]` to `0x0` to disable the path from the DFE to the Avalon memory-mapped interface testmux.
10. If the device is H-tile production, configure the ODI bandwidth for the desired data rate by setting register `{0x145[7], 0x144[7]}` to the corresponding value from the following table .

Table 94. ODI Bandwidth Data Rate Settings for H-Tile Production

Data Rate	Register Setting
> 25 Gbps	`0x0`
25 Gbps ≥ data rate > 16 Gbps	`0x2`
16 Gbps ≥ data rate > 10 Gbps	`0x1`
Data rate ≤ 10 Gbps	`0x3`

(23) To determine RX adaptation mode, read `0x161[5]`. RX adaptation is in **manual** mode when `0x161[5] = 1`.

(24) To determine DFE mode, read `0x161[6]`. DFE is **disabled** when `0x161[6] = 1`.

(25) Wait 25 µs between setting register 0x149[5:0] and reading 0x17E or 0x17F.

11. If the device is not H-tile production, configure the ODI bandwidth for the desired data rate by setting register `{0x145[7], 0x144[7]}` to the corresponding value from the following table .

Table 95. ODI Bandwidth Data Rate Settings for Non-H-Tile Production Tiles

Data Rate	Register Setting
> 20 Gbps	`0x0`
20 Gbps ≥ data rate > 12.5 Gbps	`0x2`
12.5 Gbps ≥ data rate > 6.5 Gbps	`0x1`
Data rate ≤ 6.5 Gbps	`0x3`

12. Set `0x144[6:4]` to `0x0` to set the ODI phase interpolator to 128.
13. Set `0x140[5:3]` to `0x0` to disable the ODI test pattern generator.
14. Set `0x13C[0]` to `0x0`, then set it to `0x1` to reset then release the reset on the serial bit checker control logic.
15. Set `0x171[4:1]` to `0xB`, to configure the Avalon memory-mapped interface testmux to read the ODI counter values.

To save time, you can sweep the horizontal eye opening across 128 phase steps with the vertical phase set to zero. This helps determine the extent of the eye opening. Then, you can perform a refined horizontal or vertical eye sweep to capture the two-dimensional eye diagram.

2.4.4.4.3. Scanning the Horizontal Eye Opening

To sweep the horizontal eye opening, perform these procedures:

1. Set `0x144[2]` to `0x0` and `0x156[0]` to `0x0` to capture the zero crossing for the bottom half of the eye.
2. Set `0x143[7:2]` to `0x00` to set the vertical step to 0.
3. If DFE is disabled[26], set `0x14D[0]` to 0.
4. If the DFE is enabled[26], set `0x14D[0]` to `0x1` to select positive speculation.
    a. If `DFE_tap1_sign` = 0, set `0x156[1]` to `0x1`.
    b. If `DFE_tap1_sign` = 1, set `0x156[1]` to `0x0`.
5. If the targeted device is H-tile production or H-tile ES3:
    a. Set `0x100[4]` to 1.
    b. Set `0x000[7:0]` to `0x01` to request PreSICE to calibrate the ODI samplers.
    c. Read `0x481[2]` until it becomes 0.
6. Set `0x171[4:1]` to `0xB` to configure the Avalon memory-mapped interface testmux.
7. Define a floating point array with 128 members called `ODI_error_count`, and set all the values to 0.

[26] To determine DFE mode, read `0x161[6]`. DFE is **disabled** when `0x161[6] = 1`.

8. The L-tiles and H-tiles have two eyes: an odd eye and an even eye. You must capture both eyes. To start, set `0x157[3:2]` to `0x2` to capture the odd eye.
9. Create an integer called `horizontal_phase` and set it to 1. Repeat Step 10 through Step 23 while incrementing `horizontal_phase` until it becomes 128.
10. Set `0x145[6:0]` to the encoded phase in Table 96 on page 163, for example, `0x71` for `horizontal_phase` = 1.
11. Set `0x168[2]` to `0x0` to reset the serial bit counter.
12. Set `0x168[2]` to `0x1` to release the reset on the serial bit counter.
13. Set `0x149[5:0]` to `0x1C` to be able to read the ODI status.
14. Read `0x17E[1]`[27] until it becomes `0x1` to indicate that the ODI has received the selected number of bits and has completed.
15. Set `0x149[5:0]` to `0x1B` to read out the number of ODI error bits.
16. Read `0x17E[7:0]`[27], and save this value as an integer `ODI_count_A`.
17. Set `0x149[5:0]` to `0x1A`.
18. Read `0x17E[7:0]`[27], and save this value as an integer `ODI_count_B`.
19. Set `0x149[5:0]` to `0x19`.
20. Read `0x17E[7:0]`[27], and save this value as an integer `ODI_count_C`.
21. Set `0x149[5:0]` to `0x18`.
22. Read `0x17E[7:0]`[27], and save this value as an integer `ODI_count_D`.
23. `ODI_error_count[horizontal_phase]` = `ODI_count_A` * 2^{24} + `ODI_count_B` * 2^{16} + `ODI_count_C` * 2^{8} + `ODI_count_D` + `ODI_error_count[horizontal_phase]`
    If the device is not H-tile production, `ODI_error_count` may be greater than the actual count by 1.
24. Now, set `0x157[3:2]` to `0x1` to capture the even eye, and repeat Step 9.
25. Scan through the `ODI_error_count` array, find the phases that have no errors, and determine the left eye opening and right eye opening. Store the phases as `left_phase` and `right_phase`.

2.4.4.4.4. Scanning the Horizontal and Vertical Phases

Sweep the horizontal and vertical phases to get a two-dimensional eye diagram.

Note: The horizontal phase steps (`left_phase` and `right_phase`) correspond to zero BER in the previous sweep. The phase steps with no BER may wrap around, for example, from phase 110 to phase 20.

[27] Wait 25 µs between setting register 0x149[5:0] and reading 0x17E or 0x17F.

1. Create a 130*130 2D floating point array called `ODI_error_count` and initialize it to 0.
2. Create a 130*130 2D floating point array called `ODI_pattern_count` and initialize it to 0.
3. If DFE is disabled[28], repeat Step 5 to Step 37 two times:
    a. In the first iteration, set `0x144[2]` to `0x0` and `0x14D[0]` to `0x0`.
    b. In the second iteration, set `0x144[2]` to `0x1` and `0x14D[0]` to `0x1`.
4. If the DFE is enabled[28], repeat Step 5 to Step 37 four times:
    a. In the first iteration, set `0x144[2]` to `0x0` and `0x14D[0]` to `0x0`.
        i. If `DFE_tap1_sign` = 0, set `0x156[2]` to `0x0`.
        ii. If `DFE_tap1_sign` = 1, set `0x156[2]` to `0x1`.
    b. In the second iteration, set `0x144[2]` to `0x0` and `0x14D[0]` to `0x1`.
        i. If `DFE_tap1_sign` = 0, set `0x156[2]` to `0x1`.
        ii. If `DFE_tap1_sign` = 1, set `0x156[2]` to `0x0`.
    c. In the third iteration, set `0x144[2]` to `0x1` and `0x14D[0]` to `0x0`.
        i. If `DFE_tap1_sign` = 0, set `0x156[2]` to `0x0`.
        ii. If `DFE_tap1_sign` = 1, set `0x156[2]` to `0x1`.
    d. In the fourth iteration, set `0x144[2]` to `0x1` and `0x14D[0]` to `0x1`.
        i. If `DFE_tap1_sign` = 0, set `0x156[2]` to `0x1`.
        ii. If `DFE_tap1_sign` = 1, set `0x156[2]` to `0x0`.
5. If the targeted device is H-tile production or H-tile ES3:
    a. Set `0x100[4]` to `0x1`.
    b. Set `0x0[7:0]` to `0x01` to request PreSICE to calibrate the ODI samplers.
    c. Read `0x481[2]` until it becomes `0x0`.
6. Set `0x171[4:1]` to `0xB` to configure the Avalon memory-mapped interface testmux.
7. Set `0x157[3:2]` to `0x2` to capture the odd eye.
8. Set integer variable `vertical_phase` to 0 and repeat Step 9 to Step 11 while incrementing `vertical_phase` until it reaches 126.
9. If the vertical phase < `0x3F`:
    a. Set `0x156[0]` to `0x1` to capture the top half of the eye.
    b. Set `0x143[7:2]` to `0x3F` – `vertical_phase`.
10. If the vertical phase >= `0x3F`:
    a. Set `0x156[0]` to `0x0` to capture the bottom half of the eye.
    b. Set `0x143[7:2]` to `vertical_phase` – `0x3F`.

[28] To determine DFE mode, read `0x161[6]`. DFE is **disabled** when `0x161[6] = 1`.

11. Set an integer horizontal_phase to left_phase – 10 and repeat Step 12 to Step 36 while incrementing horizontal_phase until it reaches right_phase + 10.
    a. If right_phase < left_phase, in other words, the eye is wrapped around, increment right_phase by 0x80.
12. If horizontal_phase is < 1, increment it by 0x80.
13. If horizontal_phase > 128, decrement it by 0x80.
14. Set 0x145[6:0] to the encoded horizontal_phase.
15. Set 0x168[2] to 0x0 to reset the serial bit counter.
16. Set 0x168[2] to 0x1 to release the reset on the serial bit counter.
17. Set 0x149[5:0] to 0x1C to read the ODI status.
18. Read 0x17E[1][29] until it becomes 0x1 to indicate that the ODI has received the selected number of bits and has completed.
19. Set 0x149[5:0] to 0x1B to read the number of ODI error bits.
20. Read 0x17E[7:0][29], and save it as an integer ODI_count_A.
21. Set 0x149[5:0] to 0x1A to read the number of ODI error bits.
22. Read 0x17E[7:0][29], and save it as an integer ODI_count_B.
23. Set 0x149[5:0] to 0x19 to read the number of ODI error bits.
24. Read 0x17E[7:0][29], and save it as an integer ODI_count_C.
25. Set 0x149[5:0] to 0x18 to read the number of ODI error bits.
26. Read 0x17E[7:0][29], and save it as an integer ODI_count_D.
27. ODI_error_count[horizontal_phase][vertical_phase] = ODI_count_A * 2^{24} + ODI_count_B * 2^{16} + ODI_count_C * 2^{8} + ODI_count_D + ODI_error_count[horizontal_phase][vertical_phase]
    If the device is not H-tile production, the ODI_error_count may be greater than the actual count by 1.
28. Set 0x149[5:0] to 0x17 to read the number of ODI error bits.
29. Read 0x17E[7:0][29], and save it as an integer ODI_pattern_A.
30. Set 0x149[5:0] to 0x16 to read the number of ODI error bits.
31. Read 0x17E[7:0][29], and save it as an integer ODI_pattern_B.
32. Set 0x149[5:0] to 0x15 to read the number of ODI error bits.
33. Read 0x17E[7:0][29], and save it as an integer ODI_pattern_C.
34. Set 0x149[5:0] to 0x14 to read the number of ODI error bits.
35. Read 0x17E[7:0][29], and save it as an integer ODI_pattern_D.

(29) Wait 25 µs between setting register 0x149[5:0] and reading 0x17E or 0x17F.

36. `ODI_pattern_count[horizontal_phase][vertical_phase]` = `ODI_pattern_A` * 2^{24} + `ODI_pattern_B` * 2^{16} + `ODI_pattern_C` * 2^{8} + `ODI_pattern_D` + `ODI_pattern_count[horizontal_phase][vertical_phase]`
37. Set `0x157[3:2]` to `0x1` to capture the even eye, and repeat Step 8.
38. The BER at the horizontal phase and vertical phase = `ODI_error_count[horizontal_phase][vertical_phase]/ODI_Pattern_count[horizontal_phase][vertical_phase]`.

2.4.4.4.5. How to Disable ODI

After ODI is complete, disable ODI with the following procedure.

1. Set register `0x168[0]` to `0x0`.
2. Set register `0x158[5]` to `0x0` to disable ODI control from the Avalon memory-mapped interface.
3. If the RX adaptation mode is manual[30], set `0x148[0]` to `0x0` to disable the adaptation logic and save power.
4. If the device is H-tile production and you want to enable background calibration, set `0x542[0]` to `0x1`.

2.4.4.4.6. Horizontal Phase Step Mapping

Table 96. BER Resolution Settings

This table shows the mapping between the horizontal phase step and register `0x145[6:0]`.

Step	0x145[6:0]	Step	0x145[6:0]	Step	0x145[6:0]	Step	0x145[6:0]
1	7'b1110001	33	7'b1000000	65	7'b0010001	97	7'b0100000
2	7'b1110000	34	7'b1000001	66	7'b0010000	98	7'b0100001
3	7'b1110011	35	7'b1000010	67	7'b0010011	99	7'b0100010
4	7'b1110010	36	7'b1000011	68	7'b0010010	100	7'b0100011
5	7'b1110111	37	7'b1000110	69	7'b0010111	101	7'b0100110
6	7'b1110110	38	7'b1000111	70	7'b0010110	102	7'b0100111
7	7'b1110101	39	7'b1000100	71	7'b0010101	103	7'b0100100
8	7'b1110100	40	7'b1000101	72	7'b0010100	104	7'b0100101
9	7'b1111101	41	7'b1001100	73	7'b0011101	105	7'b0101100
10	7'b1111100	42	7'b1001101	74	7'b0011100	106	7'b0101101
11	7'b1111111	43	7'b1001110	75	7'b0011111	107	7'b0101110
12	7'b1111110	44	7'b1001111	76	7'b0011110	108	7'b0101111
13	7'b1111011	45	7'b1001010	77	7'b0011011	109	7'b0101010
14	7'b1111010	46	7'b1001011	78	7'b0011010	110	7'b0101011

continued...

[30] To determine RX adaptation mode, read `0x161[5]`. RX adaptation is in **manual** mode when `0x161[5] = 1`.

Step	0x145[6:0]	Step	0x145[6:0]	Step	0x145[6:0]	Step	0x145[6:0]
15	7'b1111001	47	7'b1001000	79	7'b0011001	111	7'b0101000
16	7'b1111000	48	7'b1001001	80	7'b0011000	112	7'b0101001
17	7'b1101001	49	7'b1011000	81	7'b0001001	113	7'b0111000
18	7'b1101000	50	7'b1011001	82	7'b0001000	114	7'b0111001
19	7'b1101011	51	7'b1011010	83	7'b0001011	115	7'b0111010
20	7'b1101010	52	7'b1011011	84	7'b0001010	116	7'b0111011
21	7'b1101111	53	7'b1011110	85	7'b0001111	117	7'b0111110
22	7'b1101110	54	7'b1011111	86	7'b0001110	118	7'b0111111
23	7'b1101101	55	7'b1011100	87	7'b0001101	119	7'b0111100
24	7'b1101100	56	7'b1011101	88	7'b0001100	120	7'b0111101
25	7'b1100101	57	7'b1010100	89	7'b0000101	121	7'b0110100
26	7'b1100100	58	7'b1010101	90	7'b0000100	122	7'b0110101
27	7'b1100111	59	7'b1010110	91	7'b0000111	123	7'b0110110
28	7'b1100110	60	7'b1010111	92	7'b0000110	124	7'b0110111
29	7'b1100011	61	7'b1010010	93	7'b0000011	125	7'b0110010
30	7'b1100010	62	7'b1010011	94	7'b0000010	126	7'b0110011
31	7'b1100001	63	7'b1010000	95	7'b0000001	127	7'b0110000
32	7'b1100000	64	7'b1010001	96	7'b0000000	128	7'b0110001

2.5. Implementing the PHY Layer for Transceiver Protocols

2.5.1. PCI Express (PIPE)

You can use Intel Stratix 10 transceivers to implement a complete PCI Express solution for Gen1, Gen2, and Gen3, at datarates of 2.5, 5.0, and 8 Gbps, respectively.

To implement PCI Express, you must select the external oscillator as the data path configuration clock. This allows you to set the frequency accurately through `OSC_CLK_1`. You must provide a free running and stable clock to the `OSC_CLK_1` pin for transceiver calibration. Refer to *Calibration* for more details.

Configure the transceivers for PCIe functionality using one of the following methods:

- **Intel Stratix 10 Hard IP for PCIe**

  This is a complete PCIe solution that includes the Transaction, Data Link, and PHY/MAC layers. The Hard IP solution contains dedicated hard logic, which connects to the transceiver PHY interface.

- **PIPE Gen1/Gen2/Gen3 Transceiver Configuration Rules for the Native PHY IP Core**

  Use the Native PHY IP core to configure the transceivers in PCIe mode, giving access to the PIPE interface (commonly called PIPE mode in transceivers). This mode enables you to connect the transceiver to a third-party MAC to create a complete PCIe solution.

  The PIPE specification (version 3.0) provides implementation details for a PCIe-compliant physical layer. The Native PHY IP Core for PIPE Gen1, Gen2, and Gen3 supports x1, x2, x4, x8 or x16 operation for a total aggregate bandwidth ranging from 2.5 to 128 Gbps. The x1 configuration uses the x1 clock network and the channel is non-bonded. The x2, x4, x8 and x16 configurations support channel bonding for two-lane, four-lane, eight-lane, and sixteen-lane links. In these bonded channel configurations, the PCS and PMA blocks of all bonded channels share common clock and reset signals.

Gen1 and Gen2 modes use 8B/10B encoding, which has a 20% overhead to overall link bandwidth. Gen3 modes use 128b/130b encoding, which has an overhead of less than 2%. Gen1 and Gen2 modes use the Standard PCS, and Gen3 mode uses the Gen3 PCS for its operation.

Table 97. Intel Stratix 10 PCIe Hard IP and PIPE Support Configuration

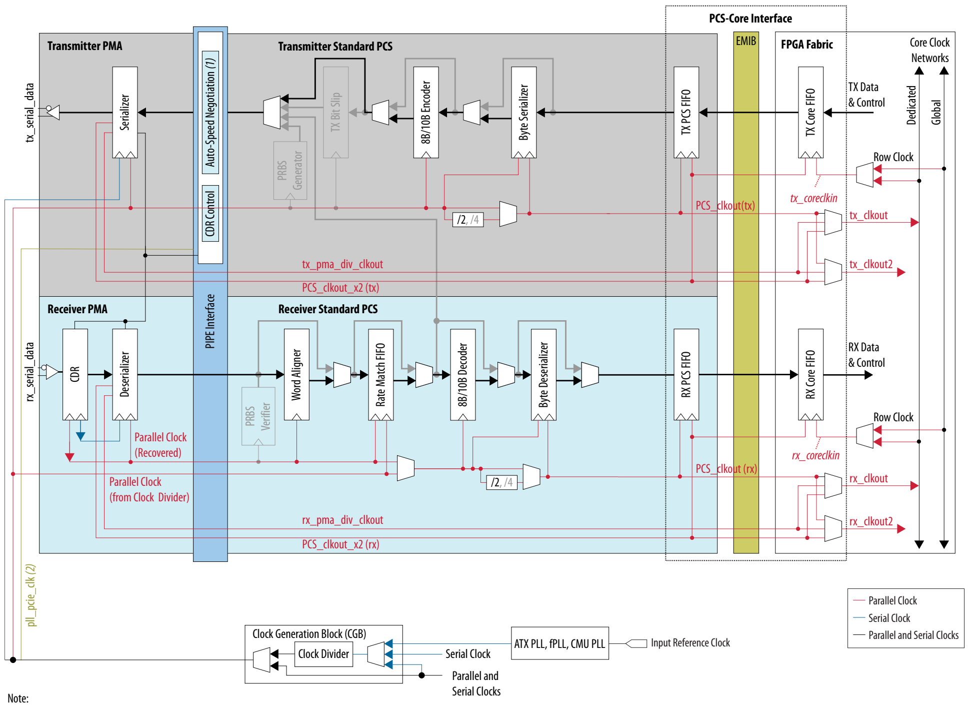
Support	Intel Stratix 10 Hard IP for PCI Express	L-Tile/H-Tile Native PHY IP Core for PCI Express (PIPE)
Gen1, Gen2, and Gen3 datarates	Yes	Yes
MAC, data link, and transaction layer	Yes	User implementation in FPGA fabric
Transceiver interface	Hard IP through PIPE 3.0 based interface	• PIPE 2.0 for Gen1 and Gen2 • PIPE 3.0 based for Gen3 with Gen1/Gen2 support

Related Information

- Calibration on page 433
- Intel PHY Interface for the PCI Express (PIPE) Architecture PCI Express

2.5.1.1. Transceiver Channel Datapath for PIPE

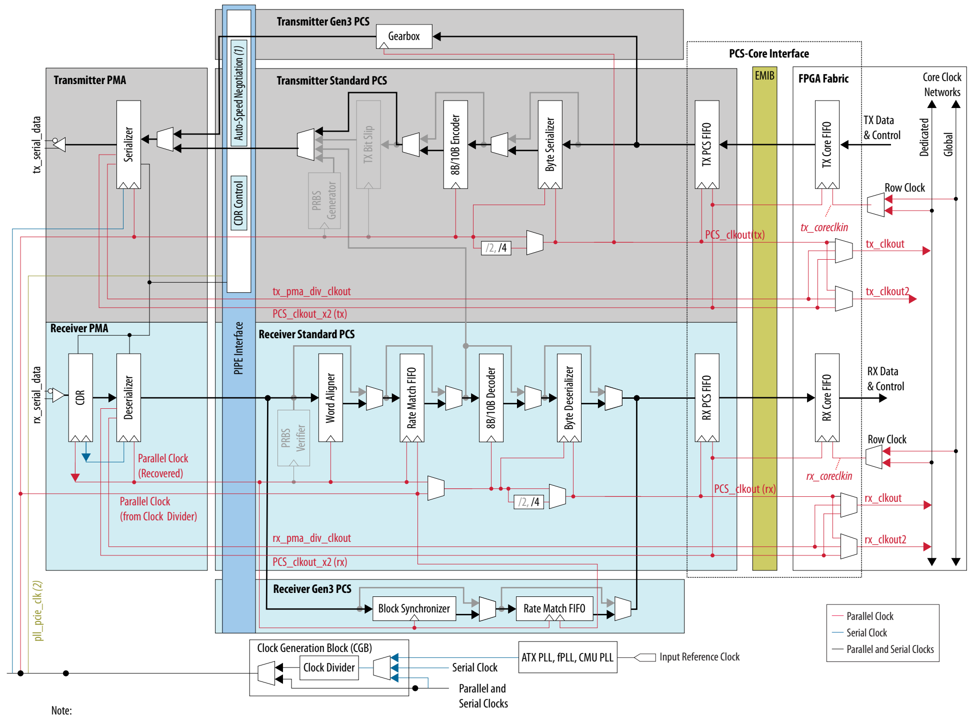
Figure 91. Transceiver Channel Datapath for PIPE Gen1/Gen2 Configurations



Note:
1. Auto-speed negotiation for Gen3 x1, x2, x4, x8, and x16.
2. hclk for auto-speed negotiation block.

Send Feedback

Figure 92. Transceiver Channel Datapath for PIPE Gen1/Gen2/Gen3 Configurations



Note:
1. Auto-speed negotiation for Gen3 x1, x2, x4, x8, and x16.
2. hclk for auto-speed negotiation block.

2.5.1.2. Supported PIPE Features

PIPE Gen1, Gen2, and Gen3 configurations support different features.

Table 98. Supported Features for PIPE Configurations

Protocol Feature	Gen1 (2.5 Gbps)	Gen2 (5 Gbps)	Gen3 (8 Gbps)
x1, x2, x4, x8, x16 link configurations	Yes	Yes	Yes
PCIe-compliant synchronization state machine	Yes	Yes	Yes
±300 ppm (total 600 ppm) clock rate compensation	Yes	Yes	Yes
Transmitter driver electrical idle	Yes	Yes	Yes
Receiver detection	Yes	Yes	Yes
8B/10B encoding/decoding disparity control	Yes	Yes	No
128b/130b encoding/decoding	No	No	Yes (supported through the Gearbox)
Scrambling/Descrambling	No	No	Yes (implemented in FPGA fabric)
Power state management	Yes	Yes	Yes
Receiver PIPE status encoding `pipe_rxstatus[2:0]`	Yes	Yes	Yes

continued...

Protocol Feature	Gen1 (2.5 Gbps)	Gen2 (5 Gbps)	Gen3 (8 Gbps)
Dynamic switching between 2.5 Gbps and 5 Gbps signaling rate	No	Yes	No
Dynamic switching between 2.5 Gbps, 5 Gbps, and 8 Gbps signaling rate	No	No	Yes
Dynamic transmitter margining for differential output voltage control	No	Yes	Yes
Dynamic transmitter buffer de-emphasis of –3.5 dB and –6 dB	No	Yes	Yes
Dynamic Gen3 transceiver pre-emphasis, de-emphasis, and equalization	No	No	Yes
PCS PMA interface width (bits)	10	10	32
Receiver Electrical Idle Inference (EII)	Implement in FPGA fabric	Implement in FPGA fabric	Implement in FPGA fabric

Related Information

- Intel Stratix 10 PCI Express Gen3 PCS Architecture on page 386
  For more information about PIPE Gen3
- Intel PHY Interface for the PCI Express (PIPE) Architecture PCI Express 2.0 (Requires registration to access this site)
- Intel PHY Interface for the PCI Express (PIPE) Architecture PCI Express 3.0 (Requires registration to access this site)

2.5.1.2.1. Gen1/Gen2 Features

In a PIPE configuration, each channel has a PIPE interface block that transfers data, control, and status signals between the PHY-MAC layer and the transceiver channel PCS and PMA blocks. The PIPE configuration is based on the PIPE 2.0 specification. If you use a PIPE configuration, you must implement the PHY-MAC layer using soft IP in the FPGA fabric.

Dynamic Switching Between Gen1 (2.5 Gbps) and Gen2 (5 Gbps)

In a PIPE configuration, Native PHY IP core provides an input signal `pipe_rate[1:0]` that is functionally equivalent to the RATE signal specified in the PCIe specification. A change in value from 2'b00 to 2'b01 on this input signal `pipe_rate[1:0]` initiates a datarate switch from Gen1 to Gen2. A change in value from 2'b01 to 2'b00 on the input signal initiates a datarate switch from Gen2 to Gen1.

Transmitter Electrical Idle Generation

The PIPE interface block puts the transmitter buffer in an electrical idle state when the electrical idle input signal is asserted. During electrical idle, the transmitter buffer differential and common mode output voltage levels are compliant with the PCIe Base Specification 2.0 for both PCIe Gen1 and Gen2 datarates.

The PCIe specification requires the transmitter driver to be in electrical idle in certain power states. For more information about input signal levels required in different power states, refer to *Power State Management*.

Related Information

Power State Management on page 169

Power State Management

To minimize power consumption, the physical layer device must support the following power states.

Table 99. Power States Defined in the PCIe Specification

Power States	Description
P0	Normal operating state during which packet data is transferred on the PCIe link.
P0s, P1, and P2	The PHY-MAC layer directs the physical layer to transition into these low-power states.

The PIPE interface provides a `pipe_powerdown` input port for each transceiver channel configured in a PIPE configuration.

The PCIe specification requires the physical layer device to implement power-saving measures when the P0 power state transitions to the low power states. Intel Stratix 10 transceivers do not implement these power-saving measures except for putting the transmitter buffer in electrical idle mode in the lower power states.

8B/10B Encoder Usage for Compliance Pattern Transmission Support

The PCIe transmitter transmits a compliance pattern when the Link Training and Status State Machine (LTSSM) enters the Polling.Compliance substate. The Polling.Compliance substate assesses if the transmitter is electrically compliant with the PCIe voltage and timing specifications.

Receiver Status

The PCIe specification requires the PHY to encode the receiver status on a 3-bit status signal `pipe_rx_status[2:0]`. This status signal is used by the PHY-MAC layer for its operation. The PIPE interface block receives status signals from the transceiver channel PCS and PMA blocks, and encodes the status on the `pipe_rx_status[2:0]` signal to the FPGA fabric. The encoding of the status signals on the `pipe_rx_status[2:0]` signal conforms to the PCIe specification.

Receiver Detection

The PIPE interface block provides an input signal `pipe_tx_detectrx_loopback` for the receiver detect operation. The PCIe protocol requires this signal to be high during the Detect state of the LTSSM. When the `pipe_tx_detectrx_loopback` signal is asserted in the P1 power state, the PIPE interface block sends a command signal to the transmitter driver in that channel to initiate a receiver detect sequence. In the P1 power state, the transmitter buffer must always be in the electrical idle state. After receiving this command signal, the receiver detect circuitry creates a step voltage at the output of the transmitter buffer. The time constant of the step voltage on the trace increases if an active receiver that complies with the PCIe input impedance requirements is present at the far end. The receiver detect circuitry monitors this time constant to determine if a receiver is present.

Note: For the receiver detect circuitry to function reliably, the transceiver on-chip termination must be used. Also, the AC-coupling capacitor on the serial link and the receiver termination values used in your system must be compliant with the PCIe Base Specification 2.0.

The PIPE core provides a 1-bit PHY status signal `pipe_phy_status` and a 3-bit receiver status signal `pipe_rx_status[2:0]` to indicate whether a receiver is detected, as per the PIPE 2.0 specifications.

Gen1 and Gen2 Clock Compensation

In compliance with the PIPE specification, Intel Stratix 10 receiver channels have a rate match FIFO to compensate for small clock frequency differences up to ±300 ppm between the upstream transmitter and the local receiver clocks.

Consider the following guidelines for PIPE clock compensation:

- Insert or delete one skip (SKP) symbol in an SKP ordered set.

  Note: The SKP symbol is also represented as K28.0, and is used for compensating for different bit rates between two communicating ports.

- A minimum limit is imposed on the number of SKP symbols in SKP ordered set after deletion. A transmitted SKP ordered set is comprised of a single COM (K28.5) symbol followed by three SKP symbols. An ordered set may have an empty COM case after deletion.
- A maximum limit is imposed on the number of the SKP symbols in the SKP ordered set after insertion. An ordered set may have more than five symbols after insertion.
- For INSERT/DELETE cases: The flag status appears on the COM symbol of the SKP ordered set where insertion or deletion occurs.
- For FULL/EMPTY cases: The flag status appears where the character is inserted or deleted.

  Note: The PIPE interface translates the value of the flag to the appropriate `pipe_rx_status` signal.

- The PIPE mode also has a "0 ppm" configuration option that you can use in synchronous systems. The Rate Match FIFO Block is not expected to do any clock compensation in this configuration, but latency will be minimized.

Figure 93. Rate Match Deletion

This figure shows an example of rate match deletion in the case where two /K28.0/ SKP symbols must be deleted from the received data at the input of the rate match FIFO (rmfifo_input_data). Only one /K28.0/ SKP symbol is deleted per SKP ordered set received.

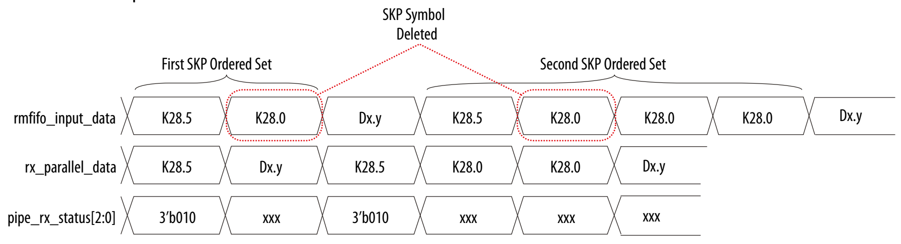


Figure 94. Rate Match Insertion

The figure below shows an example of rate match insertion in the case where two SKP symbols must be inserted. Only one /K28.0/ SKP symbol is inserted per SKP ordered set received.

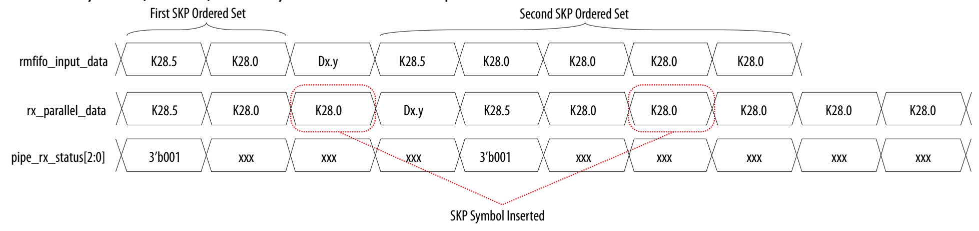


Figure 95. Rate Match FIFO Full

The rate match FIFO in PIPE mode automatically deletes the data byte that causes the FIFO to go full and drives `pipe_rx_status[2:0]` = 3'b101 synchronous to the subsequent data byte. The figure below shows the rate match FIFO full condition in PIPE mode. The rate match FIFO becomes full after receiving data byte D4.

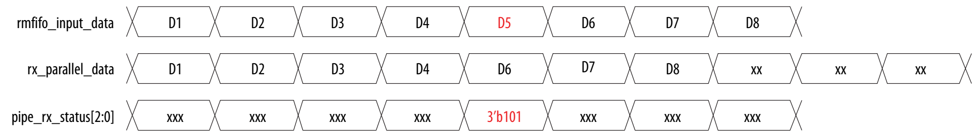
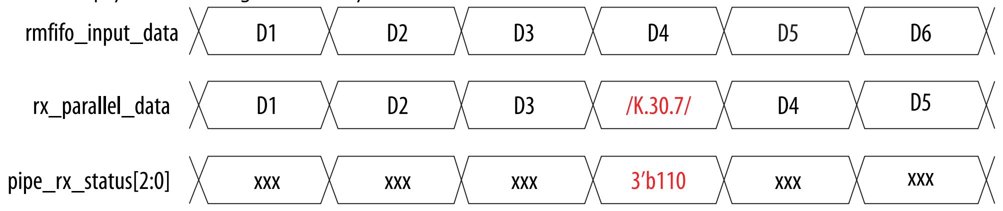


Figure 96. Rate Match FIFO Empty

The rate match FIFO automatically inserts the EDB symbol, /K30.7/ (9'h1FE), after the data byte that causes the FIFO to become empty and drives `pipe_rx_status[2:0]` = 3'b110 synchronous to the inserted /K30.7/ (9'h1FE). The figure below shows rate match FIFO empty condition in PIPE mode. The rate match FIFO becomes empty after reading out data byte D3.



PIPE 0 ppm

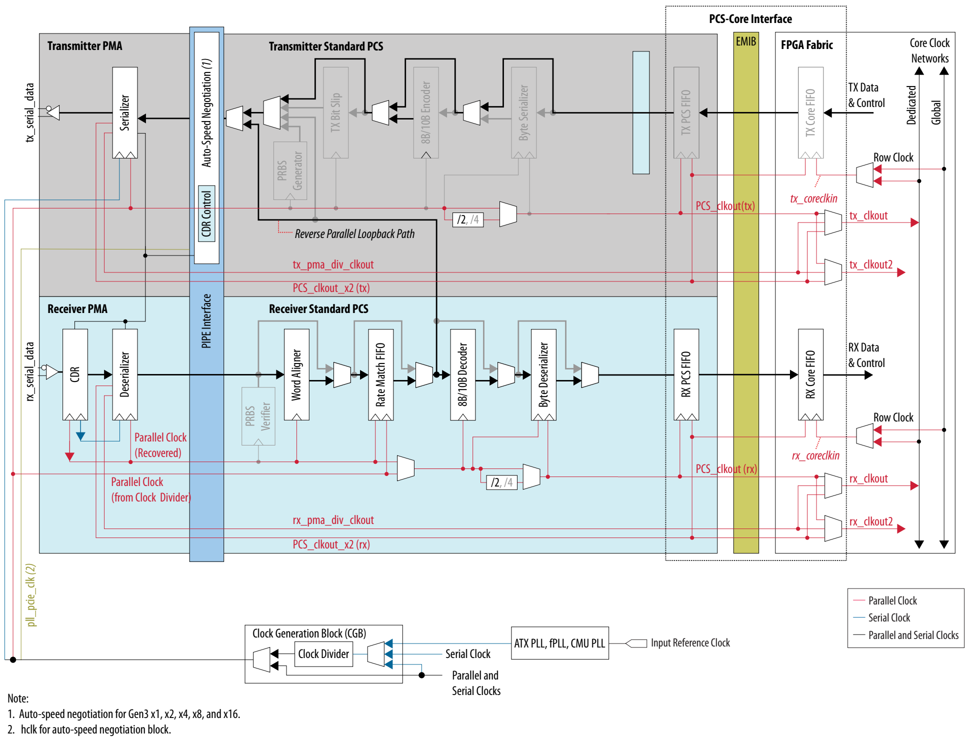
The PIPE mode also has a "0 ppm" configuration option that can be used in synchronous systems. The rate match FIFO is not expected to do any clock compensation in this configuration, but latency will be minimized.

PCIe Reverse Parallel Loopback

PCIe reverse parallel loopback is only available for PCIe Gen1, Gen2, and Gen3 datarates. The received serial data passes through the receiver CDR, deserializer, word aligner, and rate match FIFO. The data is then looped back to the transmitter serializer and transmitted out through the transmitter buffer. The received data is also available to the FPGA fabric through the `rx_parallel_data` port. This loopback mode is based on PCIe specification 2.0. Intel Stratix 10 devices provide an input signal `pipe_tx_detectrx_loopback` to enable this loopback mode.

Note: This is the only loopback option supported in PIPE configurations.

Figure 97. PCIe Reverse Parallel Loopback Mode Datapath



Related Information

- Intel Stratix 10 Standard PCS Architecture on page 366
- Intel PHY Interface for the PCI Express (PIPE) Architecture PCI Express 2.0

2.5.1.2.2. Gen3 Features

The following subsections describes the Intel Stratix 10 transceiver block support for PIPE Gen3 features.

The PCS supports the PIPE 3.0 base specification. The 32-bit wide PIPE 3.0-based interface controls PHY functions such as transmission of electrical idle, receiver detection, and speed negotiation and control.

Auto-Speed Negotiation (ASN)

PIPE Gen3 mode enables ASN between Gen1 (2.5 Gbps), Gen2 (5.0 Gbps), and Gen3 (8.0 Gbps) signaling datarates. The signaling rate switch is accomplished through frequency scaling and configuration of the PMA and PCS blocks using a fixed 32-bit wide PIPE 3.0-based interface.

The PMA switches clocks between Gen1, Gen2, and Gen3 datarates. For a non bonded x1 channel, an ASN module facilitates speed negotiation in that channel. For bonded x2, x4, x8 and x16 channels, the ASN module selects the master channel to control the rate switch. The master channel distributes the speed change request to the other PMA and PCS channels.

The PCIe Gen3 speed negotiation process is initiated when Hard IP or the FPGA fabric requests a rate change. The ASN then places the PCS in reset, and dynamically shuts down the clock paths to disengage the current active state PCS (either Standard PCS or Gen3 PCS). If a switch to or from Gen3 is requested, the ASN automatically selects the correct PCS clock paths and datapath selection in the multiplexers. The ASN block then sends a request to the PMA block to switch the datarate, and waits for a rate change done signal for confirmation. When the PMA completes the rate change and sends confirmation to the ASN block, the ASN enables the clock paths to engage the new PCS block and releases the PCS reset. Assertion of the `pipe_phy_status` signal by the ASN block indicates the successful completion of this process.

Note: In Native PHY IP core - PIPE configuration, you must set `pipe_rate[1:0]`to initiate the transceiver datarate switch sequence.

Rate Switch

This section provides an overview of auto rate change between PIPE Gen1 (2.5 Gbps), Gen2 (5.0 Gbps), and Gen3 (8.0 Gbps) modes.

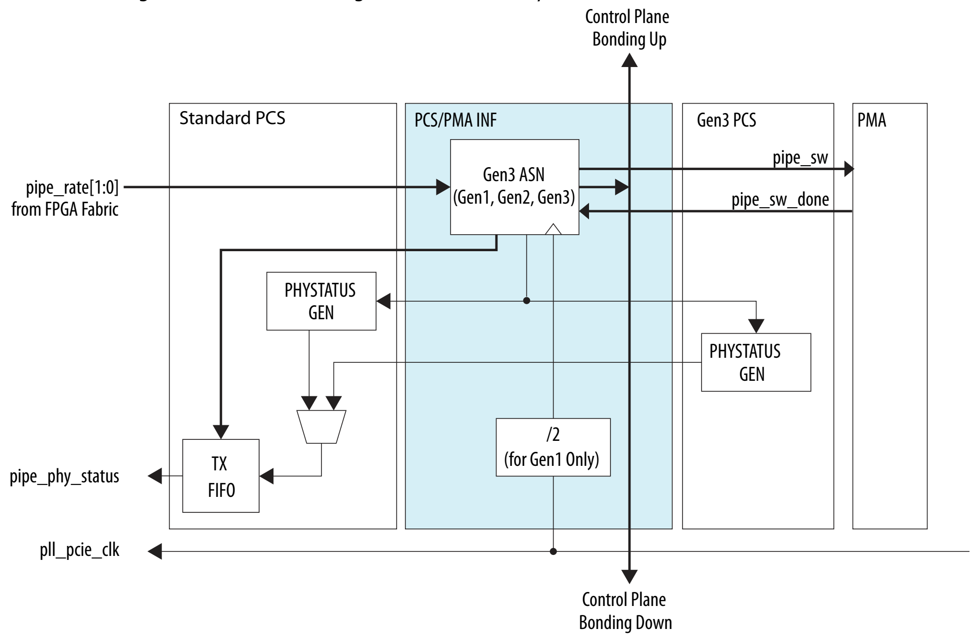
In Intel Stratix 10 devices, there is one ASN block common to the Standard PCS and Gen3 PCS, located in the PMA-PCS interface that handles all PIPE speed changes. The PIPE interface clock rate is adjusted to match the data throughput when a rate switch is requested.

Table 100. PIPE Gen3 32 bit PCS Clock Rates

PCIe Gen3 Capability Mode Enabled	Gen1	Gen2	Gen3
Lane datarate	2.5 Gbps	5 Gbps	8 Gbps
PCS clock frequency	250 MHz	500 MHz	250 MHz
FPGA fabric IP clock frequency	62.5 MHz	125 MHz	250 MHz
PIPE interface width	32-bit	32-bit	32-bit
`pipe_rate [1:0]`	2'b00	2'b01	2'b10

Figure 98. Rate Switch Change

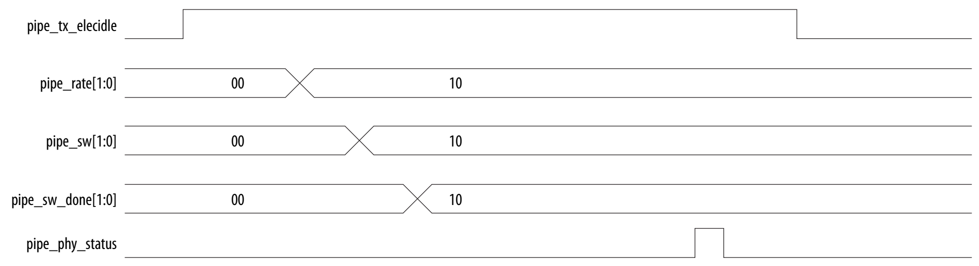
The block-level diagram below shows a high level connectivity between ASN and Standard PCS and Gen3 PCS.



The sequence of speed change between Gen1, Gen2, and Gen3 occurs as follows:

1. The PHY-MAC layer implemented in FPGA fabric requests a rate change through `pipe_rate[1:0]`.
2. The ASN block waits for the TX FIFO to flush out data. Then the ASN block asserts the PCS reset.
3. The ASN asserts the clock shutdown signal to the Standard PCS and Gen3 PCS to dynamically shut down the clock.
4. When the rate changes to or from the Gen3 speed, the ASN asserts the clock and data multiplexer selection signals.
5. The ASN uses a `pipe_sw[1:0]` output signal to send a rate change request to the PMA.
6. The ASN continuously monitors the `pipe_sw_done[1:0]` input signal from the PMA.
7. After the ASN receives the `pipe_sw_done[1:0]` signal, it deasserts the clock shut down signals to release the clock.
8. The ASN deasserts the PCS reset.
9. The ASN sends the speed change completion to the PHY-MAC interface. This is done through the `pipe_phy_status` signal to PHY-MAC interface.

Figure 99. Speed Change Sequence



Gen3 Transmitter Electrical Idle Generation

In the PIPE 3.0-based interface, you can place the transmitter in electrical idle during low power states. Before the transmitter enters electrical idle, you must send the Electrical Idle Ordered Set, consisting of 16 symbols with value 0x66. During electrical idle, the transmitter differential and common mode voltage levels are based on the *PCIe Base Specification 3.0*.

Gen3 Clock Compensation

Enable this mode from the Native PHY IP core when using the Gen3 PIPE transceiver configuration rule.

To accommodate PCIe protocol requirements and to compensate for clock frequency differences of up to ±300 ppm between source and termination equipment, receiver channels have a rate match FIFO. The rate match FIFO adds or deletes four SKP characters (32 bits) to keep the FIFO from becoming empty or full. If the rate match FIFO is almost full, the FIFO deletes four SKP characters. If the rate match FIFO is nearly empty, the FIFO inserts a SKP character at the start of the next available SKP ordered set. The `pipe_rx_status [2:0]` signal indicates FIFO full, empty, insertion and deletion.

Note: Refer to *Gen1 and Gen2 Clock Compensation* for waveforms.

Related Information

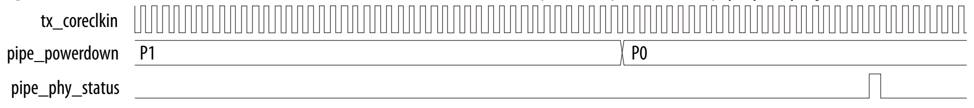
Gen1 and Gen2 Clock Compensation on page 170

Gen3 Power State Management

The PCIe base specification defines low power states for PHY layer devices to minimize power consumption. The Gen3 PCS does not implement these power saving measures, except when placing the transmitter driver in electrical idle in the low power state. In the P2 low power state, the transceivers do not disable the PIPE block clock.

Figure 100. P1 to P0 Transition

The figure below shows the transition from P1 to P0 with completion provided by `pipe_phy_status`.



CDR Control

The CDR control block performs the following functions:

- Controls the PMA CDR to obtain bit and symbol alignment
- Controls the PMA CDR to deskew within the allocated time
- Generates status signals for other PCS blocks

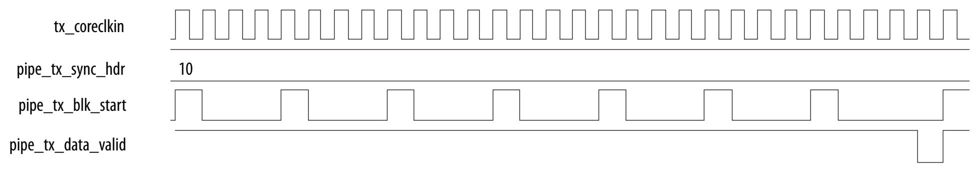
The PCIe base specification requires that the receiver L0s power state exit time be a maximum of 4 ms for Gen1, 2 ms for Gen2, and 4 ms for Gen3 signaling rates. The transceivers have an improved CDR control block to accommodate fast lock times. Fast lock times are necessary for the CDR to relock to the new multiplier/divider settings when entering or exiting Gen3 speeds.

Gearbox

As per the PIPE 3.0 specification, for every 128 bits that are moved across the Gen3 PCS, the PHY must transmit 130 bits of data. Intel uses the `pipe_tx_data_valid` signal every 16 blocks of data to transmit the built-up backlog of 32 bits of data.

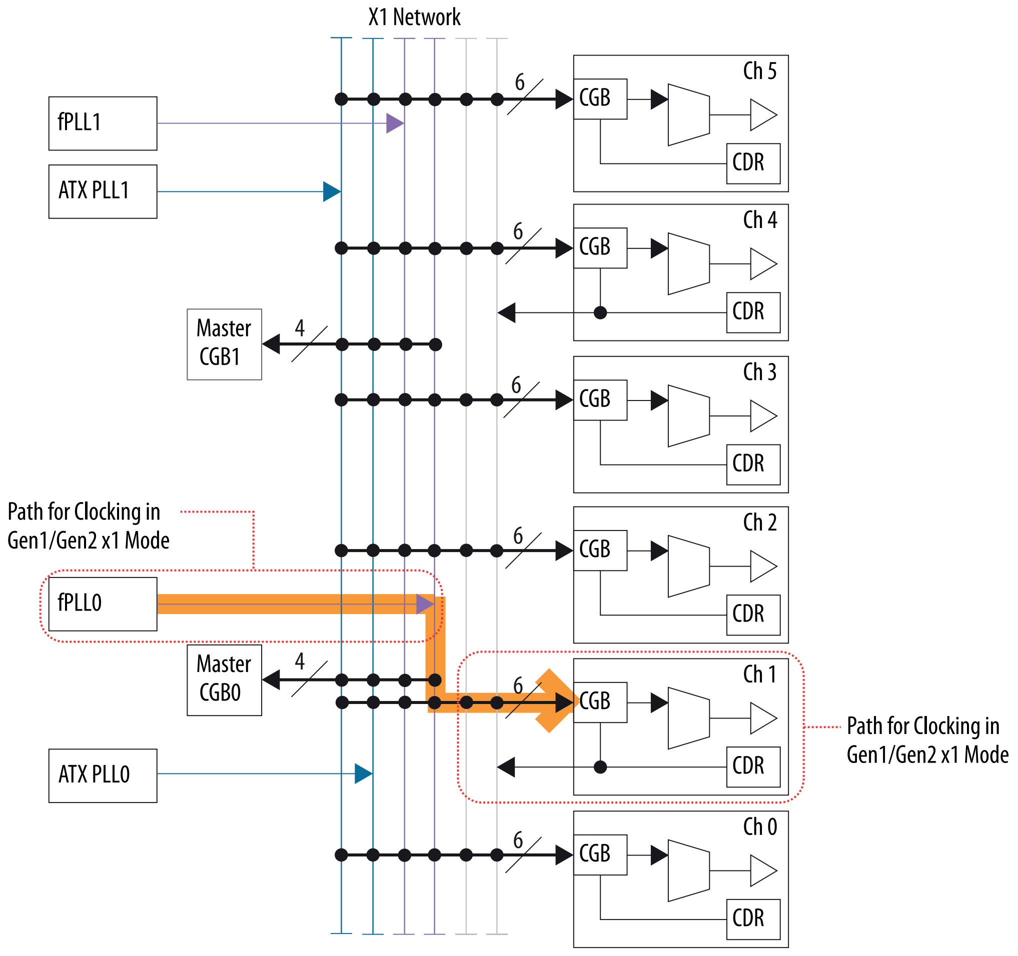
The 130-bit block is received as follows in the 32-bit data path: 34 (32+2-bit sync header), 32, 32, 32. During the first cycle, the gearbox converts the 34-bit input data to 32-bit data. During the next three clock cycles, the gearbox merges bits from adjacent cycles. For the gearbox to work correctly, a gap must be provided in the data for every 16 shifts because each shift contains two extra bits for converting the initial 34 bits to 32 bits in the gearbox. After 16 shifts, the gearbox has an extra 32 bits of data that are transmitted out. This requires a gap in the input data stream, which is achieved by driving `pipe_tx_data_valid` low for one cycle after every 16 blocks of data.

Figure 101. Gen3 Data Transmission



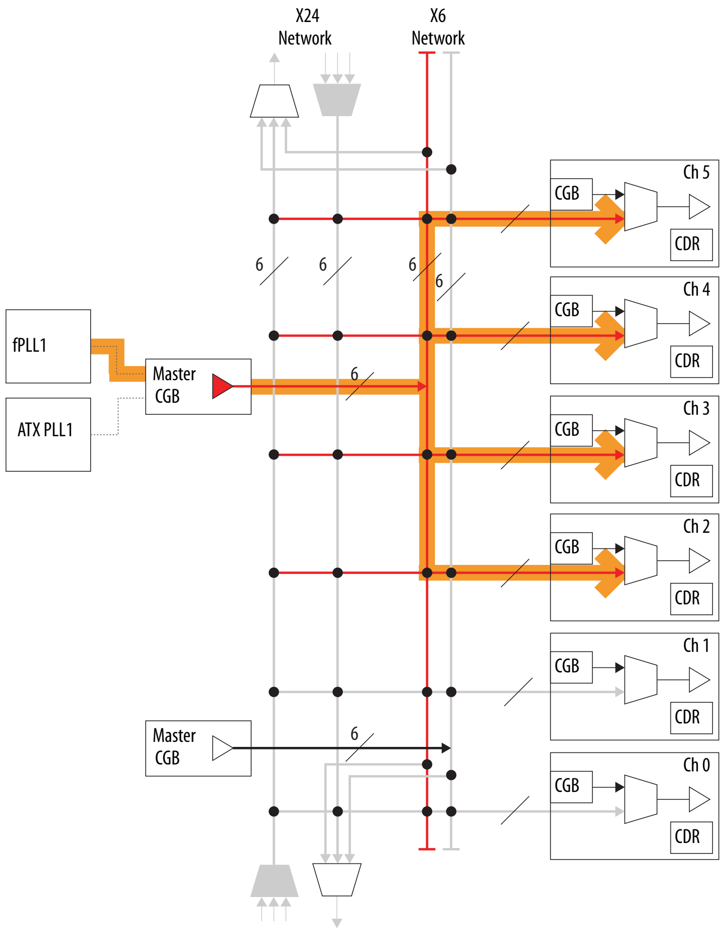
2.5.1.3. How to Connect TX PLLs for PIPE Gen1, Gen2, and Gen3 Modes

Figure 102. Use ATX PLL or fPLL for Gen1/Gen2 x1 Mode



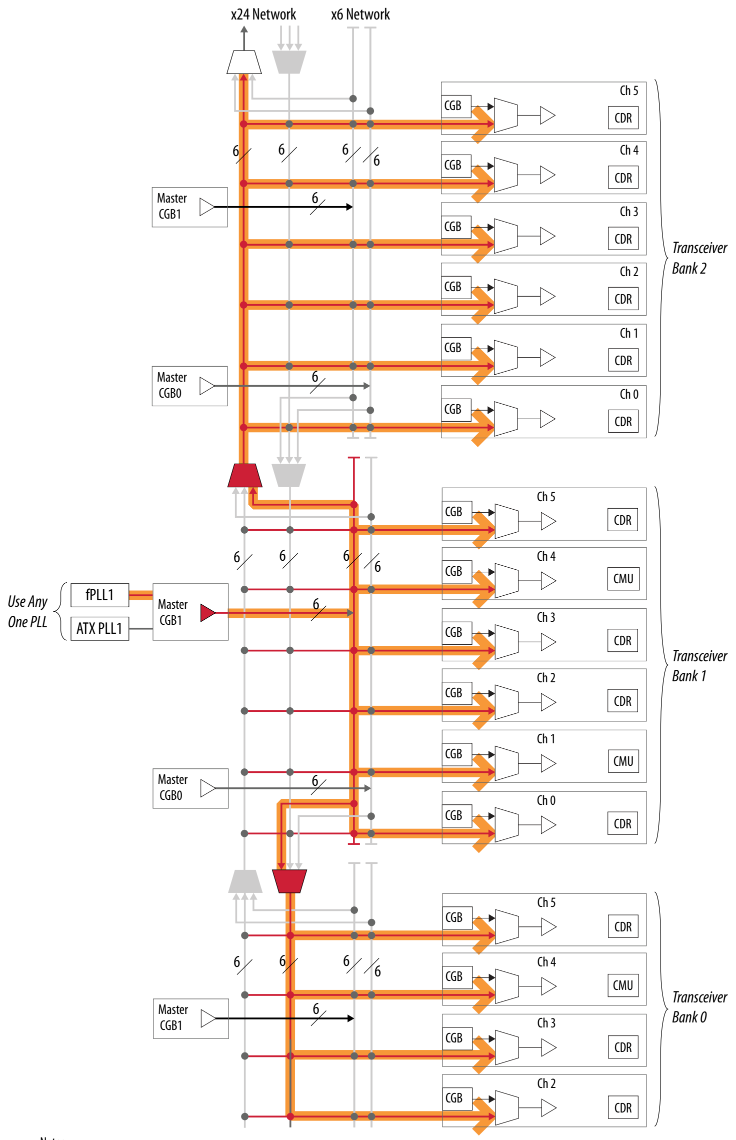
Notes:
1. The figure shown is just one possible combination for the PCIe Gen1/Gen2 x1 mode.
2. Gen1/Gen2 x1 mode uses the ATX PLL or fPLL.
3. Gen1/Gen2 x1 can use any channel from the given bank for which the ATX PLL or fPLL is enabled.
4. Use the pll_pcie_clk from either the ATX PLL or fPLL. This is the hclk required by the PIPE interface.

Figure 103. Use ATX PLL or fPLL for Gen1/Gen2 x4 Mode



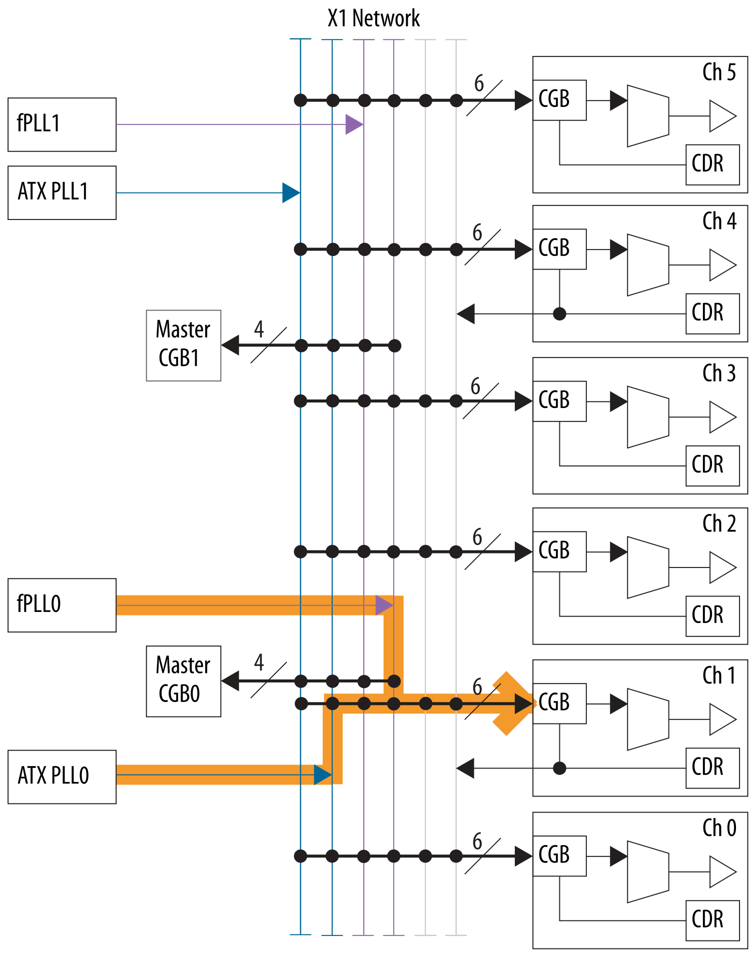
Notes:
1. The figure shown is just one possible combination for the PCIe Gen1/Gen2 x4 mode.
2. The x6 and x24 clock networks are used for channel bonding applications.
3. Each master CGB drives one set of x6 clock lines.
4. Gen1/Gen2 x4 modes use the ATX PLL or fPLL only.
5. Use the pll_pcie_clk from either the ATX or fPLL. This is the hclk required by the PIPE interface.
6. In this case the Master PCS channel is logical channel 2 (physical channel 4).

Figure 104. Use ATX PLL or fPLL for Gen1/Gen2 x16 Mode



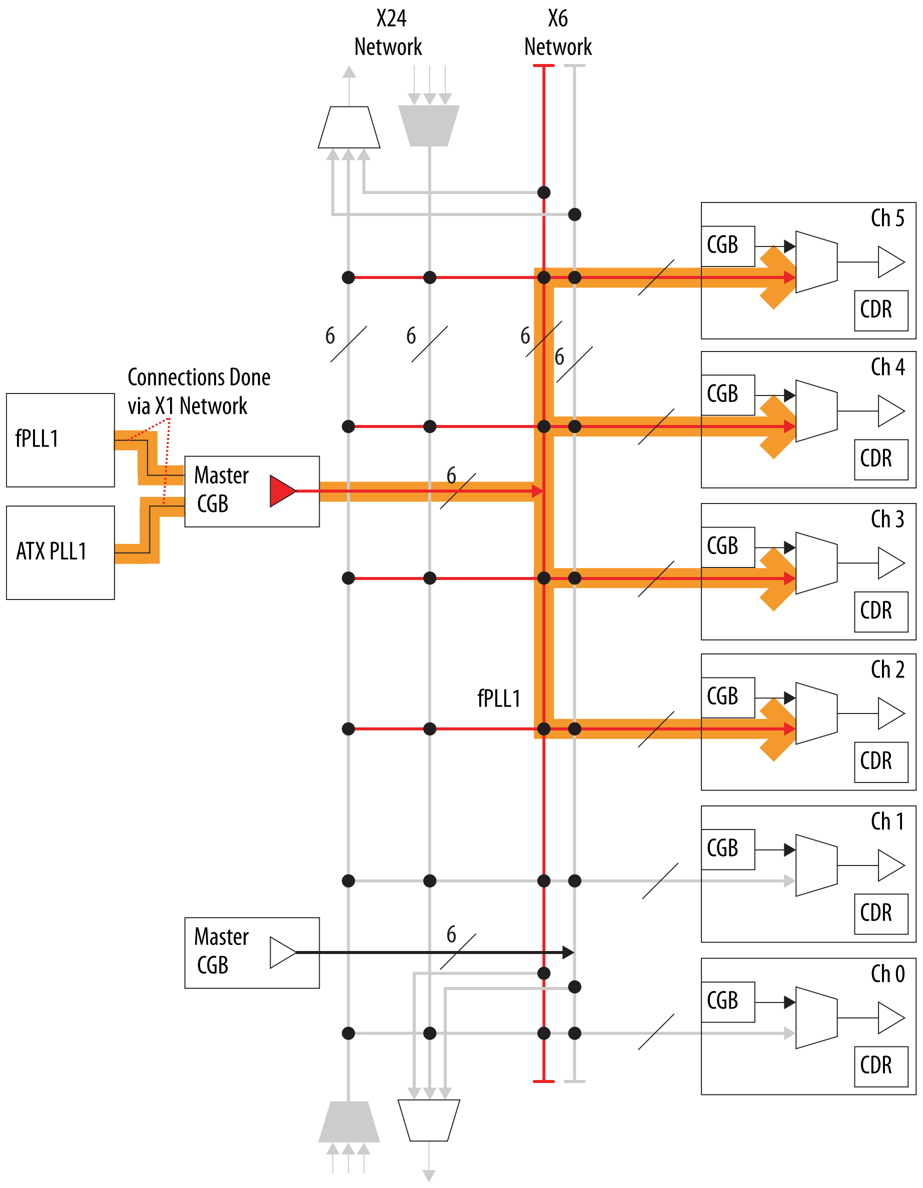
Notes:
1. The figure shown is just one possible combination for the PCIe Gen1/Gen2 x16 mode.
2. The x6 and x24 clock networks are used for channel bonding applications.
3. Each master CGB drives one set of x6 clock lines. The x6 lines further drive the x24 lines.
4. Gen1/Gen2 x16 modes use the fPLL only.
5. Use the pll_pcie_clk from the fPLL. This is the hclk required by the PIPE interface.
6. In this case, the Master PCS channel is logical channel 8 (physical channel 4 of bank 1).

Figure 105. Use ATX PLL or fPLL for Gen1/Gen2/Gen3 x1 Mode



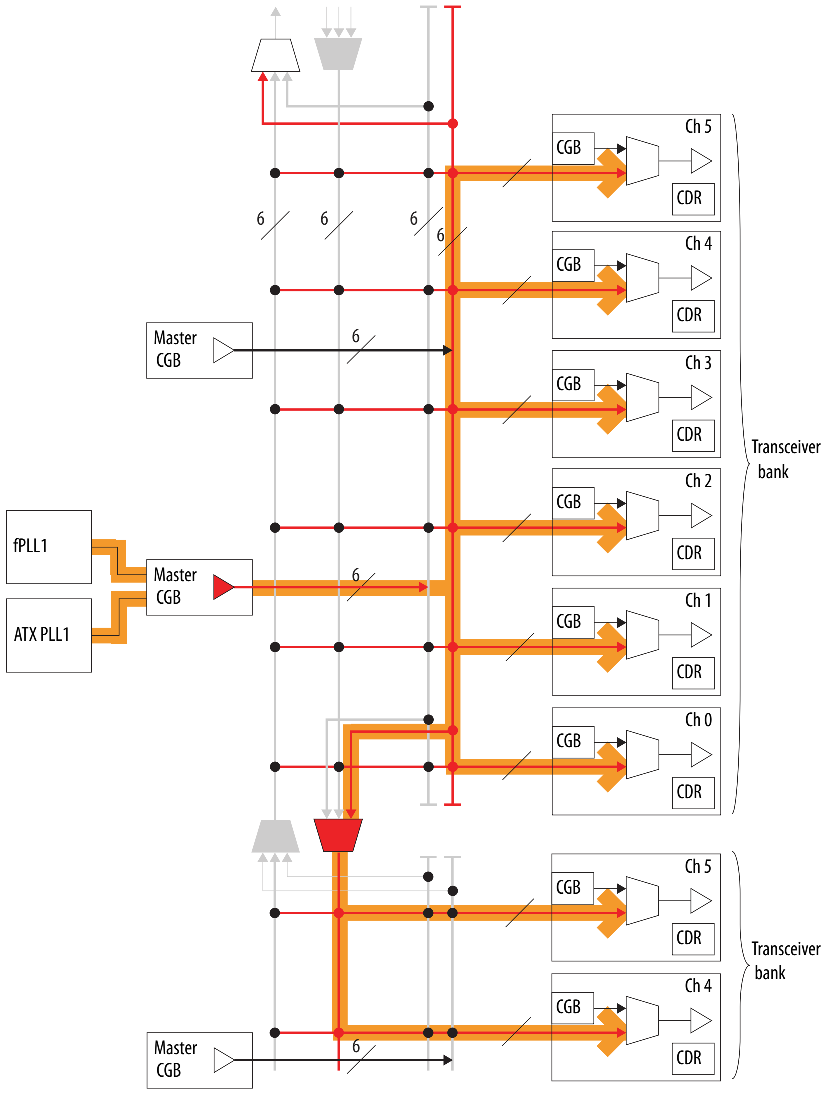
Notes:
1. The figure shown is just one possible combination for the PCIe Gen1/Gen2/Gen3 x1 mode.
2. Gen1/Gen2 modes use the fPLL only.
3. Gen3 mode uses the ATX PLL only.
4. Use the pll_pcie_clk from the fPLL, configured as Gen1/Gen2. This is the hclk required by the PIPE interface.
5. Select the number of TX PLLs (2) in the Native PHY IP core wizard.

Figure 106. Use ATX PLL or fPLL for Gen1/Gen2/Gen3 x4 Mode



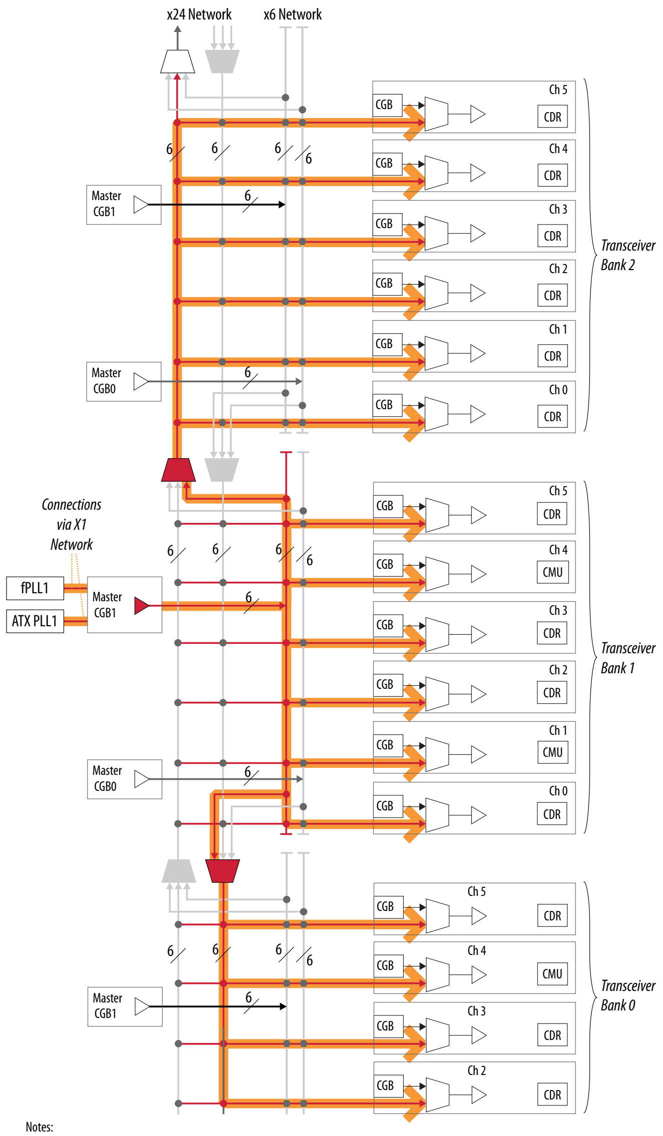
Notes:
1. The figure shown is just one possible combination for the PCIe Gen1/Gen2/Gen3 x4 mode.
2. The x6 and x24 clock networks are used for channel bonding applications.
3. Each master CGB drives one set of x6 clock lines.
4. Gen1/Gen2 modes use the fPLL only.
5. Gen3 mode uses the ATX PLL only.
6. Use the pll_pcie_clk from the fPLL, configured as Gen1/Gen2. This is the hclk required by the PIPE interface.

Figure 107. Use ATX PLL or fPLL for Gen1/Gen2/Gen3 x8 Mode



Notes:
1. The figure shown is just one possible combination for the PCIe Gen1/Gen2/Gen3 x8 mode.
2. The x6 and x24 clock networks are used for channel bonding applications.
3. Each master CGB drives one set of x6 clock lines. The x6 lines further drive the x24 lines.
4. Gen1/Gen2 x8 modes use the fPLL only.
5. Gen3 mode uses the ATX PLL only.
6. Use the pll_pcie_clk from the fPLL, configured as Gen1/Gen2. This is the hclk required by the PIPE interface.

Figure 108. Use ATX PLL or fPLL for Gen1/Gen2/Gen3 x16 Mode



Notes:
1. The figure shown is just one possible combination for the PCIe Gen3 x16 mode.
2. The x6 and x24 clock networks are used for channel bonding applications.
3. Each master CGB drives one set of x6 clock lines. The x6 lines further drive the x24 lines.
4. Gen1/Gen2 x16 modes use the fPLL only.
5. Gen3 mode uses the ATX PLL only.
6. Use the pll_pcie_clk from the fPLL. This is the hclk required by the PIPE interface.
7. In this case, the Master PCS channel is logical channel 8 (physical channel 4 of bank 1).

Related Information

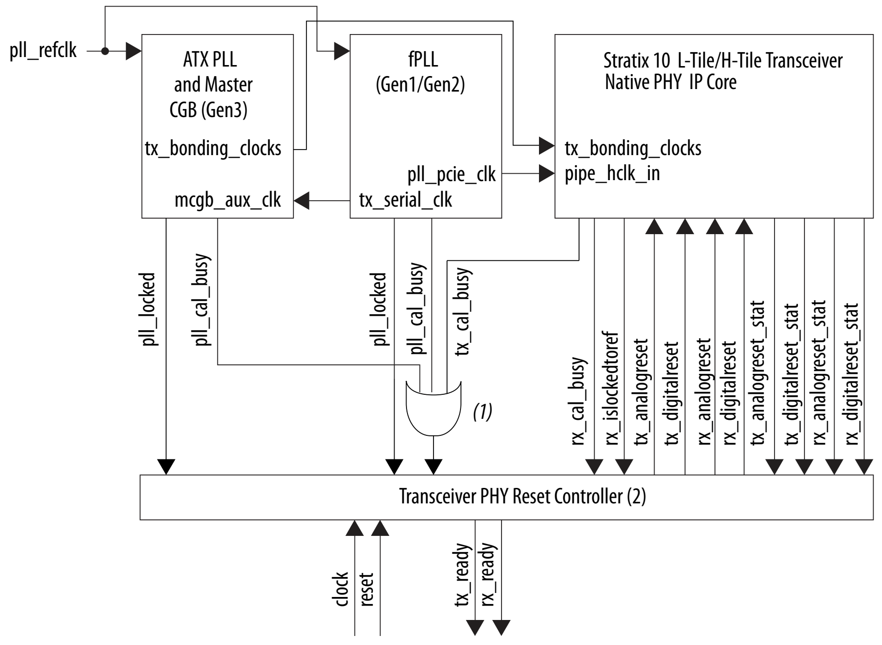
- PLLs on page 251
- Using PLLs and Clock Networks on page 304

2.5.1.4. How to Implement PCI Express (PIPE) in Intel Stratix 10 Transceivers

You must be familiar with the Standard PCS architecture, Gen3 PCS architecture, PLL architecture, and the reset controller before implementing the PCI Express protocol.

1. Instantiate the **Stratix 10 L-Tile/H-Tile Transceiver Native PHY IP** from the IP Catalog (**Installed IP > Library > Interface Protocols > Transceiver PHY > Stratix 10 L-Tile/H-Tile Transceiver Native PHY**).
2. Select **Gen1/Gen2/Gen3 PIPE** from the **Stratix 10 Transceiver configuration rules** list, located under **Datapath Options**.
3. Use the parameter values in the tables in Native PHY IP Core Parameter Settings for PIPE on page 185 as a starting point. Alternatively, you can use ***Stratix 10 L-Tile/H-Tile Transceiver Native PHY Presets*** . You can then modify the settings to meet your specific requirements.
4. Click **Finish** to generate the Native PHY IP core (this is your RTL file).
5. Instantiate and configure your PLL IP core.
6. Create a transceiver reset controller. You can use your own reset controller or use the Transceiver PHY Reset Controller.
7. Connect the Native PHY IP core to the PLL IP core and the Transceiver PHY Reset Controller. Use the information in Native PHY IP Core Ports for PIPE on page 196 to connect the ports.
8. Simulate your design to verify its functionality.

Figure 109. Native PHY IP Core Connection Guidelines for a PIPE Gen3 Design



Notes:
1. If you enable the input pll_cal_busy port in the Transceiver PHY reset controller, you can connect the pll_cal_busy output signals from the PLLs directly to the input port on the Transceiver PHY reset controller without ORing the tx_cal_busy and pll_cal_busy signals.
2. If you are using the Transceiver PHY reset controller, you must configure the TX digital reset mode and RX digital reset mode to Manual to avoid resetting the Auto Speed Negotiation (ASN) block which handles the rate switch whenever the channel PCS is reset.

For additional details refer to the Reset chapter.

Related Information

- PLLs on page 251
- Resetting Transceiver Channels on page 318
- Intel Stratix 10 Standard PCS Architecture on page 366

2.5.1.5. Native PHY IP Core Parameter Settings for PIPE

This section contains the recommended parameter values for this protocol. Refer to *Using the Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY IP Core* for the full range of parameter values.

Table 101. General, Common PMA, and Datapath Options

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
Message level for rule violations	**Error**	**Error**	**Error**
VCCR_GXB and VCCT_GXB supply voltage for the Transceiver	**1_1V, 1_0V** (31)	**1_1V, 1_0V** (31)	**1_1V, 1_0V** (31)
Transceiver Link Type	**sr, lr**	**sr, lr**	**sr, lr**
Transceiver channel type	**GX**	**GX**	**GX**
Transceiver configuration rules	**Gen 1 PIPE**	**Gen 2 PIPE**	**Gen 3 PIPE**
PMA configuration rules	**basic**	**basic**	**basic**
Transceiver mode	**TX / RX Duplex**	**TX / RX Duplex**	**TX / RX Duplex**
Number of data channels	x1: **1** x2: **2** x4: **4** x8: **8** x16: **16**	x1: **1** x2: **2** x4: **4** x8: **8** x16: **16**	x1: **1** x2: **2** x4: **4** x8: **8** x16: **16**
Data rate	**2.5 Gbps**	**5 Gbps**	**5 Gbps**(32)
Enable datapath and interface reconfiguration	**Disable**	**Disable**	**Disable**
Enable simplified data interface	**Optional** (33)	**Optional** (33)	**Optional** (33)
Enable double rate transfer mode	**Off**	**Off**	**Off**

Table 102. TX PMA Options

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
TX channel bonding mode	**x1 - Not bonded** **x2, x4, x8, x16 - PMA & PCS Bonding**	**x1 - Not bonded**	**x1 - Not bonded**

continued...

(31) Refer to the *Intel Stratix 10 Device Datasheet* for details about the minimum, typical, and maximum supply voltage specifications.

(32) The PIPE is configured in Gen1/Gen2 during Power Up. Gen3 PCS is configured for 8 Gbps.

(33) For additional details, refer to Native PHY IP Core Parameter Settings for PIPE on page 185 for bit settings when the Simplified Data Interface is disabled.

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
		x2, x4, x8, x16 - PMA & PCS Bonding	**x2, x4, x8, x16 - PMA & PCS Bonding**
PCS TX channel bonding master	**Auto** (34)	**Auto** (34)	**Auto** (34)
Actual PCS TX channel bonding master	x2: **1** (35) x4: **2** (35) x8: **4** (35) x16: **8** (35)	x2: **1** (35) x4: **2** (35) x8: **4** (35) x16: **8** (35)	x2: **1** (35) x4: **2** (35) x8: **4** (35) x16: **8** (35)
PCS reset sequence	**Simultaneous**	**Simultaneous**	**Simultaneous**
TX local clock division factor	**1**	**1**	**1**
Number of TX PLL clock inputs per channel	**1**	**1**	x1: **2** x2, x4, x8, x16: **1**
Initial TX PLL clock input selection	**0**	**0**	**0**
Enable tx_pma_iqtxrx_clkout port	**On** **Off**	**On** **Off**	**On** **Off**
Enable tx_pma_elecidleport	**Off**	**Off**	**Off**

Table 103. RX PMA Options

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
Number of CDR reference clocks	**1**	**1**	**1**
Selected CDR reference clock	**0**	**0**	**0**
Selected CDR reference clock frequency	**100 MHz**	**100 MHz**	**100 MHz**
PPM detector threshold	**1000**	**1000**	**1000**
Enable rx_pma_iqtxrx_clkout port	**Off**	**Off**	**Off**
Enable rx_pma_clkslip port	**Off**	**Off**	**Off**
Enable rx_is_lockedtodata port	**On** **Off**	**On** **Off**	**On** **Off**
Enable rx_is_lockedtoref port	**On** **Off**	**On** **Off**	**On** **Off**
Enable rx_set_locktodata and rx_set_locktoref ports	**On**	**On**	**On**

continued...

(34) Setting this parameter is placement-dependent. In AUTO mode, the Native PHY IP core will select the middle-most channel of the configuration as the default PCS TX channel bonding master. You must ensure that this selected channel is physically placed as Ch1 or Ch4 of the transceiver bank. Else, use the manual selection for the PCS TX channel bonding master to select a channel that can be physically placed as Ch1 or Ch4 of the transceiver bank. Refer to the *How to Place Channels for PIPE Configurations* section for more details.

(35) For **PCS TX channel bonding master** in Auto mode, this is the Actual PCS TX channel bonding master value. For other selections of the PCS TX channel bonding master, the value of the **Actual PCS TX channel bonding master** is the same as the **PCS TX channel bonding master** value.

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
	Off	**Off**	**Off**
Enable PRBS verifier control and status ports	**Off**	**Off**	**Off**
Enable rx_seriallpbken port	**Off**	**Off**	**Off**

Table 104. Standard PCS Options

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
Standard PCS / PMA interface width	**10**	**10**	**10**(36)
FPGA fabric / Standard TX PCS interface width	**8, 16**	**16**	**32**
FPGA fabric / Standard RX PCS interface width	**8, 16**	**16**	**32**
Enable 'Standard PCS' low latency Mode	**Off**	**Off**	**Off**
TX byte serializer mode	**Disabled** (if FPGA fabric/standard TX/RX PCS interface width =8), **Serialize x2** (if FPGA fabric/standard TX/RX PCS interface width =16)	**Serialize x2**	**Serialize x4**
RX byte deserializer mode	**Disabled** (if FPGA fabric/standard TX/RX PCS interface width =8), **Deserialize x2** (if FPGA fabric/standard TX/RX PCS interface width =16)	**Deserialize x2**	**Deserialize x4**
Enable TX 8B/10B encoder	**On**	**On**	**On**
Enable TX 8B/10B disparity control	**On**	**On**	**On**
Enable RX 8B/10B decoder	**On**	**On**	**On**
RX rate match FIFO mode	**PIPE** **PIPE 0ppm**	**PIPE** **PIPE 0ppm**	**PIPE** **PIPE 0ppm**
RX rate match insert/delete -ve pattern (hex)	**0x0002f17c (K28.5/ K28.0/)**	**0x0002f17c (K28.5/ K28.0/)**	**0x0002f17c (K28.5/ K28.0/)**
RX rate match insert/delete +ve pattern (hex)	**0x000d0e83 (K28.5/ K28.0/)**	**0x000d0e83 (K28.5/ K28.0/)**	**0x000d0e83 (K28.5/ K28.0/)**
Enable rx_std_rmfifo_full port	**Off**	**Off**	**Off**
Enable rx_std_rmfifo_empty port	**Off**	**Off**	**Off**
PCI Express Gen3 rate match FIFO mode	**Bypass**	**Bypass**	**Bypass**
Enable TX bitslip	**Off**	**Off**	**Off**

continued...

(36) "The PIPE is configured in Gen1/Gen2 during power-up. Refer to table *PIPE Gen3 32 bit PCS clock rates* for more details on the PIPE interface widths and clock frequencies for Gen3 capable configurations.

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
Enable tx_std_bitslipboundarysel port	Off	Off	Off
RX word aligner mode	synchronous state machine	synchronous state machine	synchronous state machine
RX word aligner pattern length	10	10	10
RX word aligner pattern (hex)	0x0000 00000000017c (/ K28.5/)	0x0000 00000000017c (/ K28.5/)	0x0000 00000000017c (/ K28.5/)
Number of word alignment patterns to achieve sync	3	3	3
Number of word alignment patterns to lose sync	16	16	16
Number of valid data words to decrement error count	15	15	15
Enable fast sync status reporting for deterministic latency SM	Off	Off	Off
Enable rx_std_wa_patternalign port	Off	Off	Off
Enable rx_std_wa_a1a2size port	Off	Off	Off
Enable rx_std_bitslipboundarysel port	Off	Off	Off
Enable rx_bitslip port	Off	Off	Off
Enable TX bit reversal	Off	Off	Off
Enable TX byte reversal	Off	Off	Off
Enable TX polarity inversion	Off	Off	Off
Enable tx_polinv port	Off	Off	Off
Enable RX bit reversal	Off	Off	Off
Enable rx_std_bitrev_ena port	Off	Off	Off
Enable RX byte reversal	Off	Off	Off
Enable rx_std_byterev_ena port	Off	Off	Off
Enable RX polarity inversion	Off	Off	Off
Enable rx_polinv port	Off	Off	Off
Enable rx_std_signaldetect port	Off	Off	Off
Enable PCIe dynamic datarate switch ports	Off	On	On
Enable PCIe electrical idle control and status ports	On	On	On
Enable PCIe pipe_hclk_in and pipe_hclk_out ports	On	On	On

Table 105. PCS-Core Interface Options

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
Enable PCS reset status ports	Off	Off	Off
TX Core Interface FIFO mode	Phase compensation	Phase compensation	Phase compensation

continued...

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
TX FIFO partially full threshold	5	5	5
TX FIFO partially empty threshold	2	2	2
Enable tx_fifo_full port	Off	Off	Off
Enable tx_fifo_empty port	Off	Off	Off
Enable tx_fifo_pfull port	Off	Off	Off
Enable tx_fifo_pempty port	Off	Off	Off
Enable tx_dll_lock port	Off	Off	Off
RX Core Interface FIFO mode (PCS FIFO - Core FIFO)	Phase compensation	Phase compensation	Phase compensation
RX FIFO partially full threshold	5	5	5
RX FIFO partially empty threshold	2	2	2
Enable RX FIFO alignment word deletion (Interlaken)	Off	Off	Off
Enable RX FIFO control word deletion (Interlaken)	Off	Off	Off
Enable rx_data_valid port	Off	Off	Off
Enable rx_fifo_full port	Off	Off	Off
Enable rx_fifo_empty port	Off	Off	Off
Enable rx_fifo_pfull port	Off	Off	Off
Enable rx_fifo_pempty port	Off	Off	Off
Enable rx_fifo_del port (10GBASE-R)	Off	Off	Off
Enable rx_fifo_insert port (10GBASE-R)	Off	Off	Off
Enable rx_fifo_rd_en port	Off	Off	Off
Enable rx_fifo_align_clr port (Interlaken)	Off	Off	Off
Selected tx_clkout clock source	PCS clkout	PCS clkout	PCS clkout
Enable tx_clkout2 port	On	On	On
Selected tx_clkout2 clock source	PCS clkout x2	PCS clkout x2	PCS clkout x2
TX pma_div_clkout division factor	Disabled	Disabled	Disabled
Selected tx_coreclkin clock network	Dedicated Clock	Dedicated Clock	Dedicated Clock
Selected TX PCS bonding clock network	Dedicated Clock	Dedicated Clock	Dedicated Clock
Enable tx_coreclkin2 port	Off	Off	Off
Selected rx_clkout clock source	PCS clkout	PCS clkout	PCS clkout
Enable rx_clkout2 port	Off	Off	Off
Selected rx_clkout2 clock source	Off	Off	Off
RX pma_div_clkout division factor	Disabled	Disabled	Disabled

continued...

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
Selected rx_coreclkin clock network	**Dedicated Clock**	**Dedicated Clock**	**Dedicated Clock**
Latency Measurement Options	**Off**	**Off**	**Off**
Enable latency measurement ports	**Off**	**Off**	**Off**

Table 106. Parameters for the Native PHY IP Core in PIPE Gen1, Gen2, Gen3 Modes - Analog PMA Settings

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
TX PMA analog mode rules	**pcie_cable**	**pcie_cable**	**pcie_cable**
Use default TX PMA analog settings	**On** / **Off** (Use the values listed below if this option is disabled)	**On** / **Off** (Use the values listed below if this option is disabled)	**On** / **Off** (Use the values listed below if this option is disabled)
Output swing level (VOD)	**24**	**24**	**31**
Pre-emphasis First pre-tap polarity	**Negative**	**Negative**	**Negative**
Pre-emphasis first pre-tap magnitude	**0**	**0**	**0**
Pre-emphasis First post-tap polarity	**Negative**	**Negative**	**Negative**
Pre-emphasis first post tap magnitude	**8**	**8**	**0**
Slew rate control	**4**	**5**	**5**
On-chip termination	**R_r1**	**R_r1**	**R_r1**
High speed compensation	**Disable**	**Disable**	**Enable**
RX PMA analog mode rules	**User_custom**	**User_custom**	**User_custom**
Use default RX PMA analog settings	**On** / **Off** (Use the values listed below if this option is **Off**)	**On** / **Off** (Use the values listed below if this option is **Off**)	**On** / **Off** (Use the values listed below if this option is **Off**)
RX Adaptation mode	**Manual CTLE, Manual VGA, DFE Off**	**Manual CTLE, Manual VGA, DFE Off**	**Adaptive Mode for PCIe Gen3**
RX on-chip termination	**R_r2**	**R_r2**	**R_r2**
CTLE AC gain	**10**	**10**	**10**
CTLE EQ gain	**3**	**3**	**3**
VGA DC gain	**5**	**5**	**5**
Provide sample QSF assignments	**On** / **Off**	**On** / **Off**	**On** / **Off**

Table 107. Dynamic Reconfiguration Options

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
Enable dynamic reconfiguration	**On** / **Off**	**On** / **Off**	**On** / **Off**
Enable Native PHY debug master endpoint	**On** / **Off**	**On** / **Off**	**On** / **Off**
Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE	**Off**	**Off**	**Off**

continued...

Send Feedback

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
Share reconfiguration interface	**On** / **Off**	**On** / **Off**	**On** / **Off**
Enable rcfg_tx_digitalreset_release_ctrl port	**Off**	**Off**	**Off**
Enable capability registers	**On** / **Off**	**On** / **Off**	**On** / **Off**
Set user-defined IP identifier	**<Identifier>**	**<Identifier>**	**<Identifier>**
Enable control and status registers	**On** / **Off**	**On** / **Off**	**On** / **Off**
Enable prbs soft accumulators	**On** / **Off**	**On** / **Off**	**On** / **Off**
Configuration file prefix	`<File prefix>`	`<File prefix>`	`<File prefix>`
Generate SystemVerilog package file	**On** / **Off**	**On** / **Off**	**On** / **Off**
Generate C header file	**On** / **Off**	**On** / **Off**	**On** / **Off**
Generate MIF (Memory initialization file)	**On** / **Off**	**On** / **Off**	**On** / **Off**
Enable multiple reconfiguration profiles	**Off**	**Off**	**Off**
Enable embedded reconfiguration streamer	**Off**	**Off**	**Off**

Table 108. Design Example Parameters

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
TX PLL type	**fPLL**	**fPLL**	**fPLL**
TX PLL reference clock frequency	**100.0 MHz**	**100.0 MHz**	**100.0 MHz**
Use tx_clkout2 as source for tx_coreclkin	**Disable**	**Disable**	**Disable**
Use rx_clkout2 as source for rx_coreclkin	**Disable**	**Disable**	**Disable**
Enable fast simulations	**On** (for simulation) **Off** (for hardware)	**On** (for simulation) **Off** (for hardware)	**On** (for simulation) **Off** (for hardware)
Design example filename	`<File name>`	`<File name>`	`<File name>`
Generate parameter documentation file	**On** / **Off**	**On** / **Off**	**On** / **Off**

Note: You must use the TX PLL type – fPLL for generating the design example for PCIe PIPE Gen3 configurations of all lane widths. For the PCIe PIPE Gen3 configuration, the ATX PLL is generated automatically.

Note: The signals in the left-most column are automatically mapped to a subset of a 80-bit/channel `tx_parallel_data` word when the Simplified Interface is enabled.

Note: You must set the I/O Standard and Termination Setting for the dedicated REFCLK pins through the Intel Quartus Prime Pro Edition Assignment Editor. The two available PIPE refclk assignment settings are:

- **refclk_standard** only
- **refclk_term_tristate**

When you use these pins, AC-couple or DC-couple them. For the HCSL I/O standard, it only supports DC coupling. In the PCI Express configuration, DC-coupling is allowed on the REFCLK if the selected REFCLK I/O standard is HCSL.

Refer to the *Dedicated Reference Clock Pins* section for more details.

Refer to the *Intel Stratix 10 Device Family Pin Connection Guidelines*.

Related Information

- Using the Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY Intel Stratix 10 FPGA IP Core on page 106
- How to Place Channels for PIPE Configurations on page 207
- Dedicated Reference Clock Pins on page 279
- Intel Stratix 10 Device Family Pin Connection Guidelines
- Intel Stratix 10 Device Datasheet

2.5.1.6. fPLL IP Core Parameter Settings for PIPE

This section contains the recommended parameter values for this protocol. Refer to *Using the Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY IP Core* for the full range of parameter values.

Table 109. fPLL Parameters for PCIe PIPE Gen1, Gen2, and Gen3 Modes

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
fPLL Mode	Transceiver	Transceiver	Transceiver
Message level for rule violations	Error	Error	Error
Protocol mode	PCIe G1	PCIe G2	PCIe G2
Bandwidth	Low, medium, high	Low, medium, high	Low, medium, high
Number of PLL reference clocks	1	1	1
Selected reference clock source	0	0	0
Enable fractional mode	Off	Off	Off
VCCR_GXB and VCCT_GXB supply voltage for the transceiver	1_0V, 1_1V	1_0V, 1_1V	1_0V, 1_1V
Enable PCIe clock output port	On	On	On
PLL output frequency	1250MHz	2500MHz	2500MHz
PLL output datarate	2500Mbps	5000Mbps	5000Mbps
PLL integer mode reference clock frequency	100MHz	100MHz	100MHz
Configure counters manually	Off	Off	Off

continued...

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
Multiply factor (M-counter)	N/A	N/A	N/A
Divide factor (N-counter)	N/A	N/A	N/A
Divide factor (L-counter)	N/A	N/A	N/A
Include Master clock generation block	**x1 — Off** **x2, x4, x8, x16 — On**	**x1 — Off** **x2, x4, x8, x16 — On**	**Off**
Clock division factor	**1**	**1**	N/A
Enable x24 non-bonded high – speed clock output port	**Off**	**Off**	**Off**
Enable PCIe clock switch interface	**Off**	**Off**	**Off**
Enable mcgb_rst and mcgb_rst_stat ports	**Off**	**Off**	**Off**
Number of auxiliary MCGB clock input ports	**0**	**0**	**0**
MCGB input clock frequency	**1250MHz**	**2500MHz**	**2500MHz**
MCGB output data rate	**2500Mbps**	**5000Mbps**	**5000Mbps**
Enable bonding clock output ports	**x1 — Off** **x2, x4, x8, x16 — On**	**x1 — Off** **x2, x4, x8, x16 — On**	**Off**
PMA interface width	**10**	**10**	N/A

Table 110. Master Clock Generation Block Options

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE (For Gen1/Gen2 Speeds)
Include Master Clock Generation Block	x1: **Off** x2, x4, x8, x16: **On**	x1: **Off** x2, x4, x8, x16: **On**	x1, x2, x4, x8, x16: **Off**
Clock division factor	x1: N/A x2, x4, x8, x16: **1**	x1: N/A x2, x4, x8, x16: **1**	x1, x2, x4, x8, x16: N/A
Enable x6/xN non-bonded high-speed clock output port	x1: N/A x2, x4, x8, x16: **Off**	x1: N/A x2, x4, x8, x16: **Off**	x1, x2, x4, x8, x16: **Off**
Enable PCIe clock switch interface	x1: N/A x2, x4, x8, x16: **Off**	x1: N/A x2, x4, x8, x16: **Off**	x1, x2, x4, x8, x16: **Off**
Number of auxiliary MCGB clock input ports	x1: N/A x2, x4, x8, x16: **0**	x1: N/A x2, x4, x8, x16: **0**	x1, x2, x4, x8, x16: N/A
MCGB input clock frequency	**1250 MHz**	**2500 MHz**	**2500 MHz**
MCGB output data rate	**2500 Mbps**	**5000 Mbps**	**5000 Mbps**
Enable bonding clock output ports	x1: N/A x2, x4, x8, x16: **On**	x1: N/A x2, x4, x8, x16: **On**	x1, x2, x4, x8, x16: **Off**
PMA interface width	x1: N/A x2, x4, x8, x16: **10**	x1: N/A x2, x4, x8, x16: **10**	x1, x2, x4, x8, x16: N/A

Related Information

- Using the Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY Intel Stratix 10 FPGA IP Core on page 106
- Intel Stratix 10 Device Datasheet

2.5.1.7. ATX PLL IP Core Parameter Settings for PIPE

This section contains the recommended parameter values for this protocol. Refer to *Using the Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY IP Core* for the full range of parameter values.

Table 111. ATX PLL Parameters for PCIe PIPE Gen1, Gen2, and Gen3 Modes

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
Message level for rule violations	Error	Error	Error
Protocol mode	PCIe Gen 1	PCIe G2	PCIe G3
Bandwidth	Low, medium, high	Low, medium, high	Low, medium, high
Number of PLL reference clocks	1	1	1
Selected reference clock source	0	0	0
VCCR_GXB and VCCT_GXB supply voltage for the transceiver	1_0V, 1_1V	1_0V, 1_1V	1_0V, 1_1V
Primary PLL clock output buffer	GX clock output buffer	GX clock output buffer	GX clock output buffer
Enable GX clock output port (tx_serial_clk)	On	On	On
Enable GXT clock output port to above ATX PLL (gxt_output_to_abv_atx)	Off	Off	Off
Enable GXT clock output port to below ATX PLL (gxt_output_to_blw_atx)	Off	Off	Off
Enable GXT local clock output porttx_serial_clk_gxt)	Off	Off	Off
Enable GXT clock input port from above ATX PLL (gxt_input_from_abv_atx)	Off	Off	Off
Enable GXT clock input port from below ATX PLL (gxt_input_from_blw_atx)	Off	Off	Off
Enable PCIe clock output port	On	On	Off (37)
Enable ATX to fPLL cascade clock output port	N/A	N/A	N/A
Enable GXT clock buffer to above ATX PLL	Off	Off	Off
Enable GXT clock buffer to below ATX PLL	Off	Off	Off
GXT output clock source	Disabled	Disabled	Disabled
PLL output frequency	1250MHz	2500MHz	4000MHz
PLL output datarate	2500Mbps	5000Mbps	8000Mbps

continued...

(37) Use the `pll_pcie_clk` output port from the fPLL to drive the hclk.

Parameter	Gen1 PIPE	Gen2 PIPE	Gen3 PIPE
PLL auto mode reference clock frequency(integer)	**100MHz**	**100MHz**	**100MHz**
Configure counters manually	**Off**	**Off**	**Off**
Multiply factor (M-counter)	N/A	N/A	N/A
Divide factor (N-counter)	N/A	N/A	N/A
Divide factor (L-counter)	N/A	N/A	N/A
Include Master clock generation block	**x1 — Off** **x2, x4, x8, x16 — On**	**x1 — Off** **x2, x4, x8, x16 — On**	**x1 — Off** **x2, x4, x8, x16 — On**
Clock division factor	**1**	**1**	**1**
Enable x24 non-bonded high – speed clock output port	**Off**	**Off**	**Off**
Enable PCIe clock switch interface	**Off**	**Off**	**On**
Enable mcgb_rst and mcgb_rst_stat ports	**Off**	**Off**	**Off**
Number of auxiliary MCGB clock input ports	**0**	**0**	**x1 — N/A** **x2, x4, x8, x16 — 1**
MCGB input clock frequency	**1250MHz**	**2500MHz**	**2500MHz**
MCGB output data rate	**2500Mbps**	**5000Mbps**	**8000Mbps**
Enable bonding clock output ports	**x1 — Off** **x2, x4, x8, x16 — On**	**x1 — Off** **x2, x4, x8, x16 — On**	**x1 — Off** **x2, x4, x8, x16 — On**
PMA interface width	**10**	**10**	**32**
Enable Dynamic reconfiguration	**On** / **Off**	**On** / **Off**	**On** / **Off**
Enable Native PHY debug master endpoint	**On** / **Off**	**On** / **Off**	**On** / **Off**
Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE	**Off**	**Off**	**Off**
Enable capability registers	**On** / **Off**	**On** / **Off**	**On** / **Off**
Set user – defined IP identifier	`<IP identifier>`	`<IP identifier>`	`<IP identifier>`
Enable control and status registers	**On** / **Off**	**On** / **Off**	**On** / **Off**
Configuration file prefix	`<File prefix>`	`<File prefix>`	`<File prefix>`
Generate SystemVerilog package file	**On** / **Off**	**On** / **Off**	**On** / **Off**
Generate C header file	**On** / **Off**	**On** / **Off**	**On** / **Off**
Generate MIF (Memory Initialization file)	**On** / **Off**	**On** / **Off**	**On** / **Off**
Enable multiple reconfiguration profiles	**Off**	**Off**	**Off**
Enable embedded reconfiguration streamer	**Off**	**Off**	**Off**
Generate reduced reconfiguration files	**Off**	**Off**	**Off**

Related Information

Using the Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY Intel Stratix 10 FPGA IP Core on page 106

2.5.1.8. Native PHY IP Core Ports for PIPE

Figure 110. Signals and Ports of the Native PHY IP Core for PIPE

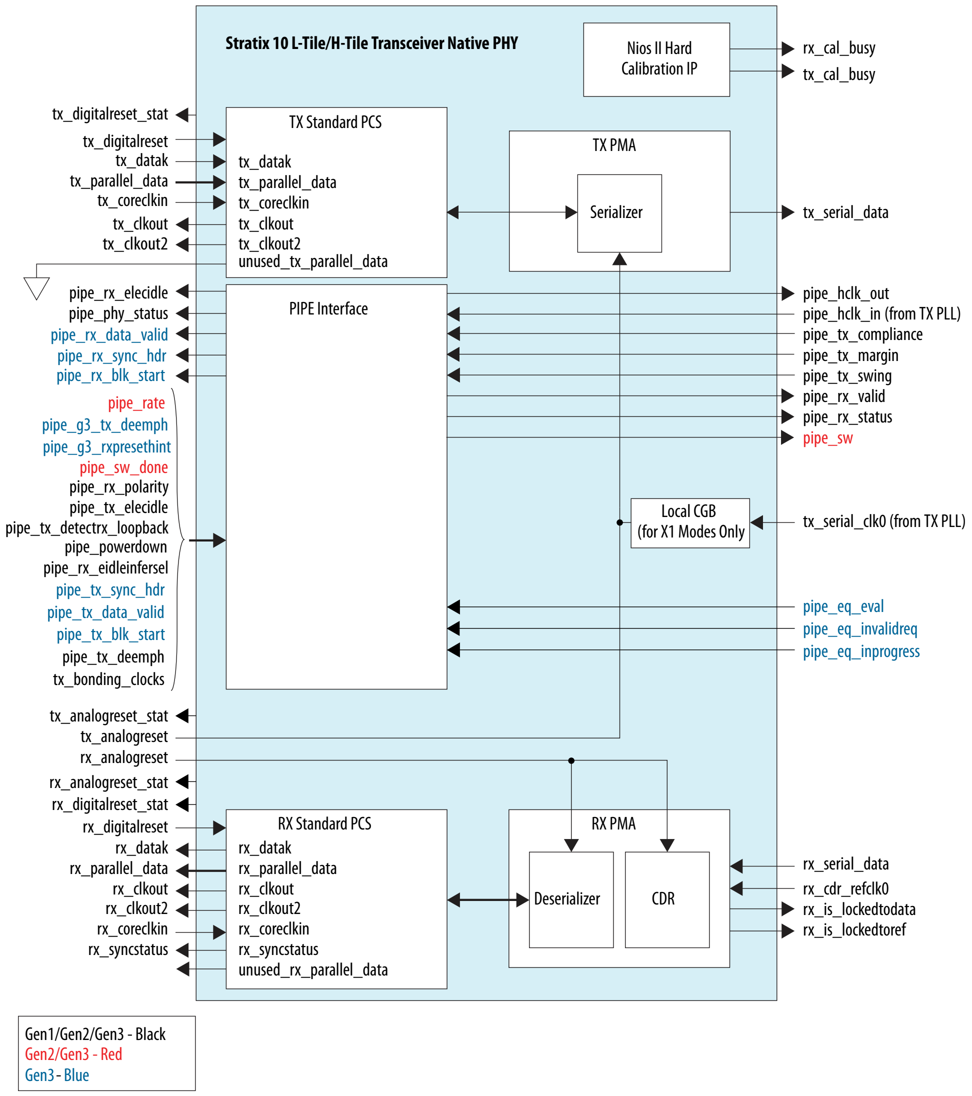


Table 112. Ports of the Native PHY IP Core in PIPE Mode

Port	Direction	Clock Domain	Description
Clocks			
`rx_cdr_refclk0`	In	N/A	The 100 MHz input reference clock source for the PHY's TX PLL and RX CDR.
`tx_serial_clk0` / `tx_serial_clk1`	In	N/A	The high speed serial clock generated by the PLL. Note: The `tx_serial_clk1` is only used in Gen3 x 1 mode.

continued...

Port	Direction	Clock Domain	Description
pipe_hclk_in[0:0]	In	N/A	The 500 MHz clock used for the ASN block. This clock is generated by the PLL, configured for Gen1/Gen2. Note: For Gen3 designs, use from the fPLL that is used for Gen1/Gen2.
pipe_hclk_out[0:0]	Out	N/A	The 500 MHz clock output provided to the PHY - MAC interface.
PIPE Input from PHY - MAC Layer			
tx_parallel_data[31:0]	In	tx_coreclkin	The TX parallel data driven from the MAC. For Gen1 this can be 8 or 16 bits. For Gen2 this is 16 bits. For Gen3 this is 32 bits. Note: unused_tx_parallel_data should be tied to '0'. Active High. Refer to Table 113 on page 201 for more details.
tx_datak[3:0], [1:0], or [0]	In	tx_coreclkin	The data and control indicator for the transmitted data. For Gen1 or Gen2, when 0, indicates that tx_parallel_data is data, when 1, indicates that tx_parallel_data is control. For Gen3, bit[0] corresponds to tx_parallel_data[7:0], bit[1] corresponds to tx_parallel_data[15:8], and so on. Active High. Refer to Table 113 on page 201 for more details.
pipe_tx_sync_hdr[(2*N-1):0]	In	tx_coreclkin	For Gen3, indicates whether the 130-bit block transmitted is a Data or Control Ordered Set Block. The following encodings are defined: 2'b10: Data block 2'b01: Control Ordered Set Block This value is read when pipe_tx_blk_start = 1b'1. Refer to *Lane Level Encoding* in the *PCI Express Base Specification, Rev. 3.0* for a detailed explanation of data transmission and reception using 128b/130b encoding and decoding. Not used for Gen1 and Gen2 datarates. Active High
pipe_tx_blk_start[(N-1):0]	In	tx_coreclkin	For Gen3, specifies the start block byte location for TX data in the 128-bit block data. Used when the interface between the PCS and PHY-MAC (FPGA Core) is 32 bits. Not used for Gen1 and Gen2 datarates. Active High
pipe_tx_elecidle[(N-1):0]	In	Asynchronous	Forces the transmit output to electrical idle. Refer to the *Intel PHY Interface for PCI Express (PIPE)* for timing diagrams. Active High

continued...

Port	Direction	Clock Domain	Description
`pipe_tx_detectrx_loopback[(N-1):0]`	In	`tx_coreclkin`	Instructs the PHY to start a receive detection operation. After power-up, asserting this signal starts a loopback operation. Refer to section 6.4 of the *Intel PHY Interface for PCI Express (PIPE)* for a timing diagram. Active High
`pipe_tx_compliance[(N-1):0]`	In	`tx_coreclkin`	Asserted for one cycle to set the running disparity to negative. Used when transmitting the compliance pattern. Refer to section 6.11 of the *Intel PHY Interface for PCI Express (PIPE) Architecture* for more information. Active High
`pipe_rx_polarity[(N-1):0]`	In	Asynchronous	When 1'b1, instructs the PHY layer to invert the polarity on the received data. Active High
`pipe_powerdown[(2*N-1):0]`	In	`tx_coreclkin`	Requests the PHY to change its power state to the specified state. The Power States are encoded as follows: 2'b00: P0 - Normal operation. 2'b01: P0s - Low recovery time, power saving state. 2'b10: P1 - Longer recovery time, lower power state . 2'b11: P2 - Lowest power state.
`pipe_tx_margin[(3*N-1):0]`	In	`tx_coreclkin`	Transmit V_{OD} margin selection. The PHY-MAC sets the value for this signal based on the value from the Link Control 2 Register. The following encodings are defined: 3'b000: Normal operating range 3'b001: Full swing: 800 - 1200 mV; Half swing: 400 - 700 mV. 3'b010:-3'b011: Reserved. 3'b100-3'b111: Full swing: 200 - 400mV; Half swing: 100 - 200 mV else reserved.
`pipe_tx_swing[(N-1):0]`	In	`tx_coreclkin`	Indicates whether the transceiver is using Full swing or Half swing voltage as defined by the `pipe_tx_margin`. 1'b0-Full swing. 1'b1-Half swing.
`pipe_tx_deemph[(N-1):0]`	In	Asynchronous	Transmit de-emphasis selection. In PCI Express Gen2 (5 Gbps) mode it selects the transmitter de-emphasis: 1'b0: –6 dB. 1'b1: –3.5 dB.
`pipe_g3_txdeemph[(18*N-1):0]`	In	Asynchronous	For Gen3, selects the transmitter de-emphasis. The 18 bits specify the following coefficients: [5:0]: C_{-1} [11:6]: C_0 [17:12]: C_{+1} Refer to the *Preset Mappings to TX De-emphasis* section for presets to TX de-emphasis mappings. In Gen3 capable designs, the TX de-emphasis for Gen2 datarate is always -6 dB. The TX de-emphasis for Gen1 datarate is always -3.5 dB.

continued...

Send Feedback

Port	Direction	Clock Domain	Description
			Refer to section 6.6 of *Intel PHY Interface for PCI Express (PIPE) Architecture* for more information.
`pipe_g3_rxpresethint[(3*N-1):0]`	In	Asynchronous	Provides the RX preset hint for the receiver. Set this to 3'b000 for PIPE Gen3 configurations.
`pipe_rx_eidleinfersel[(3*N-1):0]`	In	Asynchronous	When asserted high, the electrical idle state is inferred instead of being identified using analog circuitry to detect a device at the other end of the link. The following encodings are defined: 3'b0xx: Electrical Idle Inference not required in current LTSSM state. 3'b100: Absence of COM/SKP OS in 128 ms. 3'b101: Absence of TS1/TS2 OS in 1280 UI interval for Gen1 or Gen2. 3'b110: Absence of Electrical Idle Exit in 2000 UI interval for Gen1 and 16000 UI interval for Gen2. 3'b111: Absence of Electrical Idle exit in 128 ms window for Gen1. *Note:* Recommended to implement Receiver Electrical Idle Inference (EII) in FPGA fabric.
`pipe_rate[(2*N)-1:0]`	In	Asynchronous	The 2-bit encodings defined in the following list: 2'b00: Gen1 rate (2.5 Gbps) 2'b01: Gen2 rate (5.0 Gbps) 2'b10: Gen3 rate (8.0 Gbps)
`pipe_sw_done[1:0]`	In	N/A	Signal from the Master clock generation buffer, indicating that the rate switch has completed. Use this signal for bonding mode only. For non-bonded applications, this signal is internally connected to the local CGB.
`pipe_tx_data_valid[(N-1):0]`	In	`tx_coreclkin`	For Gen3, this signal is deasserted by the MAC to instruct the PHY to ignore `tx_parallel_data` for current clock cycle. A value of 1'b1 indicates the PHY should use the data. A value of 0 indicates the PHY should not use the data. Active High
`pipe_eq_eval`	In	`tx_coreclkin`	Assert high to start evaluation of the far end transmitter TX EQ settings. Active High
`pipe_eq_invalidreq`	In	`tx_coreclkin`	Assert high to indicate that the link partner TX EQ setting was out of range. Active High
`pipe_eq_inprogress`	In	`tx_coreclkin`	Assert high to indicate the MAC is in Phase 2 of Recovery.Equalization. Active High
PIPE Output to PHY - MAC Layer			
`rx_parallel_data[31:0], [15:0],or [7:0]`	Out	`rx_coreclkin`	The RX parallel data driven to the MAC.

continued...

Port	Direction	Clock Domain	Description
			For Gen1 this can be 8 or 16 bits. For Gen2 this is 16 bits only. For Gen3 this is 32 bits. Refer to Table 113 on page 201 for more details.
`rx_datak[3:0]`, `[1:0]`, or `[0]`	Out	`rx_coreclkin`	The data and control indicator. For Gen1 or Gen2, when 0, indicates that `rx_parallel_data` is data, when 1, indicates that `rx_parallel_data` is control. For Gen3, `Bit[0]` corresponds to `rx_parallel_data[7:0]`, `Bit[1]` corresponds to `rx_parallel_data[15:8]`, and so on. Refer to Table 113 on page 201 for more details.
`pipe_rx_sync_hdr[(2*N-1):0]`	Out	`rx_coreclkin`	For Gen3, indicates whether the 130-bit block being transmitted is a Data or Control Ordered Set Block. The following encodings are defined: 2'b10: Data block 2'b01: Control Ordered Set block This value is read when `pipe_rx_blk_start` = 4'b0001. Refer to *Section 4.2.2.1. Lane Level Encoding* in the *PCI Express Base Specification, Rev. 3.0* for a detailed explanation of data transmission and reception using 128b/130b encoding and decoding.
`pipe_rx_blk_start[(N-1):0]`	Out	`rx_coreclkin`	For Gen3, specifies the start block byte location for RX data in the 128-bit block data. Used when the interface between the PCS and PHY-MAC (FPGA Core) is 32 bits. Not used for Gen1 and Gen2 datarates. Active High
`pipe_rx_data_valid[(N-1):0]`	Out	`rx_coreclkin`	For Gen3, this signal is deasserted by the PHY to instruct the MAC to ignore `rx_parallel_data` for current clock cycle. A value of 1'b1 indicates the MAC should use the data. A value of 1'b0 indicates the MAC should not use the data. Active High
`pipe_rx_valid[(N-1):0]`	Out	`rx_coreclkin`	Asserted when RX data and control are valid.
`pipe_phy_status[(N-1):0]`	Out	`rx_coreclkin`	Signal used to communicate completion of several PHY requests. Active High
`pipe_rx_elecidle[(N-1):0]`	Out	Asynchronous	When asserted, the receiver has detected an electrical idle. Active High
`pipe_rx_status[(3*N-1):0]`	Out	`rx_coreclkin`	Signal encodes receive status and error codes for the receive data stream and receiver detection. The following encodings are defined: 3'b000 - Receive data OK 3'b001 - 1 SKP added 3'b010 - 1 SKP removed 3'b011 - Receiver detected

continued...

Port	Direction	Clock Domain	Description
			3'b100 - Either 8B/10B or 128b/130b decode error and (optionally) RX disparity error 3'b101 - Elastic buffer overflow 3'b110 - Elastic buffer underflow 3'b111 - Receive disparity error, not used if disparity error is reported using 3'b100.
pipe_sw[1:0]	Out	N/A	Signal to clock generation buffer indicating the rate switch request. Use this signal for bonding mode only. For non-bonded applications this signal is internally connected to the local CGB. Active High. Refer to Table 113 on page 201 for more details.

Table 113. Bit Mappings When the Simplified Interface Is Disabled

Signal Name	Gen 1 (TX Byte Serializer and RX Byte Deserializer disabled)	Gen1 (TX Byte Serializer and RX Byte Deserializer in X2 mode), Gen2 (TX Byte Serializer and RX Byte Deserializer in X2 mode)	Gen3
tx_parallel_data	tx_parallel_data[7:0]	tx_parallel_data[18:11,7:0]	tx_parallel_data[56:49,47:40,18:11,7:0]
tx_datak	tx_parallel_data[8]	tx_parallel_data[19,8]	tx_parallel_data[57,48,19,8]
pipe_tx_compliance[(N-1):0]	tx_parallel_data[9]	tx_parallel_data[9]	tx_parallel_data[9]
pipe_tx_elecidle[(N-1):0]	tx_parallel_data[10]	tx_parallel_data[10]	tx_parallel_data[10]
pipe_tx_detectrx_loopback[(N-1):0]	tx_parallel_data[20]	tx_parallel_data[20]	tx_parallel_data[20]
pipe_powerdown[(2*N-1):0]	tx_parallel_data[22:21]	tx_parallel_data[22:21]	tx_parallel_data[22:21]
pipe_tx_margin[(3*N-1):0]	tx_parallel_data[25:23]	tx_parallel_data[25:23]	tx_parallel_data[25:23]
pipe_tx_swing[(N-1):0]	tx_parallel_data[27]	tx_parallel_data[27]	tx_parallel_data[27]
pipe_tx_deemph[(N-1):0]	N/A	tx_parallel_data[26]	N/A
pipe_tx_sync_hdr[(2*N-1):0]	N/A	N/A	tx_parallel_data[29:28]
pipe_tx_blk_start[(N-1):0]	N/A	N/A	tx_parallel_data[30]
pipe_tx_data_valid[(N-1):0]	N/A	N/A	tx_parallel_data[31]
pipe_rate[(2*N)-1):0]	tx_parallel_data[33:32]	tx_parallel_data[33:32]	tx_parallel_data[33:32]

continued...

Signal Name	Gen 1 (TX Byte Serializer and RX Byte Deserializer disabled)	Gen1 (TX Byte Serializer and RX Byte Deserializer in X2 mode), Gen2 (TX Byte Serializer and RX Byte Deserializer in X2 mode)	Gen3
pipe_rx_polarity[(N-1):0]	tx_parallel_data[34]	tx_parallel_data[34]	tx_parallel_data[34]
pipe_eq_eval	N/A	N/A	tx_parallel_data[35]
pipe_eq_inprogress	N/A	N/A	tx_parallel_data[36]
pipe_eq_invalidreq	N/A	N/A	tx_parallel_data[37]
pipe_g3_rxpresethint[(3*N-1):0]	N/A	N/A	tx_parallel_data[60:58]
pipe_g3_txdeemph[(18*N-1):0]	N/A	N/A	tx_parallel_data[78:61]
rx_parallel_data	rx_parallel_data[7:0]	rx_parallel_data[22:15,7:0]	rx_parallel_data[62:55,47:40,22:15,7:0]
rx_datak	rx_parallel_data[8]	rx_parallel_data[23,8]	rx_parallel_data[63,48,23,8]
rx_syncstatus	rx_parallel_data[10]	rx_parallel_data[25,10]	rx_parallel_data[65,50,25,10]
pipe_phystatus[(N-1):0]	rx_parallel_data[32]	rx_parallel_data[32]	rx_parallel_data[32]
pipe_rx_valid[(N-1):0]	rx_parallel_data[33]	rx_parallel_data[33]	rx_parallel_data[33]
pipe_rx_status[(3*N-1):0]	rx_parallel_data[36:34]	rx_parallel_data[36:34]	rx_parallel_data[36:34]
pipe_rx_sync_hdr[(2*N-1):0]	N/A	N/A	rx_parallel_data[31:30]
pipe_rx_blk_start[(N-1):0]	N/A	N/A	rx_parallel_data[37]
pipe_rx_data_valid[(N-1):0]	N/A	N/A	rx_parallel_data[38]

Related Information

- Using the Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY Intel Stratix 10 FPGA IP Core on page 106
- Preset Mappings to TX De-emphasis on page 206
- Intel PHY Interface for PCI Express (PIPE) Architecture

Send Feedback

2.5.1.9. fPLL Ports for PIPE

Table 114. fPLL Ports for PIPE

Port	Direction	Clock Domain	Description
`pll_reflck0`	In	N/A	Reference clock input port 0. There are five reference clock input ports. The number of reference clock ports available depends on the Number of PLL reference clocks parameter.
`tx_serial_clk`	Out	N/A	High speed serial clock output port for GX channels. Represents the x1 clock network. For Gen1x1, Gen2x1, connect the output from this port to the `tx_serial_clk` input of the native PHY IP. For Gen1x2, x4, x8, x16 use the `tx_bonding_clocks` output port to connect to the Native PHY IP. For Gen2x2, x4, x8, x16 use the `tx_bonding_clocks` output port to connect to the Native PHY IP. For Gen3x1, connect the output from this port to one of the two `tx_serial_clk` input ports on the native PHY IP. For Gen3x2, x4, x8, x16 connect the output from this port to the Auxiliary Master CGB clock input port of the ATX PLL IP.
`pll_locked`	Out	Asynchronous	Active high status signal which indicates if PLL is locked.
`pll_pcie_clk`	Out	N/A	This is the hclk required for PIPE interface. For Gen1x1, x2, x4, x8, x16 use this port to drive the `hclk` for the PIPE interface. For Gen2x1, x2, x4, x8, x16 use this port to drive the `hclk` for the PIPE interface. For Gen3x1, x2, x4, x8, x16 use the `pll_pcie_clk` from fPLL (configured as Gen1/Gen2) as the hclk for the PIPE interface.
`pll_cal_busy`	Out	Asynchronous	Status signal which is asserted high when PLL calibration is in progress. If this port is not enabled in the Transceiver PHY Reset Controller IP, then perform logical OR with this signal and the `tx_cal_busy` output signal from the Native PHY IP core to input the `tx_cal_busy` on the Transceiver PHY Reset Controller.
`mcgb_aux_clk0`	In	N/A	Used for Gen3 to switch between fPLL/ATX PLL during link speed negotiation. For Gen3x2, x4, x8, x16, use the `mcgb_aux_clk` input port on the ATX PLL.
`tx_bonding_clocks[5:0]`	Out	N/A	Optional 6-bit bus which carries the low speed parallel clock outputs from the Master CGB. It is used for channel bonding, and represents the x6/xN clock network. For Gen1x1, this port is disabled. For Gen1x2, x4, x8, x16, connect the output from this port to the `tx_bonding_clocks` input on the Native PHY IP. For Gen2x1, this port is disabled.

continued...

Port	Direction	Clock Domain	Description
			For Gen2x2, x4, x8, x16 connect the output from this port to the `tx_bonding_clocks` input on the Native PHY IP. For Gen3x1, this port is disabled. For Gen3x2, x4, x8, x16, use the `tx_bonding_clocks` output from the ATX PLL to connect to the `tx_bonding_clocks` input of the Native PHY IP.
`pcie_sw[1:0]`	In	Asynchronous	2-bit rate switch control input used for PCIe protocol implementation. For Gen1, this port is N/A For Gen2x2, x4, x8, x16, connect the `pipe_sw` output from the Native PHY IP to this port. For Gen3x2, x4, x8, x16, connect the `pipe_sw` output from the Native PHY IP to this port. For Gen3x2, x4, x8, x16, this port is not used. You must use the `pipe_sw` from the Native PHY IP to drive the `pcie_sw` input port on the ATX PLL.
`pcie_sw_done[1:0]`	Out	Asynchronous	2-bit rate switch status output used for PCIe protocol implementation. For Gen1, this port is N/A. For Gen2x2, x4, x8, x16, connect the `pcie_sw_done` output from ATX PLL to the `pipe_sw_done` input of the Native PHY IP. For Gen3x2, x4, x8, x16 connect the `pcie_sw_done` output from ATX PLL to the `pipe_sw_done` input of the Native PHY IP.

2.5.1.10. ATX PLL Ports for PIPE

Table 115. ATX PLL Ports for PIPE

Port	Direction	Clock Domain	Description
`pll_reflck0`	In	N/A	Reference clock input port 0. There are five reference clock input ports. The number of reference clock ports available depends on the Number of PLL reference clocks parameter.
`tx_serial_clk`	Out	N/A	High speed serial clock output port for GX channels. Represents the x1 clock network. For Gen1x1, Gen2x1, connect the output from this port to the tx_serial_clk input of the Native PHY IP. For Gen1x2, x4, x8, x16, use the tx_bonding_clocks output port to connect to the Native PHY IP. For Gen2x2, x4, x8, x16, use the tx_bonding_clocks output port to connect to the Native PHY IP. For Gen3x1, connect the output from this port to one of the two tx_serial_clk input ports on the Native PHY IP.

continued...

Port	Direction	Clock Domain	Description
			For Gen3x2, x4, x8, x16, this port is not used. Use the tx_serial_clk output from the fPLL to drive the Auxiliary Master CGB clock input port of the ATX PLL.
`pll_locked`	Out	Asynchronous	Active high status signal which indicates if PLL is locked.
`pll_pcie_clk`	Out	N/A	This is the hclk required for PIPE interface. For Gen1x1,x2,x4,x8, x16, use this port to drive the hclk for the PIPE interface. For Gen2x1,x2,x4,x8, x16, use this port to drive the hclk for the PIPE interface. For Gen3x1,x2,x4,x8, x16, this port is not used. Use the `pll_pcie_clk` from fPLL (configured as Gen1/Gen2) as the hclk for the PIPE interface.
`pll_cal_busy`	Out	Asynchronous	Status signal which is asserted high when PLL calibration is in progress. If this port is not enabled in the Transceiver PHY Reset Controller, then perform logical OR with this signal and the `tx_cal_busy` output signal from the Native PHY IP core to input the `tx_cal_busy` on the Transceiver PHY Reset Controller.
`mcgb_aux_clk0`	In	N/A	Used for Gen3 to switch. between fPLL/ATX PLL during link speed negotiation. For Gen3x2,x4,x8, x16, use the `tx_serial_clk output` port from fPLL (configured for Gen1/Gen2) to drive the `mcgb_aux_clk` input port on the ATX PLL.
`tx_bonding_clocks[5:0]`	Out	N/A	Optional 6-bit bus which carries the low speed parallel clock outputs from the Master CGB. Used for channel bonding, and represents the x6/xN clock network. For Gen1x1, this port is disabled. For Gen1x2,x4,x8, x16, connect the output from this port to the `tx_bonding_clocks` input on the Native PHY IP. For Gen2x1, this port is disabled For Gen2x2,x4,x8, x16, connect the output from this port to `tx_bonding_clocks` input on the Native PHY IP. For Gen3x1, this port is disabled. For Gen3x2,x4,x8, x16, use the `tx_bonding_clocks` output from the ATX PLL to connect to the `tx_bonding_clocks` input of the Native PHY IP.
`pcie_sw[1:0]`	In	Asynchronous	2-bit rate switch control input used for PCIe protocol implementation. For Gen1, this port is N/A. For Gen 2x2,x4,x8, x16, connect the `pipe_sw` output from the Native PHY IP to this port.

continued...

Port	Direction	Clock Domain	Description
			For Gen3x2,x4,x8, x16, use the `pipe_sw` output from the Native PHY IP to drive this port.
`pcie_sw_done[1:0]`	Out	Asynchronous	2-bit rate switch status output used for PCIe protocol implementation. For Gen1, this port is N/A. For Gen2x2, x4, x8, x16, connect the `pcie_sw_done` output from ATX PLL to `pipe_sw_done` input of the Native PHY IP. For Gen3x2, x4, x8, x16, `pcie_sw_done` output from ATX PLL to `pipe_sw_done` input of the Native PHY IP.

2.5.1.11. Preset Mappings to TX De-emphasis

Table 116. Intel Stratix 10 Preset Mappings to TX De-emphasis

Preset	C_{+1}	C_0	C_{-1}	VOD (38)	First Pre-tap	First Post-tap
P0	001111	101101	000000	31	—	-15
P1	001010	110010	000000	31	—	-10
P2	001100	110000	000000	31	—	-12
P3	001000	110100	000000	31	—	-8
P4	000000	111100	000000	31	—	—
P5	000000	110110	000110	31	-6	—
P6	000000	110100	001000	31	-8	—
P7	001100	101010	000110	31	-6	-12
P8	001000	101100	001000	31	-8	-8
P9	000000	110010	001010	31	-10	—
P10	010110	100110	000000	31	—	-20

Use the `pipe_g3_txdeemph[17:0]` port to select the transmitter de-emphasis. The 18 bits specify the following coefficients:

- [5:0]: C_{-1}
- [11:6]: C_0
- [17:12]: C_{+1}

The PCIe Gen3 full swing (FS) value is 60, and the PCIe Gen3 low frequency (LF) value is 20.

Intel recommends transmitting Preset P8 coefficients for the Intel Stratix 10 receiver to recover data successfully.

(38) The VOD setting is fixed at 31 (for example, FS/2 + 1 = 60/2 + 1) for both full-swing mode operation and reduced-swing mode operation. FS denotes the Full Swing value.

2.5.1.12. How to Place Channels for PIPE Configurations

The following restrictions apply when placing channels for PIPE configurations:

- The channels must be contiguous because PCIe requires bonded channels.
- The master CGB is the only way to access the x6 lines, and you must use it in bonded designs (x2, x4, x8 and x16). You cannot use the local CGB to route clock signals to slave channels because the local CGB does not have access to x6 lines.
- The logical PCS master channel in the bonded configurations must align with physical channel 1 or 4 of the bank. Refer to the *Master Channel in Bonded Configurations* section for more details.

For ATX PLL placement restrictions, refer to the "Transmit PLL Recommendations Based on Data Rates" section of the *PLLs and Clock Networks* chapter.

Related Information

- PLLs and Clock Networks on page 249
- Master Channel in Bonded Configurations on page 207

2.5.1.12.1. Master Channel in Bonded Configurations

For PCIe, TX PMA and PCS bonding must be enabled. There is no need to specify the PMA Master Channel because of the separate Master CGB in the hardware. However, you must specify the PCS Master Channel through the Native PHY IP Parameter Editor. You can do either one of the following:

- Specify the PCS Master Channel through the **Logical PCS Master Channel** parameter. When you choose any one of the data channels (part of the bonded group) as the logical PCS Master Channel, you must ensure that the logical PCS master channel aligns with Physical Channel 1 or 4 in a given transceiver bank.
- Select **AUTO** in the **Logical PCS Master Channel** parameter. This chooses the logical PCS master channel according to Table 117 on page 207. The transceiver Native PHY IP automatically selects the center channel to be the master PCS channel. This minimizes the total starting delay for the bonded group.When using the **AUTO** selection, the fitter issues an error when the chosen Logical PCS master channel does not align with Physical Channel 1 or 4 of the bank.

Note: The auto speed negotiation (ASN) block and Master CGB connectivity is only available in the hardware of channels 1 and 4 of a given transceiver bank.

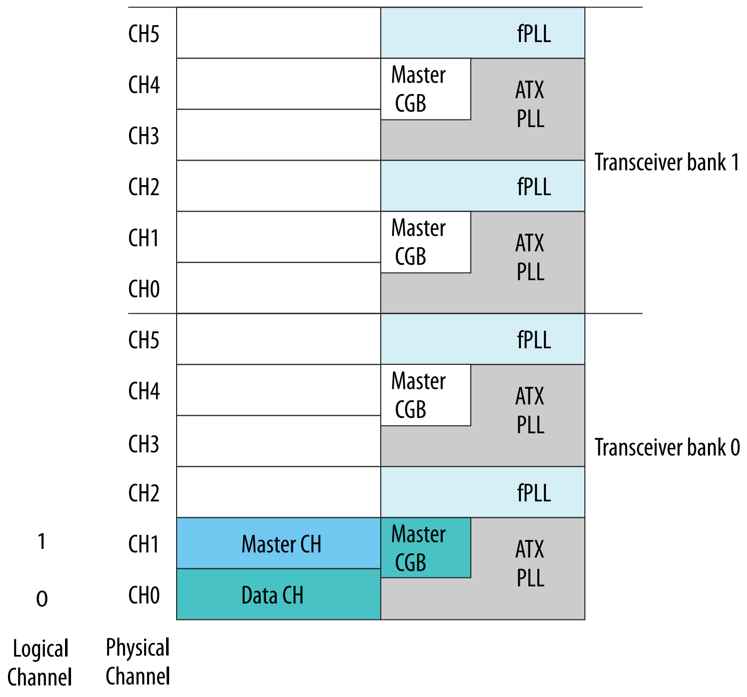
Table 117. Logical PCS Master Channel for PIPE Configuration when the AUTO Selection is Enabled

PIPE Configuration	Logical PCS Master Channel # (default)
x1	0
x2	1 [39]
x4	2 [39]
x8	4 [39]
x16	8 [39]

[39] Ensure that the Logical PCS Master Channel aligns with Physical Channel 1 or 4 in a given transceiver bank.

The following figures show the default configurations:

Figure 111. x2 Configuration



Note: The physical channel 1 aligns with logical channel 1 of transceiver bank 0.

Figure 112. x4 Configuration

The figure below shows a way of placing 4 bonded channels. In this case, the logical PCS Master Channel number 2 must be specified as Physical channel 4 of bank 0.

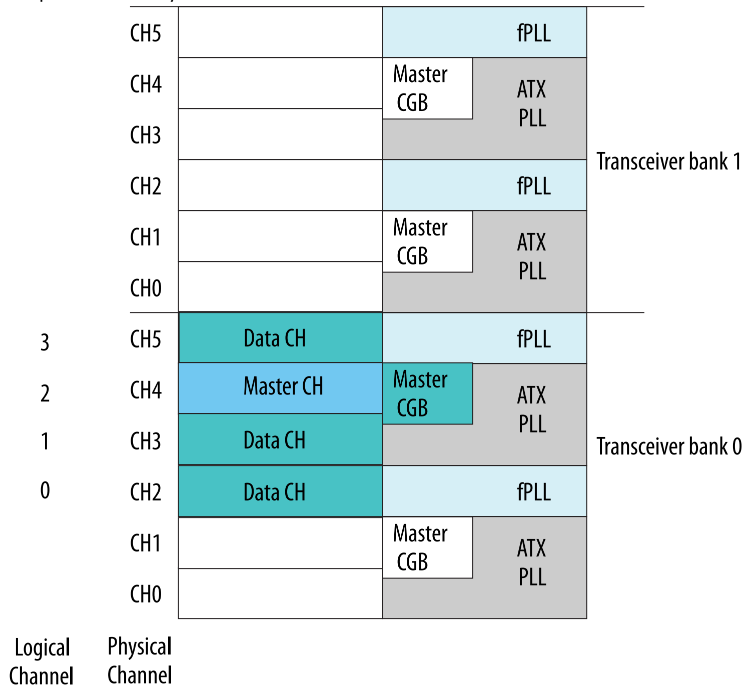


Figure 113. x8 Configuration

For x8 configurations, Intel recommends you choose a master channel that is no more than four channels away from the farthest slave channel.

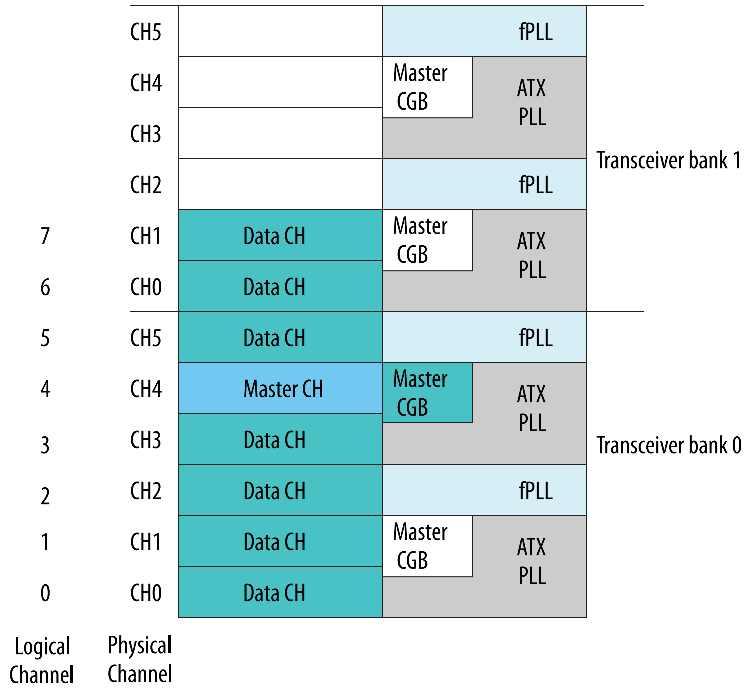
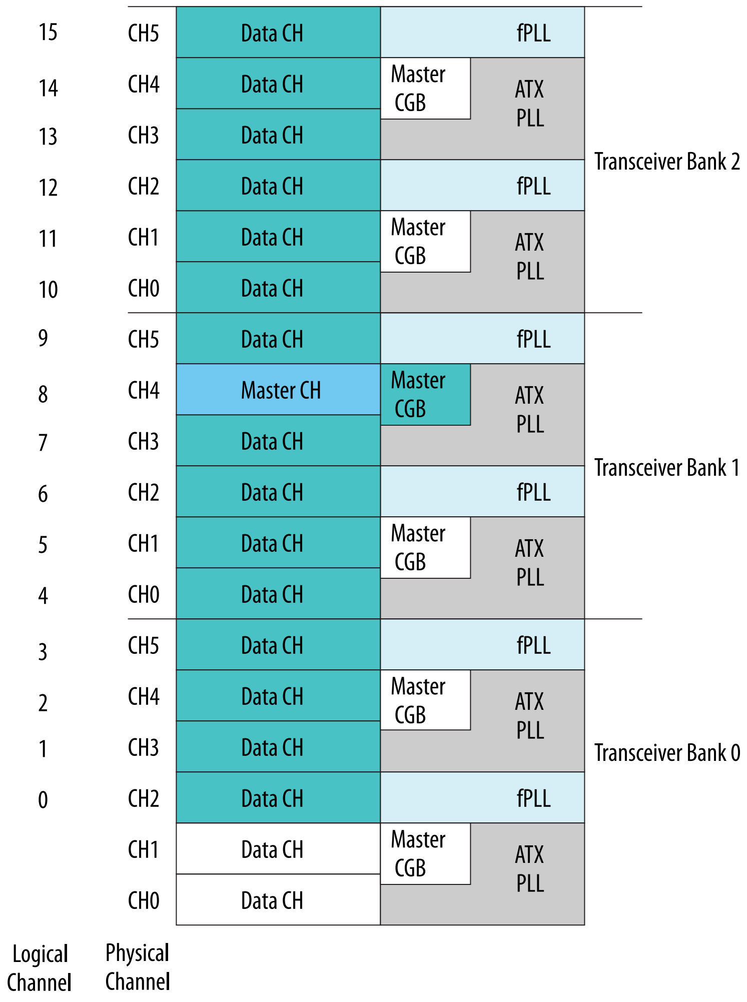


Figure 114. x16 Configuration

For x16 configurations, Intel recommends you choose a master channel that is a maximum of eight channels away from the farthest slave channel.

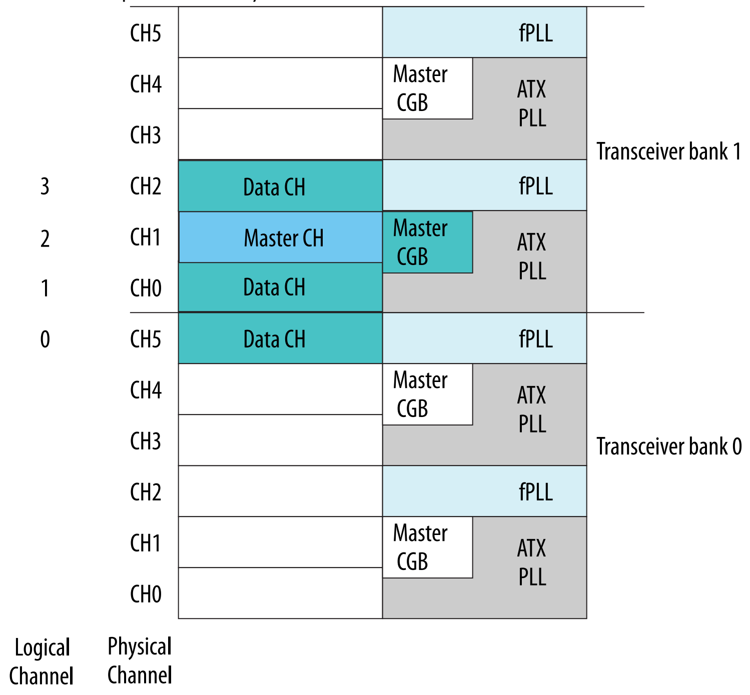


Note: The physical channel 2 aligns with logical channel 0. The logical PCS master channel 8 is specified as Physical channel 4 of bank 1.

Send Feedback

Figure 115. x4 Alternate Configuration

The figure below shows an alternate way of placing 4 bonded channels. In this case, the logical PCS Master Channel number 2 must be specified as Physical channel 1 of bank 1.



2.5.1.13. Link Equalization for Gen3

Gen3 mode requires TX and RX link equalization because of the datarate, the channel characteristics, receiver design, and process variations. The link equalization process allows the Endpoint and Root Port to adjust the TX and RX setup of each lane to improve signal quality. This process results in Gen3 links with a receiver Bit Error Rate (BER) that is less than 10^{-12}.

For detailed information about the four-stage link equalization procedure for 8.0 GT/s datarate, refer to Section 4.2.3 in the *PCI Express Base Specification, Rev 3.0*. A new LTSSM state, Recovery.Equalization with Phases 0–3, reflects progress through Gen3 equalization. Phases 2 and 3 of link equalization are optional. Each link must progress through all four phases, even if no adjustments occur. If you skip Phases 2 and 3, you speed up link training at the expense of link BER optimization.

Phase 0

Phase 0 includes the following steps:

1. The upstream component enters Phase 0 of equalization during Recovery.Rcvrconfig by sending EQ TS2 training sets with starting presets for the downstream component. EQ TS2 training sets may be sent at 2.5 GT/s or 5 GT/s.
2. The downstream component enters Phase 0 of equalization after exiting Recovery.Speed at 8 GT/s. It receives the starting presets from the training sequences and applies them to its transmitter. At this time, the upstream component has entered Phase 1 and is operating at 8 GT/s.
3. To move to Phase 1, the receiver must have a BER < 10^{-4}. The receiver should be able to decode enough consecutive training sequences.
4. To move to Equalization Phase 1, the downstream component must detect training sets with Equalization Control (EC) bits set to 2’b01.

Phase 1

During Phase 1 of the equalization process, the link partners exchange Full Swing (FS) and Low Frequency (LF) information. These values represent the upper and lower bounds for the TX coefficients. The receiver uses this information to calculate and request the next set of transmitter coefficients.

1. The upstream component moves to EQ Phase 2 when training sets with EC bits set to 2'b01 are captured on all lanes. It also sends EC=2’b10, starting pre-cursor, main cursor, and post-cursor coefficients.
2. The downstream component moves to EQ Phase 2 after detecting these new training sets.

Use the **pipe_g3_txdeemph[17:0]** port to select the transmitter de-emphasis. The 18 bits specify the following coefficients:

- [5:0]: C_{-1}
- [11:6]: C_0
- [17:12]: C_{+1}

Refer to *Preset Mappings to TX De-emphasis* for the mapping between presets and TX de-emphasis.

Phase 2 (Optional)

During Phase 2, the Endpoint tunes the TX coefficients of the Root Port. The TS1 Use Preset bit determines whether the Endpoint uses presets for coarse resolution or coefficients for fine resolution.

Note: You cannot perform Phase 2 tuning, when you are using the PHY IP Core for PCI Express (PIPE) as an Endpoint. The PIPE interface does not provide any measurement metric to the Root Port to guide coefficient preset decision making. The Root Port should reflect the existing coefficients and move to the next phase. The default Full Swing (FS) value advertised by the Intel device is 60 and Low Frequency (LF) is 20.

If you are using the PHY IP Core for PCI Express (PIPE) as the Root Port, the Endpoint can tune the Root Port TX coefficients.

The tuning sequence typically includes the following steps:

1. The Endpoint receives the starting presets from the Phase 2 training sets sent by the Root Port.
2. The circuitry in the Endpoint receiver determines the BER. It calculates the next set of transmitter coefficients using FS and LF. It also embeds this information in the Training Sets for the Link Partner to apply to its transmitter.

   The Root Port decodes these coefficients and presets, performs legality checks for the three transmitter coefficient rules and applies the settings to its transmitter and also sends them in the Training Sets. The three rules for transmitter coefficients are:

   a. $|C_{-1}| <=$ Floor (FS/4)

   b. $|C_{-1}|+C_0+|C_{+1}| =$ FS

   c. $C_0-|C_{-1}|-|C_{+1}| >=$ LF

   Where: *C_0 is the main cursor (boost)*, *C_{-1} is the pre-cursor (pre-shoot)*, and *C_{+1} is the post-cursor (de-emphasis)*.
3. This process is repeated until the downstream component's receiver achieves a BER of $< 10^{-12}$

Phase 3 (Optional)

During this phase, the Root Port tunes the Endpoint's transmitter. This process is analogous to Phase 2 but operates in the opposite direction.

You cannot perform Phase 3 tuning, when you are using the PHY IP Core for PCI Express (PIPE) as a Root Port.

After Phase 3 tuning is complete, the Root Port moves to Recovery.RcvrLock, sending EC=2'b00, and the final coefficients or preset agreed upon in Phase 2. The Endpoint moves to Recovery.RcvrLock using the final coefficients or preset agreed upon in Phase 3.

Recommendations for Tuning Link

Intel recommends transmitting Preset P8 coefficients for the Intel Stratix 10 receiver to recover data successfully.

Related Information

- Preset Mappings to TX De-emphasis on page 206
- PCI Express Base Specification

2.5.1.14. Timing Closure Recommendations

When using the Native PHY IP core to implement PCIe PIPE, observe these timing closure recommendations.

When using PCIe PIPE in bonded configurations (x2, x4, x8, x16), use the `pclk` (`tx_clkout` from the TX bonding master channel) to drive all the `tx_coreclkin` and `rx_coreclkin` clock inputs. The Timing Analyzer may report timing violations if you use the `tx_clkout` output of each channel to drive the corresponding `tx_coreclkin` and `rx_coreclkin` inputs of the Native PHY IP core.

Note: The Native PHY IP core creates all the timing constraints between each channel and the PCIe speed (Gen1, Gen2, Gen3, as applicable) for the TX and RX output clock pins (`tx_clkout`, `tx_clkout_2`, `rx_clkout`, `rx_clkout_2`). Refer to the sdc file generated by the Native PHY IP core in `<Project folder / Native PHY IP Instance / altera_xcvr_native_s10_htile_version / synth / pipe_gen3_x8_native_ip_altera_xcvr_native_s10_htile_inst.sdc>` for details about how the Native PHY IP core clocks are constrained.

Design Example

Select **Generate Example Design** to create a PCIe PIPE design example that you can simulate and download to hardware. The Intel Quartus Prime project, settings files, and the IP files are available in the following location in the project folder:

```
<Project Folder> / <…example_design>
```

Refer to the "Design Example” table in the *Native PHY IP Core Parameter Settings for PIPE* section for more details on the parameters to choose for PCIe PIPE configurations.

Related Information

Native PHY IP Core Parameter Settings for PIPE on page 185

2.5.2. Interlaken

The Interlaken interface is supported with 1 to 24 lanes running at datarates up to 17.4 Gbps per lane on Intel Stratix 10 devices. Interlaken is implemented using the Enhanced PCS.

Intel Stratix 10 devices provide three preset variations for Interlaken in the Intel Stratix 10 Transceiver Native PHY IP Parameter Editor:

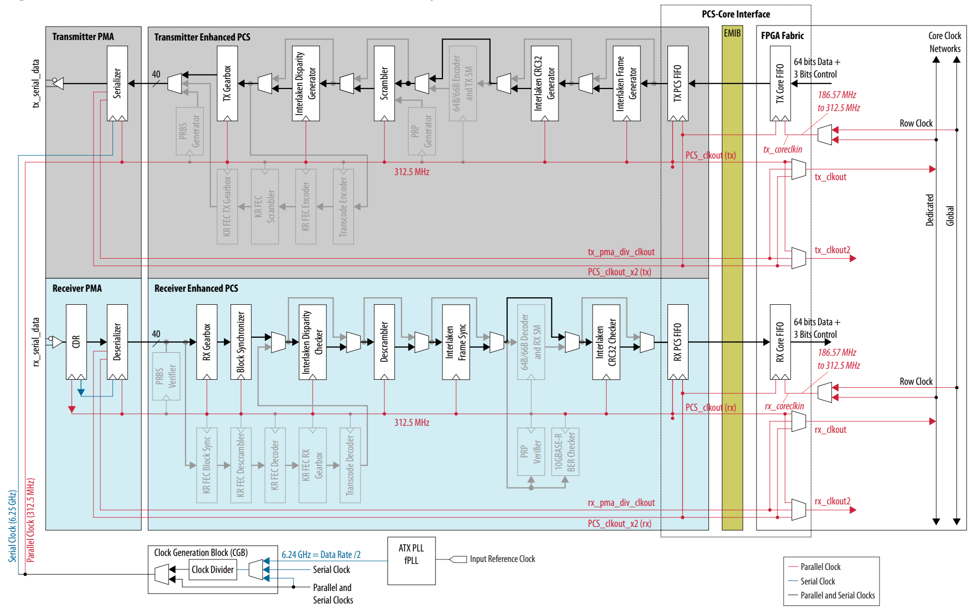
- Interlaken 10x12.5 Gbps
- Interlaken 1x6.25 Gbps
- Interlaken 6x10.3 Gbps

Depending on the line rate, the Enhanced PCS can use a PMA to PCS interface width of 32, 40, or 64 bits.

The Native PHY IP core does not support double rate transfer option when configured in Interlaken.

Figure 116. Transceiver Channel Datapath and Clocking for Interlaken

This figure assumes the serial datarate is 12.5 Gbps and the PMA width is 40 bits.



Related Information

- Interlaken Protocol Definition v1.2
- Interlaken Look-Aside Protocol Definition, v1.1

2.5.2.1. Interlaken Configuration Clocking and Bonding

The Interlaken PHY layer solution is scalable and has flexible datarates. You can implement a single lane link or bond up to 24 lanes together. You can choose a lane datarate up to 17.4 Gbps for GX devices. You can also choose between different reference clock frequencies, depending on the PLL used to clock the transceiver. Refer to the *Intel Stratix 10 Device Datasheet* for the minimum and maximum datarates that Intel Stratix 10 transceivers can support at different speed grades.

You can use an ATX PLL or fPLL to provide the clock for the transmit channel. An ATX PLL has better jitter performance compared to an fPLL. You can use a channel PLL as a CMU PLL to clock only the non-bonded Interlaken transmit channels. However, when the channel PLL is used as a CMU PLL, the channel can only be used as a transmitter channel.

For the multi-lane Interlaken interface, TX channels are usually bonded together to minimize the transmit skew between all bonded channels. Currently, the x24 bonding scheme is available to support a multi-lane Interlaken implementation. If the system tolerates higher channel-to-channel skew, you can choose to not bond the TX channels.

To implement bonded multi-channel Interlaken, all channels must be placed contiguously. The channels may all be placed in one bank (if not greater than six lanes) or they may span several banks.

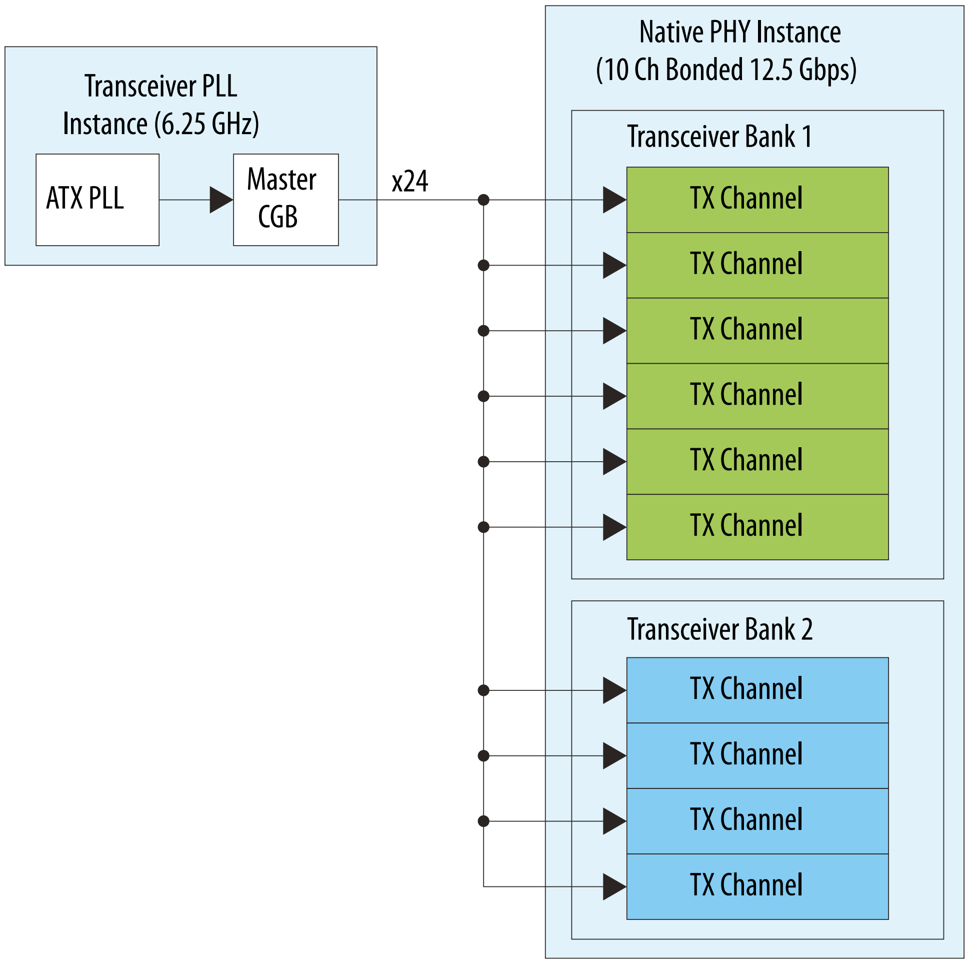
Related Information

Intel Stratix 10 Device Datasheet

2.5.2.1.1. x24 Clock Bonding Scenario

The following figure shows a x24 bonding example supporting 10 lanes. Each lane is running at 12.5 Gbps. The first six TX channels reside in one transceiver bank and the other four TX channels reside in the adjacent transceiver bank. The ATX PLL provides the serial clock to the master CGB. The CGB then provides parallel and serial clocks to all of the TX channels inside the same bank and other banks through the x24 clock network.

Figure 117. 10X12.5 Gbps x24 Bonding



2.5.2.1.2. TX Multi-Lane Bonding and RX Multi-Lane Deskew Alignment State Machine

The Interlaken configuration sets the Enhanced PCS TX and RX FIFOs in Interlaken elastic buffer mode. In this mode of operation, TX and RX FIFO control and status port signals are provided to the FPGA fabric. Connect these signals to the MAC layer as required by the protocol. Based on these FIFO status and control signals, you can implement the multi-lane deskew alignment state machine in the FPGA fabric to control the transceiver RX FIFO block.

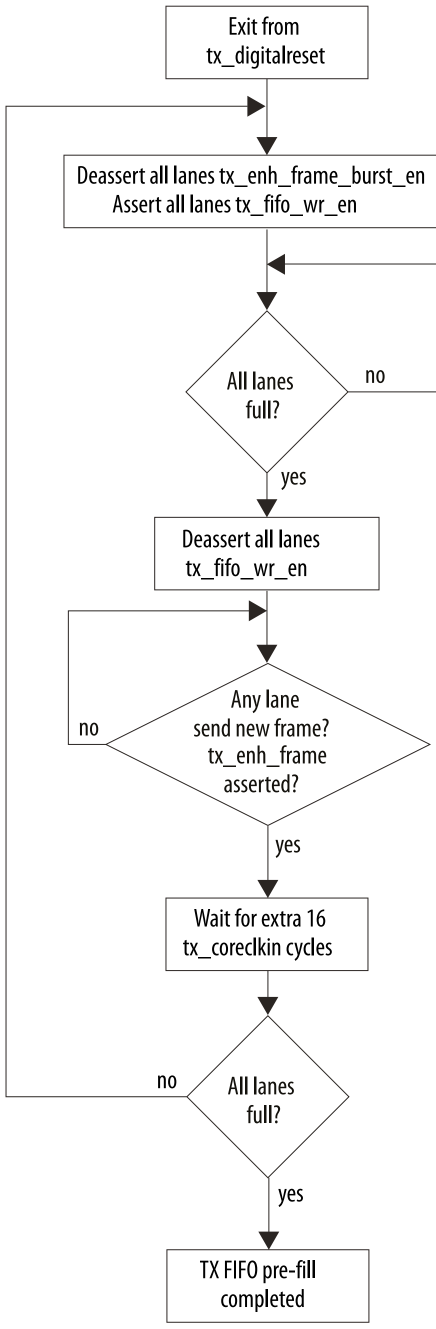
Note: You must also implement the soft bonding logic to control the transceiver TX FIFO block.

TX Soft Bonding Flow

The MAC layer logic and TX soft bonding logic control the writing of the Interlaken word to the TX FIFO with `tx_fifo_wr_en input`by monitoring the TX FIFO flags `tx_fifo_full`, `tx_fifo_pfull`, `tx_fifo_empty`, `tx_fifo_pempty` . On the TX FIFO read side, a read enable is controlled by the frame generator. If `tx_enh_frame_burst_en` is asserted high, the frame generator reads data from the TX FIFO.

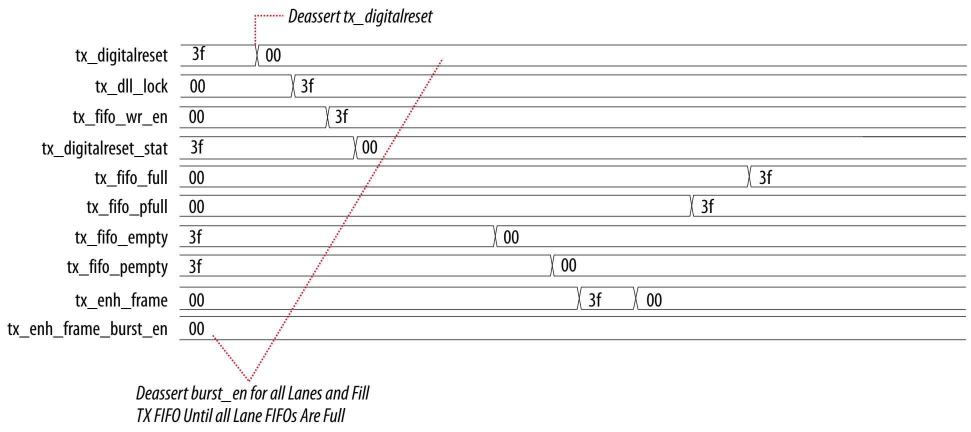
A TX FIFO pre-fill stage must be implemented to perform the TX channel soft bonding. The following figure shows the state of the pre-fill process.

Figure 118. TX Soft Bonding Flow


Exit from tx_digitalreset
Deassert all lanes tx_enh_frame_burst_en
Assert all lanes tx_fifo_wr_en
All lanes full?
no
yes
Deassert all lanes tx_fifo_wr_en
Any lane send new frame? tx_enh_frame asserted?
no
yes
Wait for extra 16 tx_coreclkin cycles
All lanes full?
no
yes
TX FIFO pre-fill completed

The following figure shows that after deasserting `tx_digitalreset`, TX soft bonding logic starts filling the TX FIFO until all lanes are full.

Figure 119. TX FIFO Pre-fill (6-lane Interface)



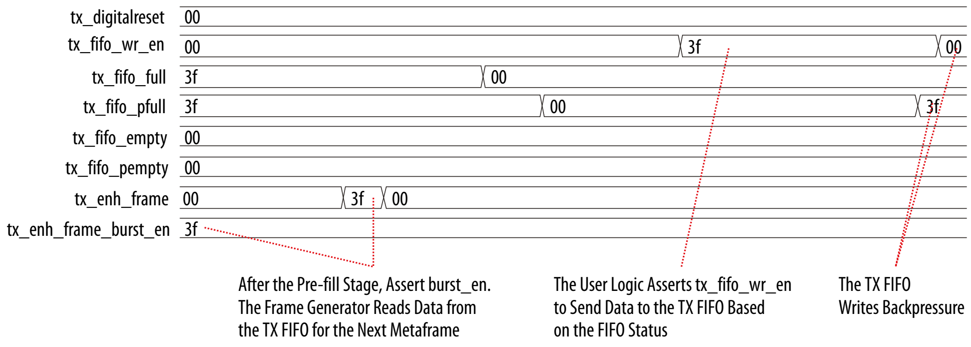
Notes:
1. tx_fifo_wr_en should be asserted 250ns after tx_dll_lock are asserted on all channels if multiple Interlaken channels are instantiated. tx_fifo_wr_en should be asserted as soon as tx_dll_lock is asserted if only a single Interlaken channel is instantiated.

After the TX FIFO pre-fill stage completes, the transmit lanes synchronize and the MAC layer begins to send valid data to the transceiver's TX FIFO. You must never allow the TX FIFO to overflow or underflow. If it does, you must reset the transceiver and repeat the TX FIFO pre-fill stage.

For a single lane Interlaken implementation, TX FIFO soft bonding is not required.

The following figure shows the MAC layer sending valid data to the Native PHY after the pre-fill stage. `tx_enh_frame_burst_en` is asserted, allowing the frame generator to read data from the TX FIFO. The TX MAC layer can now control `tx_fifo_wr_en` and write data to the TX FIFO based on the FIFO status signals.

Figure 120. MAC Sending Valid Data (6-lane Interface)

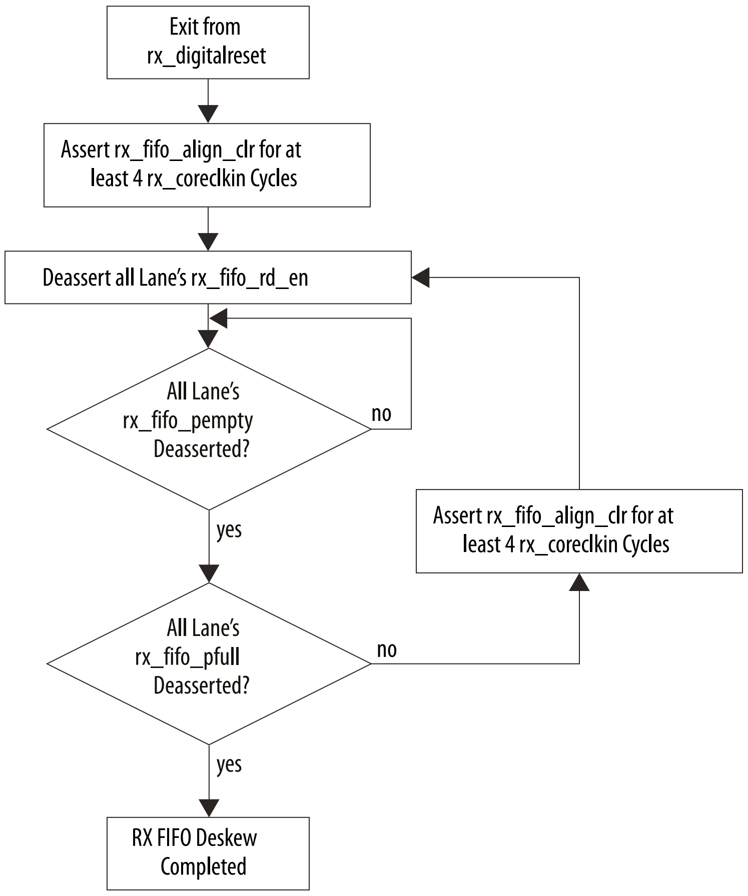

tx_digitalreset 00
tx_fifo_wr_en 00 3f 00
tx_fifo_full 3f 00
tx_fifo_pfull 3f 00 3f
tx_fifo_empty 00
tx_fifo_pempty 00
tx_enh_frame 00 3f 00
tx_enh_frame_burst_en 3f
After the Pre-fill Stage, Assert burst_en. The Frame Generator Reads Data from the TX FIFO for the Next Metaframe
The User Logic Asserts tx_fifo_wr_en to Send Data to the TX FIFO Based on the FIFO Status
The TX FIFO Writes Backpressure

RX Multi-lane FIFO Deskew State Machine

Add deskew logic at the receiver side to eliminate the lane-to-lane skew created at the transmitter of the link partner, PCB, medium, and local receiver PMA.

Implement a multi-lane alignment deskew state machine to control the RX FIFO operation based on available RX FIFO status flags and control signals.

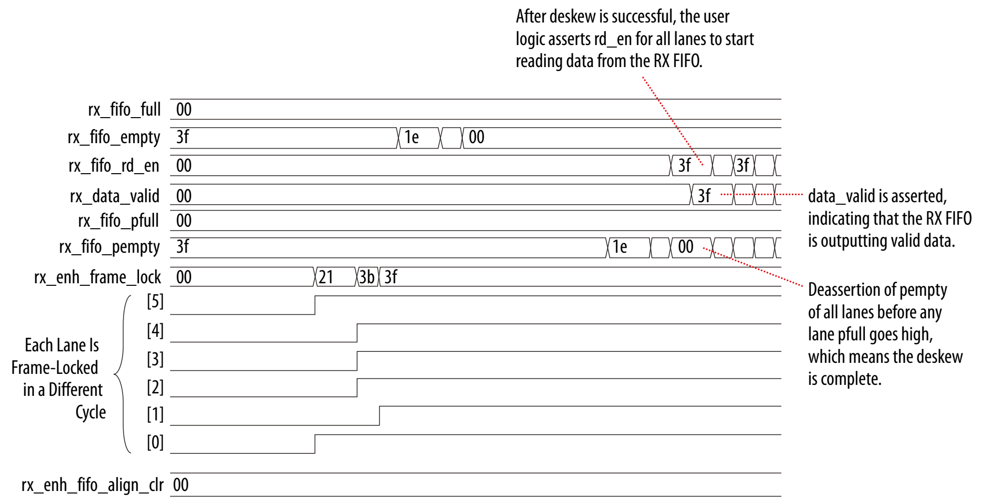
Figure 121. State Flow of the RX FIFO Deskew


Exit from rx_digitalreset
Assert rx_fifo_align_clr for at least 4 rx_coreclkin Cycles
Deassert all Lane's rx_fifo_rd_en
All Lane's rx_fifo_pempty Deasserted?
no
yes
Assert rx_fifo_align_clr for at least 4 rx_coreclkin Cycles
All Lane's rx_fifo_pfull Deasserted?
no
yes
RX FIFO Deskew Completed

You must assert `rx_fifo_align_clr` at least four `rx_coreclkin` cycles to clear the RX FIFO upon exit from `rx_digitalreset`. Each lane's `rx_fifo_rd_en` should remain deasserted before the RX FIFO deskew is completed. After frame lock is achieved (indicated by the assertion of `rx_enh_frame_lock`; this signal is not shown in the above state flow), data is written into the RX FIFO after the first alignment word (SYNC word) is found on that channel. Accordingly, the RX FIFO partially empty flag (`rx_fifo_pempty`) of that channel is asserted. The state machine monitors the `rx_fifo_pempty` and `rx_fifo_pfull` signals of all channels. If the `rx_fifo_pempty` signals from all channels deassert before any channels `rx_fifo_pfull` assert, which implies the SYNC word has been found on all lanes of the link, the MAC layer can start reading from all the RX FIFO by asserting `rx_fifo_rd_en` simultaneously. Otherwise, if the `rx_fifo_pfull` signal of any channel asserts high before the `rx_fifo_pempty` signals deassertion on all channels, the state machine needs to flush the RX FIFO by asserting `rx_fifo_align_clr` high for 4 cycles and repeating the soft deskew process.

The following figure shows one RX deskew scenario. In this scenario, all of the RX FIFO partially empty lanes are deasserted while the pfull lanes are still deasserted. This indicates the deskew is successful and the FPGA fabric starts reading data from the RX FIFO.

Figure 122. RX FIFO Deskew


After deskew is successful, the user logic asserts rd_en for all lanes to start reading data from the RX FIFO.
rx_fifo_full
rx_fifo_empty
rx_fifo_rd_en
rx_data_valid
rx_fifo_pfull
rx_fifo_pempty
rx_enh_frame_lock
Each Lane Is Frame-Locked in a Different Cycle
[5]
[4]
[3]
[2]
[1]
[0]
rx_enh_fifo_align_clr
00
3f
1e
21
3b
data_valid is asserted, indicating that the RX FIFO is outputting valid data.
Deassertion of pempty of all lanes before any lane pfull goes high, which means the deskew is complete.

2.5.3. Ethernet

The Ethernet standard comprises many different PHY standards with variations in signal transmission medium and datarates. The Native PHY IP core supports the 1G and 10G Ethernet datarates with different presets targeting different Ethernet applications as listed in the following table.

Datarate	Transceiver Configuration Rule / Preset
1G	• Gigabit Ethernet • Gigabit Ethernet 1588
10G	• 10GBASE-R • 10GBASE-R 1588 • 10GBASE-R with KR FEC • 10GBASE-R Low Latency

2.5.3.1. 10GBASE-R, 10GBASE-R with IEEE 1588v2, and 10GBASE-R with KR FEC Variants

10GBASE-R PHY is the Ethernet-specific physical layer running at a 10.3125-Gbps datarate as defined in Clause 49 of the IEEE 802.3-2008 specification. Intel Stratix 10 transceivers can implement 10GBASE-R variants like 10GBASE-R with IEEE 1588v2, and with KR forward error correction (FEC).

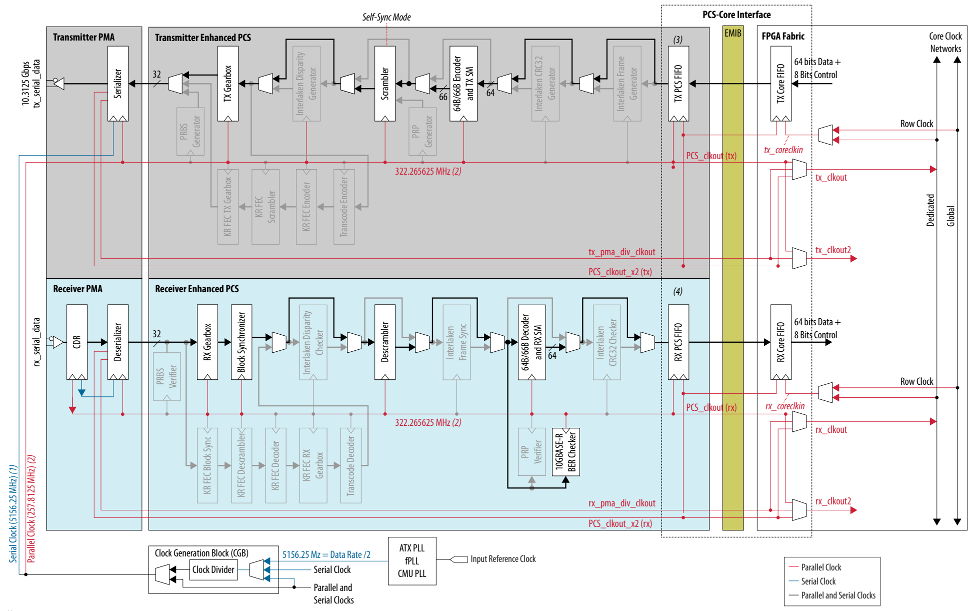
The 10GBASE-R parallel data interface is the 10 Gigabit Media Independent Interface (XGMII) that interfaces with the Media Access Control (MAC), which has the optional Reconciliation Sub-layer (RS).

The following 10GBASE-R variants are available from presets:

- 10GBASE-R
- 10GBASE-R Low Latency
- 10GBASE-R 1588
- 10GBASE-R w/ KR-FEC

Intel recommends that you use the presets for selecting the suitable 10GBASE-R variants directly if you are configuring through the Native PHY IP core.

Figure 123. Transceiver Channel Datapath and Clocking for 10GBASE-R (PCS-PMA Interface Width = 32 Bits)



Notes:
1. Value based on the clock division factor chosen.
2. Value calculated as data rate/PCS-PMA interface width.
3. This block is in phase compensation mode for the 10GBASE-R configuration and phase measuring FIFO for the 10GBASE-R with 1588 configuration.
4. This block is in 10GBASE-Rn mode for the 10GBASE-R configuration and register mode for the 10GBASE-R with 1588 configuration.

10GBASE-R with IEEE 1588v2

The output clock frequency of `tx_clkout` and `rx_clkout` to the FPGA fabric is based on the PCS-PMA interface width. For example, if the PCS-PMA interface is 32-bit, `tx_clkout` and `rx_clkout` run at 10.3125 Gbps/32-bit = 322.265625 MHz.

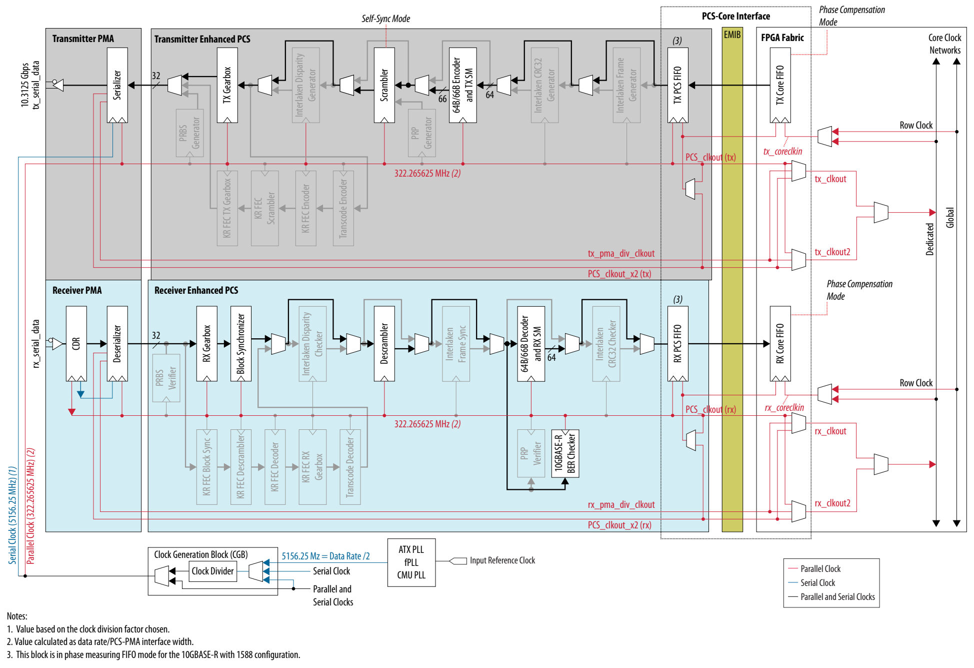
The 10GBASE-R PHY with IEEE 1588v2 uses both the TX Core FIFO and the RX Core FIFO in Phase Compensation mode. The effective XGMII data is running at 156.25 MHz interfacing with the MAC layer.

The IEEE 1588 Precision Time Protocol (PTP) is supported by the preset of the Native PHY IP core that configures 10GBASE-R PHY IP in IEEE-1588v2 mode. PTP is used for precise synchronization of clocks in applications such as:

- Distributed systems in telecommunications
- Power generation and distribution
- Industrial automation
- Robotics
- Data acquisition
- Test equipment
- Measurement

The protocol is applicable to systems communicating by local area networks including, but not limited to, Ethernet. The protocol enables heterogeneous systems that include clocks of various inherent precision, resolution, and stability to synchronize to a grandmaster clock.

Figure 124. Transceiver Channel Datapath and Clocking for 10GBASE-R with IEEE 1588v2 (PCS-PMA Interface Width = 32 Bits)



Notes:
1. Value based on the clock division factor chosen.
2. Value calculated as data rate/PCS-PMA interface width.
3. This block is in phase measuring FIFO mode for the 10GBASE-R with 1588 configuration.

10GBASE-R with KR-FEC

Intel Stratix 10 10GBASE-R has the optional FEC variant that also targets the 10GBASE-KR PHY. This provides a coding gain to increase the link budget and BER performance on a broader set of backplane channels as defined in Clause 69. It provides additional margin to account for variations in manufacturing and environment conditions. The additional TX FEC sublayer:

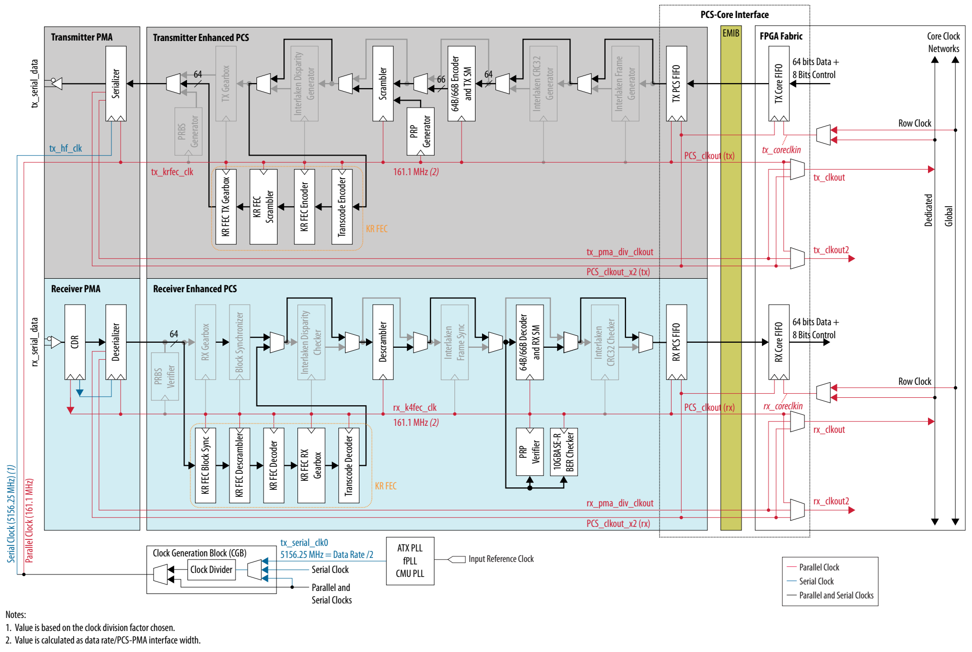
- Receives data from the TX PCS
- Transcodes 64b/66b words
- Performs encoding/framing
- Scrambles and sends the FEC data to the PMA

The RX FEC sublayer:

- Receives data from the PMA
- Performs descrambling
- Achieves FEC framing synchronization
- Decodes and corrects data where necessary and possible
- Recodes 64b/66b words and sends the data to the PCS

The 10GBASE-R with KR FEC protocol is a KR FEC sublayer placed between the PCS and PMA sublayers of the 10GBASE-R physical layer.

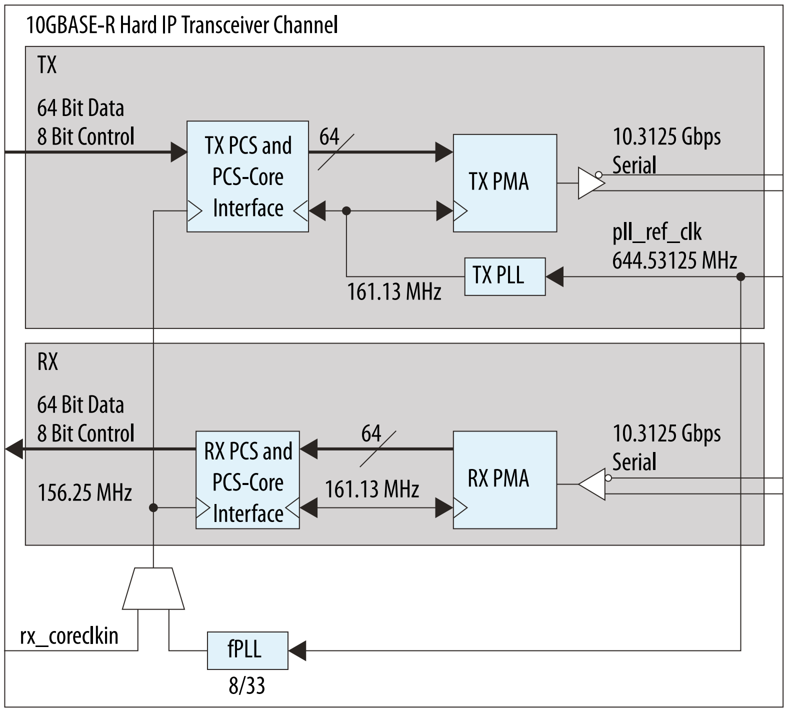
Figure 125. Transceiver Channel Datapath and Clocking for 10GBASE-R with KR FEC (PCS-PMA interface width = 64 bits)



The CMU PLL or the ATX PLLs generate the TX high speed serial clock.

Figure 126. Clock Generation and Distribution for 10GBASE-R with FEC Support

Example using a 64-bit PCS-PMA interface width.

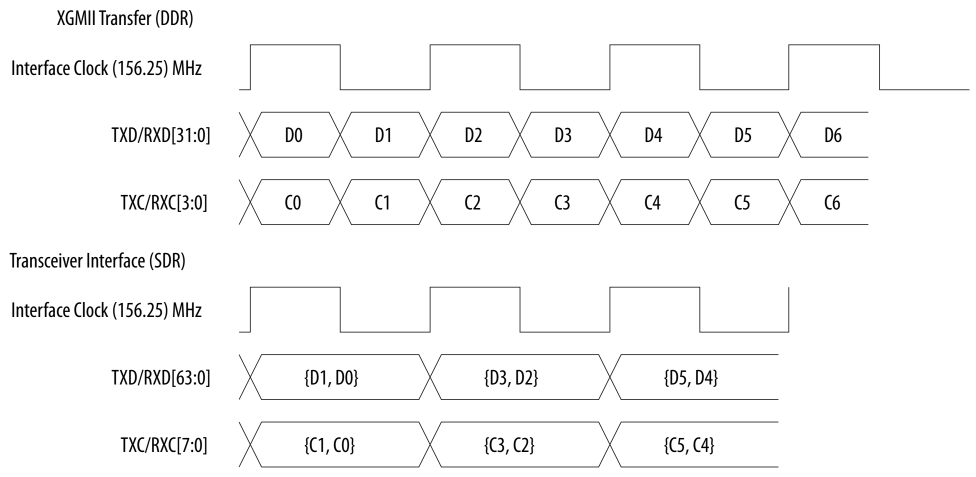


2.5.3.1.1. The XGMII Interface Scheme in 10GBASE-R

The XGMII interface, specified by IEEE 802.3-2008, defines the 32-bit data and 4-bit wide control character. These characters are clocked between the MAC/RS and the PCS at both the positive and negative edge (double datarate – DDR) of the 156.25 MHz interface clock.

The transceivers do not support the XGMII interface to the MAC/RS as defined in the IEEE 802.3-2008 specification. Instead, they support a 64-bit data and 8-bit control single datarate (SDR) interface between the MAC/RS and the PCS.

Figure 127. XGMII Interface (DDR) and Transceiver Interface (SDR) for 10GBASE-R Configurations



Note: Clause 46 of the IEEE 802.3-2008 specification defines the XGMII interface between the 10GBASE-R PCS and the Ethernet MAC/RS.

The dedicated reference clock input to the variants of the 10GBASE-R PHY can be run at either 322.265625 MHz or 644.53125 MHz.

For 10GBASE-R, you must achieve 0 ppm of the frequency between the read clock of TX phase compensation FIFO (PCS data) and the write clock of TX phase compensation FIFO (XGMII data in the FPGA fabric). This can be achieved by using the same reference clock as the transceiver dedicated reference clock input as well as the reference clock input for a core PLL (fPLL, for example) to produce the XGMII clock. The same core PLL can be used to drive the RX XGMII data. This is because the RX clock compensation FIFO is able to handle the frequency ppm difference of ±100 ppm between RX PCS data driven by the RX recovered clock and RX XGMII data.

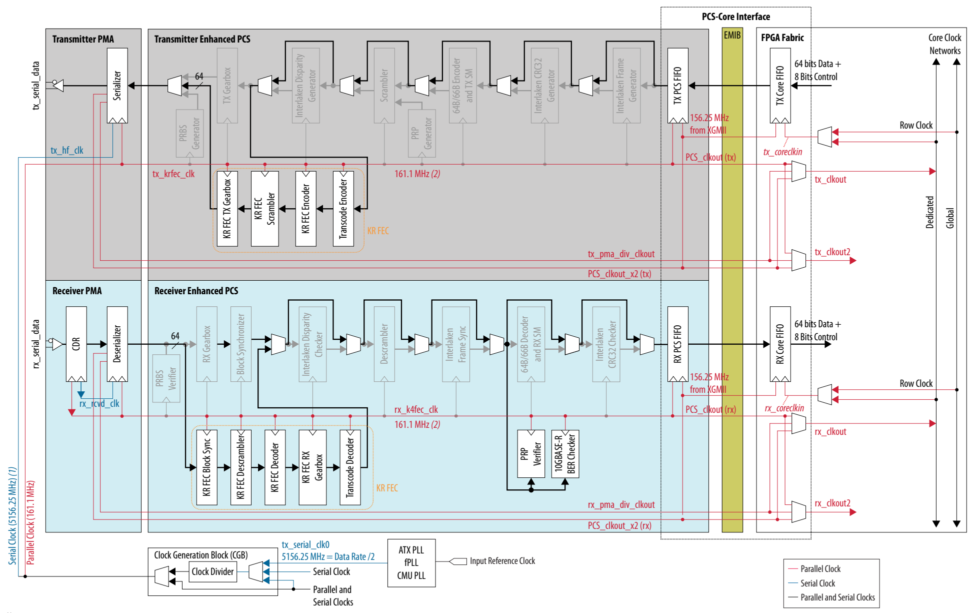
Note: 10GBASE-R is the single-channel protocol that runs independently. Therefore Intel recommends that you use the presets for selecting the suitable 10GBASE-R variants directly. If it is being configured through the Native PHY IP, the channel bonding option should be disabled. Enabling the channel bonding for multiple channels could degrade the link performance in terms of TX jitter eye and RX jitter tolerance.

Refer to the *Enhanced PCS FIFO Operation* section for more information about 10GBASE-R configurations.

2.5.3.2. 40GBASE-R with KR FEC Variant

The Native PHY IP core includes a configuration rule for the 40GBASE-R with KR FEC variant. 40GBASE-R is a four-channel protocol which uses four transceivers running at a 10.3125-Gbps datarate. Ethernet packets are stripped into four lanes and aligned at the receiving channels.

Figure 128. Transceiver Channel Datapath and Clocking for 40GBASE-R with KR FEC (PCS-PMA interface width = 64 bits)



Notes:
1. Value is based on the clock division factor chosen.
2. Value is calculated as data rate/PCS-PMA interface width.

2.5.4. CPRI

The common public radio interface (CPRI) is an industry cooperation that defines a publicly-available specification for the key internal interface of radio base stations between the radio equipment control (REC) and the radio equipment (RE).

The CPRI specification enables flexible and efficient product differentiation for radio base stations and independent technology evolution for RE and REC.

2.5.4.1. CPRI Line Rate Revisions

The following table shows the CPRI versions supported in the L-/H-Tile. Configure the Native PHY IP core based on your intended line rate and encoding scheme.

Table 118. CPRI Line Rate Revisions

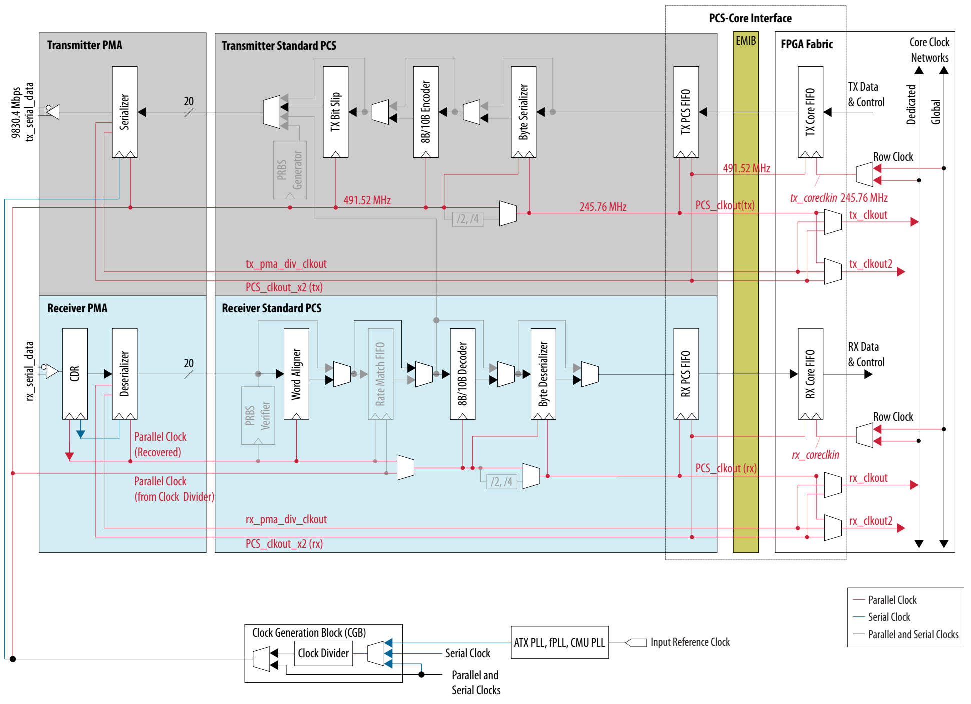
CPRI version	Line Rate	Encoding Scheme	PCS
CPRI 1.4	614.4 Mbps	8B/10B line coding	Standard PCS
CPRI 1.4	1228.8 Mbps	8B/10B line coding	Standard PCS
CPRI 1.4	2457.6 Mbps	8B/10B line coding	Standard PCS
CPRI 3.0	3072.0 Mbps	8B/10B line coding	Standard PCS
CPRI 4.1	4915.2 Mbps	8B/10B line coding	Standard PCS
CPRI 4.1	6144.0 Mbps	8B/10B line coding	Standard PCS
CPRI 4.2	9830.4 Mbps	8B/10B line coding	Standard PCS
CPRI 6.1	8110.08 Mbps	64B/66B line coding	Enhanced PCS

continued...

CPRI version	Line Rate	Encoding Scheme	PCS
CPRI 6.0	10137.6 Mbps	64B/66B line coding	Enhanced PCS
CPRI 6.1	12165.12 Mbps	64B/66B line coding	Enhanced PCS
CPRI 7.0	24330.24 Mbps	64B/66B line coding Implements in core logic	PCS Direct

2.5.4.2. Transceiver Channel Datapath and Clocking for CPRI

Figure 129. Transceiver Channel Datapath and Clocking for CPRI using the Standard PCS



Note: The PRBS generator, verifier, and rate match FIFO blocks are not available in the CPRI implementation.

Table 119. Clock Frequencies for Various CPRI Data Rates using the Standard PCS

These clock frequencies are based on a -1 transceiver speed grade device with 8B/10B encoding and the Standard PCS. Some configurations may not be available depending on the speed grade of your device..

Protocol, Data Rate (Mbps)	PMA Interface	PLD Interface	Byte Serializer	Double Rate Transfer	Data Transfer Mode	TX Core FIFO tx_coreclkin AND RX Core FIFO rx_coreclkin (MHz)
CPRI 1.4, 614.4	10	8/10	—	OFF	Half Rate	61.44
		16/20	x2	ON	Double Rate	61.44
				OFF	Full Rate	30.72
	20	16/20	—	ON	Double Rate	61.44

continued...

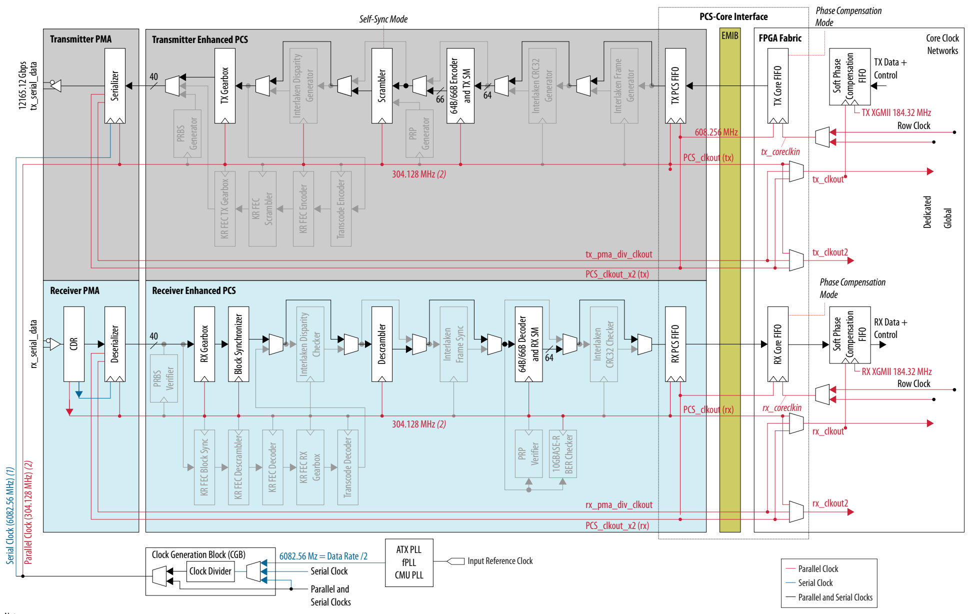
Protocol, Data Rate (Mbps)	PMA Interface	PLD Interface	Byte Serializer	Double Rate Transfer	Data Transfer Mode	TX Core FIFO tx_coreclkin AND RX Core FIFO rx_coreclkin (MHz)
				OFF	Half Rate	30.72
		32/40	x2	ON	Double Rate	30.72
				OFF	Full Rate	15.36
CPRI 1.4, 1228.8	10	8/10	—	OFF	Half Rate	122.88
		16/20	x2	ON	Double Rate	122.88
				OFF	Full Rate	61.44
	20	16/20	—	ON	Double Rate	122.88
				OFF	Half Rate	61.44
		32/40	x2	ON	Double Rate	61.44
				OFF	Full Rate	30.72
CPRI 1.4, 2457.6	10	8/10	—	OFF	Half Rate	245.76
		16/20	x2	ON	Double Rate	245.76
				OFF	Full Rate	122.88
	20	16/20	—	ON	Double Rate	245.76
				OFF	Half Rate	122.88
		32/40	x2	ON	Double Rate	122.88
				OFF	Full Rate	61.44
CPRI 3.0, 3072	10	8/10	—	OFF	Half Rate	307.2
		16/20	x2	ON	Double Rate	307.2
				OFF	Full Rate	153.6
	20	16/20	—	ON	Double Rate	307.0
				OFF	Half Rate	153.6
		32/40	x2	ON	Double Rate	153.6
				OFF	Full Rate	76.8
CPRI 4.1, 4915.2	10	8/10	—	OFF	Half Rate	N/A [40]
		16/20	x2	ON	Double Rate	N/A [40]
				OFF	Full Rate	N/A [40]
	20	16/20	—	ON	Double Rate	491.52
				OFF	Half Rate	245.76
		32/40	x2	ON	Double Rate	245.76
				OFF	Full Rate	122.88

continued...

[40] This is not a possible implementation for a -1 speed grade device.

Protocol, Data Rate (Mbps)	PMA Interface	PLD Interface	Byte Serializer	Double Rate Transfer	Data Transfer Mode	TX Core FIFO tx_coreclkin AND RX Core FIFO rx_coreclkin (MHz)
CPRI 4.1, 6144	10	8/10	—	OFF	Half Rate	N/A [(40)]
		16/20	x2	ON	Double Rate	N/A [(40)]
				OFF	Full Rate	N/A [(40)]
	20	16/20	—	ON	Double Rate	614.4
				OFF	Half Rate	307.2
		32/40	x2	ON	Double Rate	307.2
				OFF	Full Rate	153.6
CPRI 4.2, 9830.4	10	8/10	—	OFF	Half Rate	N/A [(40)]
		16/20	x2	ON	Double Rate	N/A [(40)]
				OFF	Full Rate	N/A [(40)]
	20	16/20	—	ON	Double Rate	N/A [(40)]
				OFF	Half Rate	491.52
		32/40	x2	ON	Double Rate	491.52
				OFF	Full Rate	245.76

Figure 130. Transceiver Channel Datapath and Clocking for CPRI using the Enhanced PCS



Notes:
1. Value based on the clock division factor chosen.
2. Value calculated as data rate/PCS-PMA interface width.

Table 120. Clock Frequencies for Various CPRI Data Rates using the Enhanced PCS

These clock frequencies are based on a -1 transceiver speed grade device with 64B/66B encoding and the Enhanced PCS. Some of these configurations are not available depending on the speed grade of your device.

Protocol,Datarate (Mbps)	PMA Interface	FPGA Fabric to/ from PCS-Core Interface	Double Rate Transfer	Data Transfer Mode	TX Core FIFO tx_coreclkin OR RX Core FIFO rx_coreclkin (MHz)
CPRI 6.1, 8110.08	40	66	OFF	Full Rate	202.752
			ON	Double Rate	405.504
CPRI 6.0, 10137.6	40	66	OFF	Full Rate	253.44
			ON	Double Rate	506.88
CPRI 6.1, 12165.12	40	66	OFF	Full Rate	304.128
			ON	Double Rate	608.256

Table 121. Interface Width Options for 10.1376 Gbps and 12.16512 Gbps Data Rates

Serial Data Rate (Mbps)	Interface Width	
	FPGA fabric - Enhanced PCS (bits)	Enhanced PCS - PMA (bits)
10137.6	66	32, 40, 64
12165.12	66	40, 64

2.5.4.2.1. TX PLL Selection for CPRI

Choose a transmitter PLL that fits your required data rate.

Table 122. TX PLL Supported Data Rates

ATX and fPLL support the clock bonding feature.

TX PLLs	Supported Data Rate (Mbps)
ATX	614.4, 1228.8, 2457.6, 3072, 4915.2, 6144, 8110.08, 9830.4, 10137.6, 12165.12, 24330.24
fPLL	614.4, 1228.8, 2457.6, 3072, 4915.2, 6144, 8110.08, 9830.4, 10137.6, 12165.12
CMU	614.4, 1228.8, 2457.6, 3072, 4915.2, 6144, 9830.4, 10137.6

Note:
- Channels that use the CMU PLL cannot be bonded. The CMU PLL that provides the clock can only drive channels in the transceiver bank where it resides.
- Over-sampling is required to implement 614.4Mbps.

2.5.4.2.2. Auto-Negotiation

When auto-negotiation is required, the channels initialize at the highest supported frequency and switch to successively lower data rates if frame synchronization is not achieved. If your design requires auto-negotiation, choose a base data rate that minimizes the number of PLLs required to generate the clocks required for data transmission.

By selecting an appropriate base data rate, you can change data rates by changing the local clock generation block (CGB) divider. If a single base data rate is not possible, you can use an additional PLL to generate the required data rates.

Table 123. Recommended Base Data Rates and Clock Generation Blocks for Available Data Rates

Data Rate (Mbps)	Base Data Rate (Mbps)	Local CGB Divider
1228.8	9830.4	8
2457.6	9830.4	4
3072.0	6144.0	2
4915.2	9830.4	2
6144.0	6144.0	1
8110.08	8110.08	1
9830.4	9830.4	1
10137.6	10137.6	1
12165.12	24330.24	2
24330.24	24330.24	1

2.5.4.3. Supported Features for CPRI

The CPRI protocol places stringent requirements on the amount of latency variation that is permissible through a link that implements these protocols.

CPRI (Auto) and CPRI (Manual) transceiver configuration rules are both available for CPRI designs in the Native PHY IP core. Both modes use the same functional blocks, but the configuration mode of the word aligner is different between the Auto and Manual modes. In CPRI (Auto) mode, the word aligner works in deterministic mode. In CPRI (Manual) mode, the word aligner works in manual mode.

To avoid transmission interference in time division multiplexed systems, every radio in a cell network requires accurate delay estimates with minimal delay uncertainty. Lower delay uncertainty is always desired for increased spectrum efficiency and bandwidth. The Intel Stratix 10 transceivers are designed with features to minimize the delay uncertainty for both RECs and REs.

Refer to the *Word Aligner in Deterministic Latency Mode* section for more information.

Related Information

- Deterministic Latency Use Model on page 146
- Word Aligner in Deterministic Latency Mode for CPRI on page 116

2.5.4.3.1. Word Aligner in Manual Mode for CPRI

When configuring the word aligner in CPRI (Manual), the word aligner parses the incoming data stream for a specific alignment character.

After `rx_digitalreset` deasserts, asserting the `rx_std_wa_patternalign` triggers the word aligner to look for the predefined word alignment pattern or its complement in the received data stream. Note that the behavior of the word aligner in Manual mode operates in different ways depending on the PCS-PMA interface width.

Table 124. Word Aligner Signal Status Behaviors in Manual Mode

PCS-PMA Interface Width	rx_std_wa_patternalign Behavior	rx_syncstatus Behavior	rx_patterndetect Behavior
10 bits	Level sensitive	One parallel clock cycle (When three control patterns are detected)	One parallel clock cycle
20 bits	Edge sensitive	Remains asserted until next rising edge of `rx_std_wa_patternalign`	One parallel clock cycle

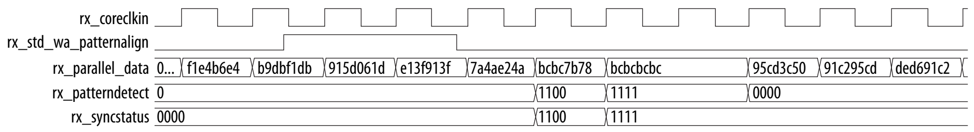
PCS-PMA Width = 10 bits

When the PCS-PMA interface width is 10 bits, 3 consecutive word alignment patterns found after the initial word alignment in a different word boundary causes the word aligner to resynchronize to this new word boundary if the `rx_std_wa_patternalign` remains asserted. `rx_std_wa_patternalign` is level sensitive. If you deassert `rx_std_wa_patternalign`, the word aligner maintains the current word boundary even when it finds the alignment pattern in a new word boundary. When the word aligner is synchronized to the new word boundary, `rx_patterndetect` and `rx_syncstatus` are asserted for one parallel clock cycle.

PCS-PMA Width =20 bits

When the PMA-PCS width is 20 bits, any alignment pattern found after the initial alignment in a different word boundary causes the word aligner to resynchronize to this new word boundary on the rising edge of `rx_std_wa_patternalign`. `rx_std_wa_patternalign` is edge sensitive. The word aligner maintains the current word boundary until the next rising edge of `rx_std_wa_patternalign`. When the word aligner is synchronized to the new word boundary, `rx_patterndetect` asserts for one parallel clock cycle and `rx_syncstatus` remains asserted until the next rising edge of `rx_std_wa_patternalign`.

Figure 131. Word Aligner in Manual Alignment Mode Waveform



2.5.4.4. Deterministic Latency

Refer to the *Deterministic Latency Use Model* section for details on latency calculation guidelines applicable to the transceiver PHY configured for CPRI systems.

Related Information

Deterministic Latency Use Model on page 146

2.6. Unused or Idle Transceiver Channels

Unused or idle transceiver clock network performance can degrade over time under the following conditions:

- FPGA devices are powered up to normal operating conditions and not configured.
- Designs that plan to use unused transceiver channels in the future by using dynamic reconfiguration or a new device programming file.
- Transceiver channels are used in the design but switched to the idle state during operation.

If you do not plan to use the unused transceiver channels in the future, no action is needed. Active transceiver channels and non-transceiver circuits are not impacted. For active transceiver channels, do not assert the `rx_analogreset` and `tx_analogreset` signals indefinitely.

To preserve the performance of unused transceiver channels, the Intel Quartus Prime software can switch the TX/RX channels on and off at low frequency using an internally-generated clock. To create clock activity on unused channels by way of a Quartus Settings File (`.qsf`) variable, either:

- Make a global assignment:

  ```
  set_global_assignment -name PRESERVE_UNUSED_XCVR_CHANNEL ON
  ```

- Use a per-pin assignment:

  ```
  set_instance_assignment -name PRESERVE_UNUSED_XCVR_CHANNEL ON -to pin_name
  ```

  For example, if the pin_name is Pin AB44, structure the per-pin assignment with the following syntax.

  ```
  set_instance_assignment -name PRESERVE_UNUSED_XCVR_CHANNEL ON -to AB44
  ```

When you perform this procedure, the Intel Quartus Prime software instantiates the clock data recovery (CDR) PLL corresponding to each unused receiver channel. The CDR PLL uses OSC_CLK_1 as reference clock and is configured to run at 1 Gbps. To use OSC_CLK_1 as the reference clock, the pin must be assigned a 25, 100, or 125 MHz clock. When you implement these assignments, it causes a power consumption increase per receiver channel.

Use the `.qsf` variable to preserve the performance of an unused receiver channel under the following conditions:

- When the transceiver channel is unused
- When the transceiver channel is configured as a simplex TX channel
- When the CDR in the receiver channel is configured as a CMU PLL
- When the receiver pin is configured as a reference clock pin

Use the `.qsf` variable to preserve the performance of an unused transmitter channel under the following conditions:

- When the transceiver channel is unused
- When the transceiver channel is configured as a simplex RX channel

If you do not perform this procedure, a critical warning similar to the following appears:

```
Critical Warning (19527): There are 95 unused RX channels and 95
unused TX channels in the design.
```

```
Info(19528): Add the QSF assignment 'set_instance_assignment -
name PRESERVE_UNUSED_XCVR_CHANNEL ON -to <pin_name>' for each
unused channel you want to preserve.

Info(19529): The above QSF assignment preserves the performance
of specified channels over time.
```

To disable this warning, use the following assignment:

```
set_instance_assignment -name MESSAGE_DISABLE 19527
```

When you apply the `.qsf` variable to a transceiver tile, at least one channel in a transceiver tile needs to be instantiated in the design. If all channels in a tile are unused but you plan to activate one or more in the future, instantiate a dummy channel in the tile.

If a transceiver channel in use is switched to an unused or idle state on the fly, implement the follow steps to preserve the performance of the channel during idle state:

1. If the channel is designed to support multiple configuration profiles, dynamically reconfigure the channel to the lowest speed profile in the design for less power consumption.
2. If the channel is not designed to support multiple configuration profiles, create a 1 Gbps channel profile in the Native PHY IP. Dynamically reconfigure the channel to this profile.
3. Use *Direct Reconfiguration Flow* to turn on internal serial loopback.
    - You can enable internal serial loopback by asserting the `rx_seriallpbken` control input port.
    - Alternatively, write 1'b1 to the RX Serial Loopback transceiver register. Refer to *Optional Reconfiguration Logic PHY- Control & Status*.
4. Use *Direct Reconfiguration Flow* to turn on the built-in PRBS generator in the TX PMA. Refer to *PRBS Generator*.
5. Turn off the TX buffer. This step is optional to prevent data transmission on the link. Assert the `tx_pma_elecidle` port to turn off the TX buffer output.
6. If DFE is enabled for the channel, use *Direct Reconfiguration Flow* to turn on DFE with all taps on. This incurs additional power. Refer to *Setting RX PMA Adaptation Modes*.

Related Information

- Direct Reconfiguration Flow on page 409
- Setting RX PMA Adaptation Modes on page 459
- PRBS Generator on page 463
- Optional Reconfiguration Logic PHY- Control & Status on page 467

2.7. Simulating the Native PHY IP Core

Use simulation to verify the Native PHY transceiver functionality. The Intel Quartus Prime Pro Edition software supports register transfer level (RTL) and gate-level simulation in both ModelSim®Intel and third-party simulators. You run simulations using your Intel Quartus Prime project files.

The following simulation flows are available:

- Scripting IP Simulation—In this flow you perform the following actions:
  1. Run the ip-setup-simulation utility to generate a single simulation script that compiles simulation files for all the underlying IPs in your design. This script needs to be regenerated whenever you upgrade or modify IPs in the design.
  2. Create a top-level simulation script for compiling your testbench files and simulating the testbench. It sources the script generated in the first action. You do not have to modify this script even if you upgrade or modify the IPs in your design.
- Custom Flow—This flow allows you to customize simulation for more complex requirements. You can use this flow to compile design files, IP simulation model files, and Intel simulation library models manually.

You can simulate the following netlist:

- The RTL functional netlist—This netlist provides cycle-accurate simulation using Verilog HDL, SystemVerilog, and VHDL design source code. Intel and third-party EDA vendors provide the simulation models.

Prerequisites to Simulation

Before you can simulate your design, you must have successfully passed Intel Quartus Prime Pro Edition Analysis and Synthesis.

Note: When simulating your design, you must apply a power-on reset (two `reconfig_clk` cycles) to the `reconfig_reset` signal.

Related Information

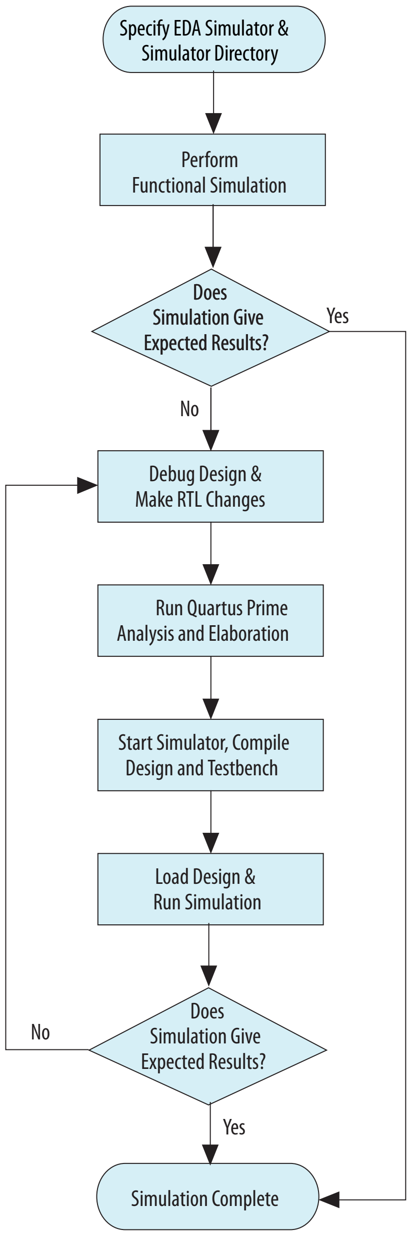
- Simulating Intel Designs
- Intel FPGA Knowledge Base

2.7.1. How to Specify Third-Party RTL Simulators

The following figure illustrates the high-level steps for using the NativeLink with Third-Party EDA RTL simulator.

Send Feedback

Figure 132. Using NativeLink with Third-Party Simulators


Specify EDA Simulator & Simulator Directory
Perform Functional Simulation
Does Simulation Give Expected Results?
Yes
No
Debug Design & Make RTL Changes
Run Quartus Prime Analysis and Elaboration
Start Simulator, Compile Design and Testbench
Load Design & Run Simulation
No
Does Simulation Give Expected Results?
Yes
Simulation Complete

Complete the following steps to specify the directory path and testbench settings for your simulator:

1. On the **Assignments** menu, click **Settings**, and then click **EDA Tool Settings**.
2. In the **Simulation** list, select your simulator. The following table lists the directories for supported third-party simulators:

Table 125. Simulator Path

Simulator	Path
Mentor Graphics ModelSim Mentor Graphics QuestaSim	*<drive>*:\*<simulator install path>*\win32 (Windows) /*<simulator install path>*/**bin** (Linux)
Synopsys VCS/VCS MX	/*<simulator install path>*/**bin** (Linux)
Cadence Incisive Enterprise	/*<simulator install path>*/**tools**/**bin** (Linux)
Aldec Active-HDL Aldec Riviera-Pro	*<drive>*:\*<simulator install path>*\bin (Windows) /*<simulator install path>*/**bin** (Linux)

3. To enable your simulator, on the **Tools** menu, click **Options** and then click **License Setup** . Make necessary changes for EDA tool licenses.
4. Compile your design and testbench files.
5. Load the design and run the simulation in the EDA tool.

To learn more about third-party simulators, click the appropriate link below.

Related Information

- Mentor Graphics ModelSim and QuestaSim Support
- Synopsys VCS and VCS MX Support
- Cadence Incisive Enterprise Simulator Support
- Aldec Active-HDL and Riviera-Pro Support

2.7.2. Scripting IP Simulation

The Intel Quartus Prime Pro Edition software supports the use of scripts to automate simulation processing in your preferred simulation environment. You can use your preferred scripting methodology to control simulation.

Intel recommends the use of a version-independent top-level simulation script to control design, testbench, and IP core simulation. Because Intel Quartus Prime Pro Edition-generated simulation file names may change.

You can use the ip-setup simulation utility to generate or regenerate underlying setup scripts after any software or IP version upgrade or regeneration. Use of a top-level script and ip-setup-simulation eliminates the requirement to manually update simulation scripts.

2.7.2.1. Generating a Combined Simulator Setup Script

Platform Designer system generation creates the interconnect between components. It also generates files for synthesis and simulation, including the **.spd** files necessary for the ip-setup-simulation utility.

The Intel Quartus Prime Pro Edition software provides utilities to help you generate and update IP simulation scripts. You can use the ip-setup-simulation utility to generate a combined simulator setup script, for all Intel FPGA IP in your design, for each supported simulator. You can subsequently rerun ip-setup-simulation to automatically update the combined script. Each simulator's combined script file contains a rudimentary template that you can adapt for integration of the setup script into a top-level simulation script.

Related Information

Quartus Prime Pro Edition Handbook Volume 3: Verification
Provides detailed information about the steps to generate top-level simulation script.

2.7.2.2. Steps for a .do file for Simulation

The following procedure shows the steps required to generate a **.do** file for simulation.

STEP 3: Generate IP Simulation setup script and create top_level simulation script.

1. From the Intel Quartus Prime Tools menu, select **Generate Simulator Setup Script for IP**.
2. In the **Generate Simulator Setup Script for IP** dialog box, do not enable **Compile all design files to the default library work**.
3. Leave **Use relative paths whenever possible** checked.
4. Click **OK**.
5. Once script generation is complete, in a file explorer window, go to the **project directory**.
6. Inside the **mentor directory**, open the file **msim_setup.tcl** in a text editor.
7. In the **msim_setupt.tcl** file, scroll up to the commented line that reads `# # TOP-LEVEL TEMPLATE – BEGIN`. Then scroll down to locate the line that reads `# # TOP-LEVEL TEMPLATE – END`.
8. Copy the two commented lines above and all of the lines between them to the clipboard.
9. Create a new text file in a text editor and paste the contents from the clipboard into this new file.

   Again, this new file should begin with `# # TOP-LEVEL TEMPLATE – BEGIN` and end with `# # TOP-LEVEL TEMPLATE – END`. (Doing so correctly aligns the upcoming modification instructions.)
10. Save the file as **mentor_top.do** into the **project directory** (NOT the **mentor directory**).
11. Make the following modifications to **mentor_top.do**:
    - Uncomment line 11 of the **DO file**, the `setQSYS_SIMDIR`… command.

      Change <script generation output directory> to the project directory in which the simulation runs.
    - Uncomment line 14, the `source $QSYS_SIMDIR/mentor/msim_setup.tcl` command.

      This command sources the ModelSim simulation setup script you generated.
    - Uncomment line 20, the `dev_com` command.

      This command uses the `dev_com` alias to compile all of the device-specific simulation library files.
    - Uncomment line 23, the `com` command.

      This command compiles all of the IP core-specific simulation files.
    - Uncomment line 29, the `vlog` command.

- For this `vlog` command, type the following:

  `vlog -work work -vlog01compat < all top_level design files > <test bench>`

  This compiles the entire non-IP core files used in the simulation.
- Uncomment line 34, the `set TOP_LEVEL_NAME`... command. Replace `<simulation top >` with `<name of test bench file >`. For example, if the test bench is **design_tb.v**, then issue the command as follows:

  `set TOP_LEVEL_NAME design_tb`
- Uncomment line 37, the `set USER_DEFINED_ELAB_OPTIONS`...command. Replace `<elaboration options>` with `voptargs="+acc"`.

  This line lets you specify arguments that get called with the simulator's `vsim` command. In particular, you are allowing for simulator optimization while maintaining full visibility of internal signals.
- Uncomment line 40, the `elab` command.
- This alias launches the simulation.
- Uncomment line 43, run `–a` command.

12. Save the file **mentor top.do**.

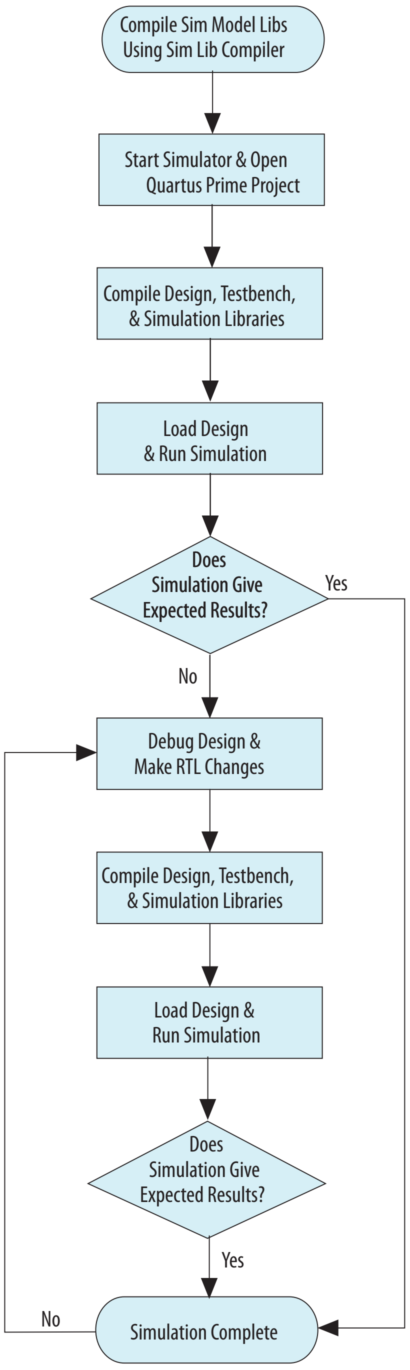
2.7.3. Custom Simulation Flow

The custom simulation flow allows you to customize the simulation process for more complex simulation requirements. This flow allows you to control the following aspects of your design:

- Component binding
- Compilation order
- Run commands
- IP cores
- Simulation library model files

The following figure illustrates the steps for custom flow simulation. If you use a simulation script, you can automate some of the steps.

Figure 133. Custom flow Simulation


Compile Sim Model Libs Using Sim Lib Compiler
Start Simulator & Open Quartus Prime Project
Compile Design, Testbench, & Simulation Libraries
Load Design & Run Simulation
Does Simulation Give Expected Results?
Yes
No
Debug Design & Make RTL Changes
Compile Design, Testbench, & Simulation Libraries
Load Design & Run Simulation
Does Simulation Give Expected Results?
Yes
No
Simulation Complete

2.7.3.1. How to Use the Simulation Library Compiler

The Simulation Library Compiler compiles Intel simulation libraries for supported simulation tools, and saves the simulation files in the output directory you specify.

Note: Because the software provides precompiled simulation libraries, you do not have to compile simulation libraries if you are using the software.

Complete the following steps to compile the simulation model libraries using the Simulation Library Compiler:

1. On the Tools menu, click **Launch Simulation Library Compiler**.
2. Under **EDA simulation tool**, for the **Tool name**, select your simulation tool.
3. Under **Executable location**, browse to the location of the simulation tool you specified. You must specify this location before you can run the EDA Simulation Library Compiler.
4. Under **Library families**, select one or more family names and move them to the **Selected families** list.
5. Under **Library language**, select **Verilog**, **VHDL**, or both.
6. In the **Output directory** field, specify a location to store the compiled libraries.
7. Click **Start Compilation**.

Complete the following steps to add the simulation files to your project:

1. On the Assignments menu, click **Settings**.
2. In the **Category** list, select **Files**.
3. Click **Browse** to open the **Select File** dialog box and select one or more files in the **Files** list to add to your project.
4. Click **Open**, and then **Add** to add the selected files to your project.
5. Click **OK** to close the **Settings** dialog box.

Related Information

- Preparing for EDA Simulation
- Intel Simulation Models

2.7.3.2. Custom Simulation Scripts

You can automate simulations by creating customized scripts. You can generate scripts manually. In addition, you can use ip-setup-simulation utility to generate a simulation script as a template and then make the necessary changes. The following table shows a list of script directories NativeLink generates.

Table 126. Custom Simulation Scripts for Third Party RTL Simulation

Simulator	Simulation File	Use
Mentor Graphics ModelSim or QuestaSim	**/simulation/ modelsim/ modelsim_setup.do** Or **mentor/msim_setup.tcl**	Source directly with your simulator. Run `do msim_setup.tcl`, followed by `ld_debug`. If you have more than one IP, each IP has a dedicated `msim_setup.tcl` file. Make sure that you combine all the files included in the `msim_setup.tcl` files into one common `msim_setup.tcl` file.
Aldec Riviera Pro	**/simulation/ aldec/ rivierapro_setup.tcl**	Source directly with your simulator.

continued...

Simulator	Simulation File	Use
Synopsys VCS	**/simulation/synopsys/vcs/vcs_setup.sh**	Add your testbench file name to this file to pass the testbench file to VCS using the `–file` option. If you specify a testbench file for NativeLink and do not choose to simulate, NativeLink generates a script that runs VCS.
Synopsys VCS MX	**/simulation/synopsys/vcsmx/vcsmx_setup.sh**	Run this script at the command line using `quartus_sh–t` *`<script>`*. Any testbench you specify with NativeLink is included in this script.
Cadence Incisive (NCSim)	**/simulation/cadence/ncsim_setup.sh**	Run this script at the command line using `quartus_sh –t` *<script>*. Any testbench you specify with NativeLink is included in this script.

2.8. Implementing the Transceiver Native PHY Layer in L-Tile/H-Tile Revision History

Document Version	Changes
2020.03.03	Made the following changes: • Updated the following figures to make it clear that `rx_clkout` is driven by CDR. — FIFO Latency Calculation — Transceiver Channel Datapath for PIPE Gen1/Gen2 Configurations — Transceiver Channel Datapath for PIPE Gen1/Gen2/Gen3 Configurations — PCIe Reverse Parallel Loopback Mode Datapath — Transceiver Channel Datapath and Clocking for Interlaken — Transceiver Channel Datapath and Clocking for 10GBASE-R (PCS-PMA Interface Width = 32 Bits) — Transceiver Channel Datapath and Clocking for 10GBASE-R with IEEE 1588v2 (PCS-PMA Interface Width = 32 Bits) — Transceiver Channel Datapath and Clocking for 10GBASE-R with KR FEC (PCSPMA interface width = 64 bits) — Transceiver Channel Datapath and Clocking for 40GBASE-R with KR FEC (PCSPMA interface width = 64 bits) — Transceiver Channel Datapath and Clocking for CPRI using the Standard PCS — Transceiver Channel Datapath and Clocking for CPRI using the Enhanced PCS • Clarified ODI support for L-Tile. • Updated *Unused or Idle Transceiver Channels*.
2020.01.29	Made the following change: • Added "Implements in core logic" to the "Encoding Scheme" column in *CPRI Line Rate Revisions* and changed the PCS column from “Enhanced PCS” to “PCS Direct.”
2019.10.02	Made the following changes: • For **TX Core Interface FIFO Mode ➤ Register** and **RX PCS-Core Interface FIFO Mode ➤ Register**, added the restriction that "This mode is limited to PCS Direct with interface widths of 40 bits or less." • Added the **Enable PIPE EIOS RX protection** parameter in the "General, Common PMA Options, and Datapath Options" table.
2019.06.07	Made the following changes: • Added two parameters to the Pre-Emphasis First Pre-Tap Polarity setting in Analog PMA Settings Parameters. • Added clarification about the polarity inversion support differences between Standard and Enhanced PCS to Polarity Inversion.
2019.03.22	Made the following changes: • Added assignments for "Output Swing Level (VOD)" and "Pre-Emphasis First Post-Tap Polarity." • Updated *How to Enable ODI*, *Scanning the Horizontal Eye Opening*, *Scanning the Horizontal and Vertical Phases*, and *How to Disable ODI*.

continued...

Document Version	Changes
2018.10.23	Made the following change: • Changed `rx_enh_data_valid` in the "RX PCS-Core Interface FIFO" table to an output.
2018.10.05	Made the following changes: • Changed the pin requirements for `OSC_CLK_1` in the "Unused or Idle Transceiver Channels" section. • Changed the FIFO mode from Register to Phase Compensation in the "Transceiver Channel Datapath and Clocking for 10GBASE-R with IEEE 1588v2 (PCS-PMA Interface Width = 32 Bits)" figure. • Changed the critical warning messages in the "Unused or Idle Transceiver Channels" section. • Added note to the "Simulating the Native PHY IP Core" section. • Changed the direction of the `rx_word_marking_bit` port in the "RX PCS-Core Interface Ports: Parallel Data, Control, and Clocks" table. • Changed the preset coefficients recommendation in the "Preset Mappings to TX De-emphasis" section. • Changed the preset coefficients recommendation in the "Link Equalization for Gen3" section. • Changed the 10GBASE-R configuration description in the "Enhanced PCS FIFO Operation" section.
2018.07.06	Made the following changes: • Clarified the values for the **Slew Rate Control** parameter in the "TX Analog PMA Settings Options" table. • Clarified the attribute values for slew rate in the "Transmitter QSF Assignment Attributes" table. • Removed the note about the `rx_std_wa_patternalign` port from the "Word Aligner Synchronous State Machine Mode" section. • Changed the "Transceiver Channel Datapath and Clocking for 10GBASE-R with IEEE 1588v2 (PCS-PMA Interface Width = 32 Bits)" figure. • Clarified the requirements for `rx_fifo_align_clr` duration in the "RX Multi-lane FIFO Deskew State Machine" section. • Clarified the requirement for `rx_fifo_align_clr` duration on exit from reset in the "State Flow of the RX FIFO Deskew" figure. • Changed the description of `rx_std_byterev_ena[<n>-1:0]` in the "Bit Reversal and Polarity Inversion" table. • Added a note about designing with the **Use default TX PMA analog settings** and **Use default RX PMA analog settings** options as a starting point in the "Analog PMA Settings Parameters" section. • Added further description of the **Provide sample QSF assignments** in the "Sample QSF Assignment Option" table. • Changed the PCS-PMA width to 32 in the "Transceiver Channel Datapath and Clocking for 10GBASE-R (PCS-PMA Interface Width = 32 Bits)" figure. • Changed the command to disable the warning message for unused transceiver channels in the "Unused or Idle Transceiver Channels" section. • Added QSF syntax examples for most of the parameters in the "TX Analog PMA Settings Options" table. • Added QSF syntax examples for most of the parameters in the "RX Analog PMA Settings Options" table. • Changed the data rate ranges and register settings in the "ODI Bandwidth Settings" table. • Changed *Scanning the Horizontal Eye Opening*. • Changed *Scanning the Horizontal and Vertical Phases*. • Added a footnote to the `code_violation_status` signal in the "Simplified Data Interface=Disabled, Double-Rate Transfer=Enabled" table. • Updated the critical warning message in the "Unused or Idle Transceiver Channels" section. • Added clarification about the adaptation engine in the "Using RX in Adaptive Mode" section. • Added ODI support on L-Tile devices in the "On-die Instrumentation" section. • Changed the base data rate for the 12165.12 Mbps data rate in the "Recommended Base Data Rates and Clock Generation Blocks for Available Data Rates" table. • Added an example to the per-pin `.qsf` assignment instruction in the "Unused or Idle Transceiver Channels" section.
2018.03.16	Made the following changes:

continued...

Document Version	Changes
	• Changed the functionality and description of`rx_control` bit [9:8] to "Unused" in the "Bit Encodings for Basic (Enhanced PCS) with 66-bit word, Basic with KR FEC, 40GBASE-R with KR FEC" table. • Added steps 5, 6, and 12 in the "How to Implement Double Rate Transfer Mode" section. • Changed the following figures in the "RX Bitslip" section: — "RX Bitslip in 8-bit Mode" — "RX Bitslip in 10-bit Mode" — "RX Bitslip in 16-bit Mode" — "RX Bitslip in 20-bit Mode" • Changed the following figure in the "Word Aligner Manual Mode" section: — "Manual Mode when the PCS-PMA Interface Width is 8 Bits" • Added details about how to enable the transceiver toolkit capability in the "Dynamic Reconfiguration Parameters" section. • Added the **Enable tx_coreclkin2** port parameter to the "PCS-Core Interface Parameters" table. • Added the **RX PMA analog mode rules** parameter to the "Analog PMA Settings" table. • Removed the "Manual Mode when the PCS-PMA Interface Width is 10 Bits" figure. • Removed the "Manual Mode when the PCS-PMA Interface Width is 16 Bits" figure. • Removed the "Manual Mode when the PCS-PMA Interface Width is 20 Bits" figure. • Changed the "Synchronization State Machine Mode when the PCS-PMA Interface Width is 20 Bits" figure. • Removed the "Word Aligner in Deterministic Latency Mode Waveform" figure. • Removed the "High BER" figure. • Removed the "Block Lock Assertion" figure. • Changed the "Idle Ordered-Set Generation Example" figure. • Changed the "RX Polarity Inversion" figure. • Changed the following figures in the "RX Data Bitslip" section: — "RX Bitslip in 8-bit Mode" — "RX Bitslip in 10-bit Mode" — "RX Bitslip in 16-bit Mode" — "RX Bitslip in 20-bit Mode" • Changed the "TX Bit Reversal" figure. • Changed the "RX Bit Reversal" figure. • Changed the "TX Byte Reversal" figure. • Changed the "RX Byte Reversal" figure. • Updated the value of the **Output Swing Level (VOD)** parameter in the "TX Analog PMA Settings Options" table. • Changed the descriptions for the following parameters in the "RX Analog PMA Settings Options" table: — CTLE AC Gain — CTLE EQ Gain — VGA DC Gain • Added clarification about ways to configure TX PMA settings in the "TX PMA Use Model" section. • Added clarification about ways to select your CTLE gain value in the "Manual Mode" section. • Defined the SKP symbol and added a note describing why it is used in the "Gen1 and Gen2 Clock Compensation" section. • Changed the values or added descriptions for the following parameters in the "TX Analog PMA Settings Options" table: — **Output Swing Level (VOD)** — **Pre-Emphasis First Pre-Tap Polarity** — **Pre-Emphasis First Pre-Tap Magnitude** — **Pre-Emphasis First Post -Tap Polarity** — **Pre-Emphasis First Post -Tap Magnitude** — **On-Chip Termination** — **Slew Rate Control**

continued...

Document Version	Changes
	• Changed the values or added descriptions for the following parameters in the "RX Analog PMA Settings Options" table: — **RX On-chip Termination** — **CTLE AC Gain** — **CTLE EQ Gain** — **VGA DC Gain** • Changed the "Synchronization State Machine Mode when the PCS-PMA Interface Width is 16 Bits" figure. • Changed the "Word Aligner in Deterministic Latency Mode 16 Bits Waveform" figure. • Changed the following figures in the TX Data Bit Slip section: — TX Bitslip in 8-bit Mode — TX Bitslip in 10-bit Mode — TX Bitslip in 16-bit Mode — TX Bitslip in 20-bit Mode • Changed the "Idle Oredered-Set Generation Example" figure. • Changed the polarity inversion mode selections in the "RX Data Polarity Inversion" section. • Changed the bit reversal mode selections in the "Transmitter Bit Reversal" section. • Changed the bit reversal mode selections in the "Receiver Bit Reversal" section. • Changed the byte reversal mode selections in the "Receiver Byte Reversal" section. • Changed the byte reversal mode selections in the "Transmitter Byte Reversal" section. • Changed the note in the "Debug Functions" section. • Changed the note in the "On-die Instrumentation" section. • Added further description to the `rx_set_locktoref[<n>-1:0]` and `rx_set_locktoref[<n>-1:0]` ports in the "RX PMA Ports" table. • Added a note to the "Transceiver PHY PCS-to-Core Interface Reference Port Mapping" section. • Added description for how to enable/disable serial loopback in the "Enabling and Disabling Loopback" section. • Changed the list of restrictions for placing channels for PIPE configurations in the "How to Place Channels for PIPE Configurations" section. • Changed the name of the Rate Match modes in the "Clock Compensation Using the Standard PCS" section. • Added a sentence for "Enable TX bit/byte reversal" parameter: "TX bit/byte reversal ports are not available but can be changed via soft registers. RX bit reversal ports are available." • Added "`rx_syncstatus` is bus dependent on the width of the parallel data. For example, when the parallel data width is 32 bits, then `rx_syncstatus` is a 4 bit bus. The final expected value is 1'hf, indicating the control character is identified at the correct location in the 32 bit parallel word." to the "rx_syncstatus[<n><w>/<s>-1:0]". • Added "The 2 alignment markers valid status is captured in the 2 bit of `rx_std_wa_ala2size` signal. When both the markers are matched, then the value of the signal is 2'b11." to "rx_std_wa_a1a2size[<n>-1:0]". • Added the "Timing Closure Recommendations" section. • Added the two available PIPE refclk assignment settings to "Native PHY IP Core Parameter Settings for PIPE."
2017.08.11	Made the following changes: • Added link to the PCIe solutions guidelines in the "Configuring the Native PHY IP Core" section. • Added details about the number of GXT channels supported per tile in the "GXT Channels" section. • Changed the description for the **Enable rx_pma_clkslip port** parameter in the "RX PMA Optional Ports" table. • Added units of measure to the **RX On-chip Termination** parameter values in the "RX Analog PMA Settings Options" table. • Added descriptions for the following parameters in the "KR-FEC Parameters" table: — **Enable tx_enh_frame port** — **Enable rx_enh_frame port** — **Enable rx_enh_frame_diag_status port** • Added further description to the **PCI Express Gen3 rate match FIFO mode** parameter in the "Rate Match FIFO Parameters" table.

continued...

Document Version	Changes
	• Changed the description of the **Share reconfiguration interface** parameter in the "Dynamic Reconfiguration" table. • Changed the direction and updated the description of the `rx_pma_clkslip` signal in the "RX PMA Ports" table. • Changed the clock domains of the following signals in the "TX PCS-Core Interface FIFO" table: — `tx_fifo_empty[<n>-1:0]` — `tx_fifo_pempty[<n>-1:0]` — `rx_fifo_full[<n>-1:0]` — `rx_fifo_pfull[<n>-1:0]` • Added the "Calculating Latency through the Word Aligner" section. • Added the "CPRI Line Rate Revisions" table. • Changed the "Transceiver Channel Datapath and Clocking for CPRI using the Standard PCS" figure. • Added the "Clock Frequencies for Various CPRI Data Rates using the Standard PCS" table. • Added the "Transceiver Channel Datapath and Clocking for CPRI using the Enhanced PCS" figure. • Added the "Clock Frequencies for Various CPRI Data Rates using the Enhanced PCS" table. • Added the "FIFO Latency Calculation" section. • Added the CPRI chapter. • Updated the descriptions for the **Enable rx_fifo_pfull port** and **Enable rx_fifo_pempty port** parameters in the " PCS-Core Interface Parameters" table. • Added descriptions for the **CTLE AC Gain**, **CTLE EQ Gain**, and **VGA DC Gain** parameters in the "RX Analog PMA Settings Options" table. • Updated the descriptions for the `tx_fifo_pfull[<n>-1:0]` and `tx_fifo_pempty[<n>-1:0]` ports in the "TX PCS-Core Interface FIFO" table. • Added the PCIe Adaptive Mode column to and changed the bit settings for `adp_dc_ctle_mode_sel`, `adp_dc_ctle_mode0_win_start`, `adp_dc_ctle_onetime`, and `adp_vga_ctle_low_limit` in the "RX Adaption Mode Attributes" table. • Changed the footnote in the "Register Chain Minimum Hold Time Calculations" table. • Completely restructured the table in and added supporting text to the "Transceiver to FPGA Fabric Transfer" section. • Changed the description of the **Share reconfiguration interface** parameter in the "Dynamic Reconfiguration" table.
2017.06.06	Made the following changes: • Removed QPI options from the "TX PMA Options" table. • Added the "PMA Functions" section. • Added the "Debug Functions" section. • Removed the **Enable feedback compensation bonding** parameter from the "fPLL IP Core Parameter Settings for PIPE" section. • Removed the **Enable feedback compensation bonding** parameter from the "ATX PLL IP Core Parameter Settings for PIPE" section. • Changed the parameter name, **Store current configuration to profile**, to match the GUI in the "Configuration Profiles" table. • Changed the figures in the "How to Connect TX PLLs for PIPE Gen1, Gen2, and Gen3 Modes" section. • Added the "Standard PCS Options" table. • Changed the logical PCS master channel number for the x1 PIPE configuration in the "Logical PCS Master Channel for PIPE Configuration" table. • Changed the values for **Pre-Emphasis First Pre-Tap Magnitude** and **Pre-Emphasis First Post - Tap Magnitude** in the "TX Analog PMA Settings Options" table. • Updated some parameters and descriptions in the "General, Common PMA Options, and Datapath Options" table. • Removed the **Selected TX PCS bonding clock network** parameter from the "TX Clock Options" table. • Removed the **VGA Half BW Enable** parameter from the "RX Analog PMA Settings" table. • Changed the name of the parameters in the "Byte Serializer and Deserializer Parameters" table to match the GUI. • Changed the description for `rx_bitslip[<n>-1:0]` in the "Gearbox" table.

continued...

Document Version	Changes
	• Added the "Setting RX PMA Adaptation Modes" section. • Added the "Word Aligner in Deterministic Latency Mode for CPRI" section. • Removed the Best Case column from the "Register Chain Minimum Hold Time Calculations" table. • Removed the Best Case row from the "Register Chain Minimum Hold Time Calculations" table. • Updated the list of ports in the "PRBS Control and Status Ports" section. • Added the "PRBS Soft Accumulators Use Model" section.
2017.03.08	Made the following changes: • Changed the description of **VGA Half BW Enable** in the "RX Analog PMA Settings Options" table.
2017.02.17	Made the following changes: • Added the "GXT Channels" section. • Added the "Reconfiguring Between GX and GXT Channels" section. • Changed the description for the **Enable rx_pma_clkslip port** option in the "RX PMA Optional Ports" table. • Changed the list options for TX and RX analog PMA settings in the "Analog PMA Settings Parameters" section. • Removed parameters from the "TX Analog PMA Settings Options" and "RX Analog PMA Settings Options" tables. • Removed the "TX PMA Optional Ports - PMA QPI Options" table. • Changed the description for the `rx_pma_clkslip` port in the "RX PMA Ports" table. • Added the **Enable PCS reset status ports** option in the "PCS-Core Interface Parameters" table. • Added the "Implementing the PHY Layer for Transceiver Protocols" section.
2016.12.21	Initial release.

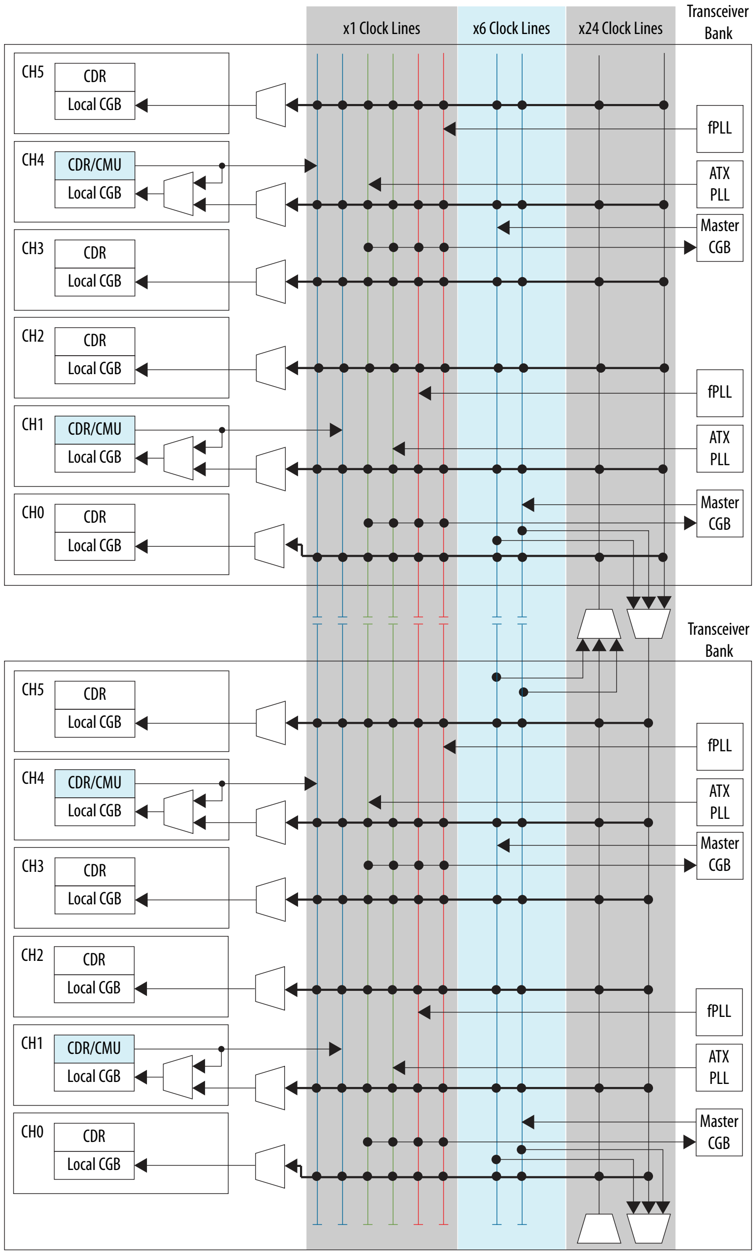
3. PLLs and Clock Networks

This chapter describes the transceiver phase locked loops (PLLs), internal clocking architecture, and the clocking options for the transceiver and the FPGA fabric interface.

Transceiver banks have six transceiver channels. There are two advanced transmit (ATX) PLLs, two fractional PLLs (fPLL), two CMU PLLs, and two Master clock generation blocks (CGB) in each bank.

The Intel Stratix 10 transceiver clocking architecture supports both bonded and non-bonded transceiver channel configurations. Channel bonding is used to minimize the clock skew between multiple transceiver channels. For Intel Stratix 10 transceivers, the term bonding can refer to PMA bonding as well as PMA and PCS bonding. For more details, refer to the *Channel Bonding* section.

Figure 134. Intel Stratix 10 PLLs and Clock Networks



Related Information

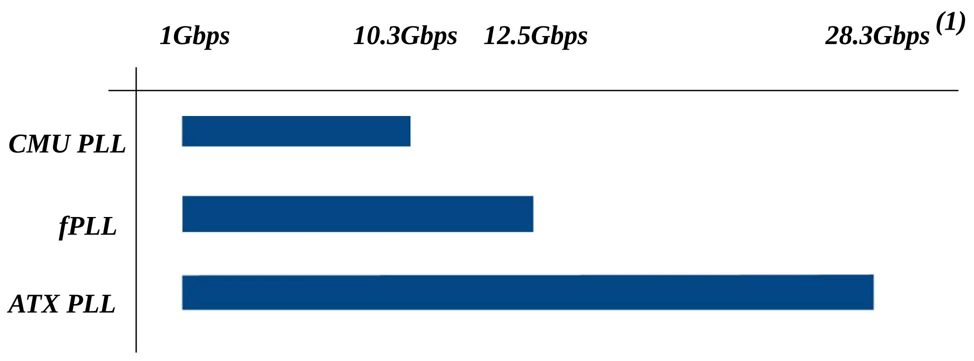
- L-Tile/H-Tile Layout in Intel Stratix 10 Device Variants on page 8
- Channel Bonding on page 299

3.1. PLLs

Table 127. Transmit PLLs in Intel Stratix 10 Devices

PLL Type	Characteristics
Advanced Transmit (ATX) PLL	• Best jitter performance • LC tank based voltage controlled oscillator (VCO) • Supports fractional synthesis mode (in cascade mode only) • Used for both bonded and non-bonded channel configurations
Fractional PLL (fPLL)	• Ring oscillator based VCO • Supports fractional synthesis mode • Used for both bonded and non-bonded channel configurations
Clock Multiplier Unit (CMU) PLL or Channel PLL (41)	• Ring oscillator based VCO • Used as an additional clock source for non-bonded applications

Figure 135. Transmit PLL Recommendation Based on Data Rates



Note:
1. This maximum data rate is for H-Tile. For L-Tile, it will be 26.6 Gbps

Related Information

Using PLLs and Clock Networks on page 304

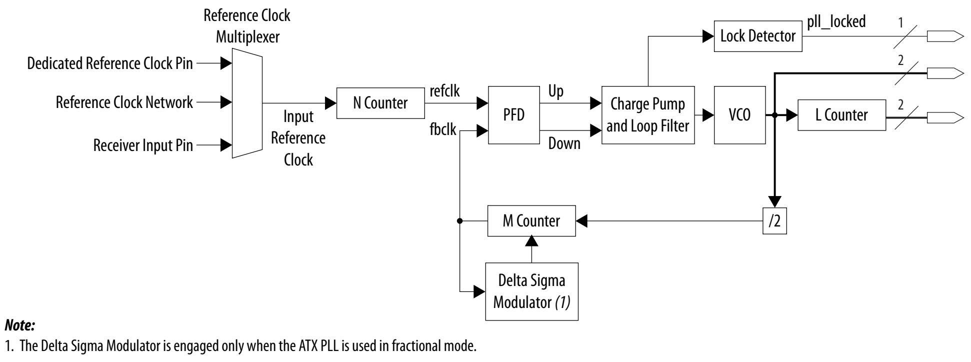
3.1.1. ATX PLL

The ATX PLL contains LC tank-based voltage controlled oscillators (VCOs). These LC VCOs have different frequency ranges to support a continuous range of operation.

When driving the Transceiver directly, the ATX PLL only supports the integer mode. In cascade mode, the ATX PLL only supports the fractional mode.

(41) The CMU PLL or Channel PLL of channel 1 and channel 4 can be used as a transmit PLL or as a clock data recovery (CDR) block. The channel PLL of all other channels (0, 2, 3, and 5) can only be used as a CDR.

Figure 136. ATX PLL Block Diagram


Reference Clock Multiplexer
Dedicated Reference Clock Pin
Reference Clock Network
Receiver Input Pin
Input Reference Clock
N Counter
refclk
fbclk
PFD
Up
Down
Charge Pump and Loop Filter
VCO
Lock Detector
pll_locked
1
2
2
L Counter
/2
M Counter
Delta Sigma Modulator (1)
Note:
1. The Delta Sigma Modulator is engaged only when the ATX PLL is used in fractional mode.

Input Reference Clock

This is the dedicated input reference clock source for the ATX PLL.

The input reference clock can be driven from one of the following sources. The sources are listed in order of performance, with the first choice giving the highest performance.

- Dedicated reference clock pin
- Reference clock network (with two new high quality reference clock lines)
- Receiver input pin

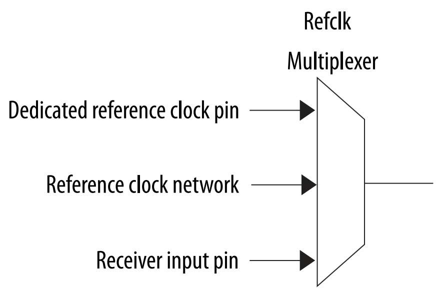
The input reference clock is a differential signal. Intel recommends using the dedicated reference clock pin as the input reference clock source for the best jitter performance. The input reference clock must be stable and free-running at device power-up for proper PLL operation and PLL calibration. If the reference clock is not available at device power-up, then you must recalibrate the PLL when the reference clock is available.

Note: The ATX PLL calibration is clocked by the OSC_CLK_1 clock, which must be stable and available for calibration to proceed. Refer to the *Calibration* chapter and *Intel Stratix 10 GX, MX, and SX Device Family Pin Connection Guidelines* for more details about the OSC_CLK_1 clock.

Reference Clock Multiplexer

The reference clock (`refclk`) multiplexer selects the reference clock to the PLL from the various reference clock sources available.

Figure 137. Reference Clock Multiplexer



N Counter

The N counter divides the `refclk` mux's output. The division factors supported are 1, 2, 4, and 8.

Phase Frequency Detector (PFD)

The reference clock(`refclk`) signal at the output of the N counter block and the feedback clock (`fbclk`) signal at the output of the M counter block are supplied as inputs to the PFD. The output of the PFD is proportional to the phase difference between the `refclk` and `fbclk` inputs. It is used to align the `refclk` signal at the output of the N counter to the feedback clock (`fbclk`) signal. The PFD generates an "Up" signal when the reference clock's falling edge occurs before the feedback clock's falling edge. Conversely, the PFD generates a "Down" signal when the feedback clock's falling edge occurs before the reference clock's falling edge.

Charge Pump and Loop Filter

The PFD output is used by the charge pump and loop filter (CP and LF) to generate a control voltage for the VCO. The charge pump translates the "Up" or "Down" pulses from the PFD into current pulses. The current pulses are filtered through a low pass filter into a control voltage that drives the VCO frequency. The charge pump, loop filter, and VCO settings determine the bandwidth of the ATX PLL.

Lock Detector

The lock detector block indicates when the reference clock and the feedback clock are phase aligned in integer mode, and frequency aligned in fractional mode. The lock detector generates an active high `pll_locked` signal to indicate that the PLL is locked to its input reference clock.

Voltage Controlled Oscillator

The voltage controlled oscillator (VCO) used in the ATX PLL is LC tank based. The output of charge pump and loop filter serves as an input to the VCO. The output frequency of the VCO depends on the input control voltage.

L Counter

The L counter divides the differential clocks generated by the ATX PLL. The L counter is not in the feedback path of the PLL.

M Counter

The M counter's output is the same frequency as the N counter's output. The VCO frequency is governed by the equation:

VCO freq = 2 * M * input reference clock/N

An additional divider divides the high speed serial clock output of the VCO by 2 before it reaches the M counter.

The M counter supports division factors in a continuous range from 8 to 127 in integer frequency synthesis mode and 11 to 123 in fractional mode.

Delta Sigma Modulator

The fractional mode is only supported when the ATX PLL is configured as a cascade source for OTN and SDI protocols. The delta sigma modulator is used in fractional mode. It modulates the M counter divide value over time so that the PLL can perform fractional frequency synthesis. In fractional mode, the M value is as follows:

M (integer) + $K/2^{32}$, where K is the Fractional multiply factor (K) in the ATX PLL IP Core Parameter Editor.

The legal values of K are greater than 1% and less than 99% of the full range of 2^{32} and can only be manually entered in the ATX PLL IP Core Parameter Editor in the Intel Quartus Prime Pro Edition.

The output frequencies cannot be exact when the ATX PLL is configured in fractional mode. Due to the K value 32-bit resolution, translating to 1.63 Hz step for a 7 GHz VCO frequency, not all desired fractional values can be achieved exactly.

Related Information

- Calibration on page 433
- ATX PLL IP Core - Parameters, Settings, and Ports on page 260
- ATX PLL on page 251
- Intel Stratix 10 Device Family Pin Connection Guidelines

3.1.1.1. ATX PLL to fPLL Spacing Requirements

When using ATX PLLs and fPLLs operating at the same VCO frequency or within 100 MHz, you must observe the spacing requirements listed in the following table.

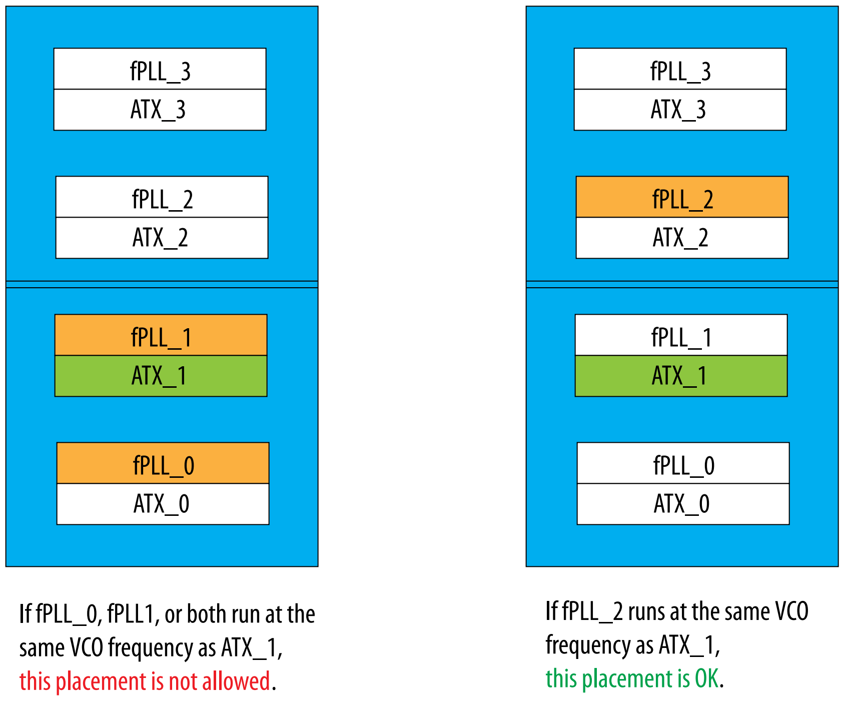
Table 128. ATX PLL to fPLL Spacing Requirements

ATX PLL to fPLL Spacing	Spacing Requirement
ATX PLL to fPLL spacing	• Skip 1 ATX PLL when ATX PLL and fPLL VCO frequencies are within 100 MHz and the fPLL's L counter = 1 OR • None if fPLL L counter ≥ 2[42]

There is no ATX PLL placement restriction between two different tiles.

[42] You can find the L Counter value in the **Advanced Parameter** tab of the fPLL IP.

Figure 138. ATX PLL to fPLL Placement Example


fPLL_3
ATX_3
fPLL_2
ATX_2
fPLL_1
ATX_1
fPLL_0
ATX_0
If fPLL_0, fPLL1, or both run at the same VCO frequency as ATX_1, this placement is not allowed.
fPLL_3
ATX_3
fPLL_2
ATX_2
fPLL_1
ATX_1
fPLL_0
ATX_0
If fPLL_2 runs at the same VCO frequency as ATX_1, this placement is OK.

3.1.1.2. Using the ATX PLL for GXT Channels

An ATX PLL can act as the transmit PLL for up to 6 GXT channels on H-Tile (4 GXT channels on L-Tile) through a dedicated clock network. This is accomplished by instantiating 3 ATX PLL instances:

- Main ATX PLL is configured as a transmit PLL
- Adjacent top ATX PLL is configured as a GXT clock buffer, passing the center ATX PLL's serial clock to the adjacent GXT channels
- Adjacent bottom ATX PLL is configured as a GXT clock buffer, passing the center ATX PLL's serial clock to the adjacent GXT channels

No GXT clock buffer ATX PLLs are needed, if 2 GXT channels are required and they are adjacent (channels 0 and 1 in a bank and the transmit ATX PLL is located in the bottom of the bank or channels 3 and 4 in a bank and the transmit ATX PLL is located in the top of the bank) to the transmit ATX PLL. The same rule applies, if a single GXT channel is required.

A single GXT clock buffer ATX PLL is needed, if 4 GXT channels are required and they are adjacent (channels 0, 1, 3 and 4 in a bank or channels 0 and 1 in a bank and channels 3 and 4 in the bank below). The transmit ATX PLL can be the ATX PLL adjacent to the top or bottom 2 GXT channels. The same rule applies, if 3 GXT channels are required.

The reference clock for GXT channels must be located in the same triplet as the master ATX PLL.

Note: For L-Tile, you can have up to eight GXT channels per tile: channels 0/1, 3/4 in banks 1D/1L/1F/1N, 4D/4L/4F/4N (as applicable by package). For H-Tile, you can have up to 16 GXT channels per tile.

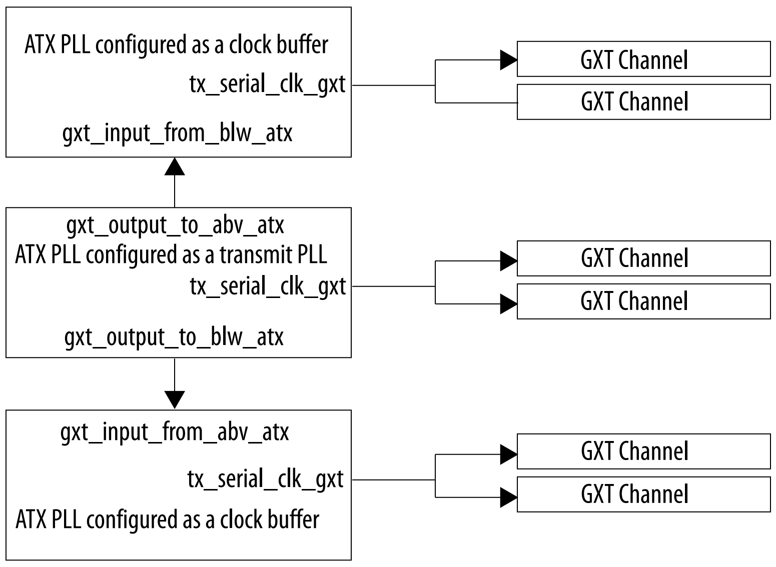
There are 5 ports in the Intel Stratix 10 L-Tile/H-Tile ATX PLL IP to support GXT channels:

- `tx_serial_clk_gxt` output port on transmit and GXT clock buffer ATX PLLs. The `tx_serial_clk_gxt` connects to the `tx_serial_clk` port in the Intel Stratix 10 L-Tile/H-Tile Native PHY IP
- `gxt_output_to_abv_atx` output port on ATX PLLs configured as transmit PLLs, outputs the GXT serial clock to the above ATX PLL configured as a GXT clock buffer
- `gxt_output_to_blw_atx` output port on ATX PLLs configured as transmit PLLs, outputs the GXT serial clock to the below ATX PLL configured as a GXT clock buffer
- `gxt_input_from_blw_atx` input port on ATX PLLs configured as GXT clock buffer inputs the GXT serial clock from the below ATX PLL configured as a transmit PLL
- `gxt_input_from_abv_atx` input port on ATX PLLs configured as GXT clock buffer inputs the GXT serial clock from the above ATX PLL configured as a transmit PLL

Port `gxt_output_to_abv_atx` of the transmit ATX PLL needs to be connected to port `gxt_input_from_blw_atx` of the above GXT clock buffer ATX PLL.

Port `gxt_output_to_blw_atx` of the transmit ATX PLL needs to be connected to port `gxt_input_from_abv_atx` of the below GXT clock buffer ATX PLL.

Figure 139. ATX PLL GXT Clock Connection



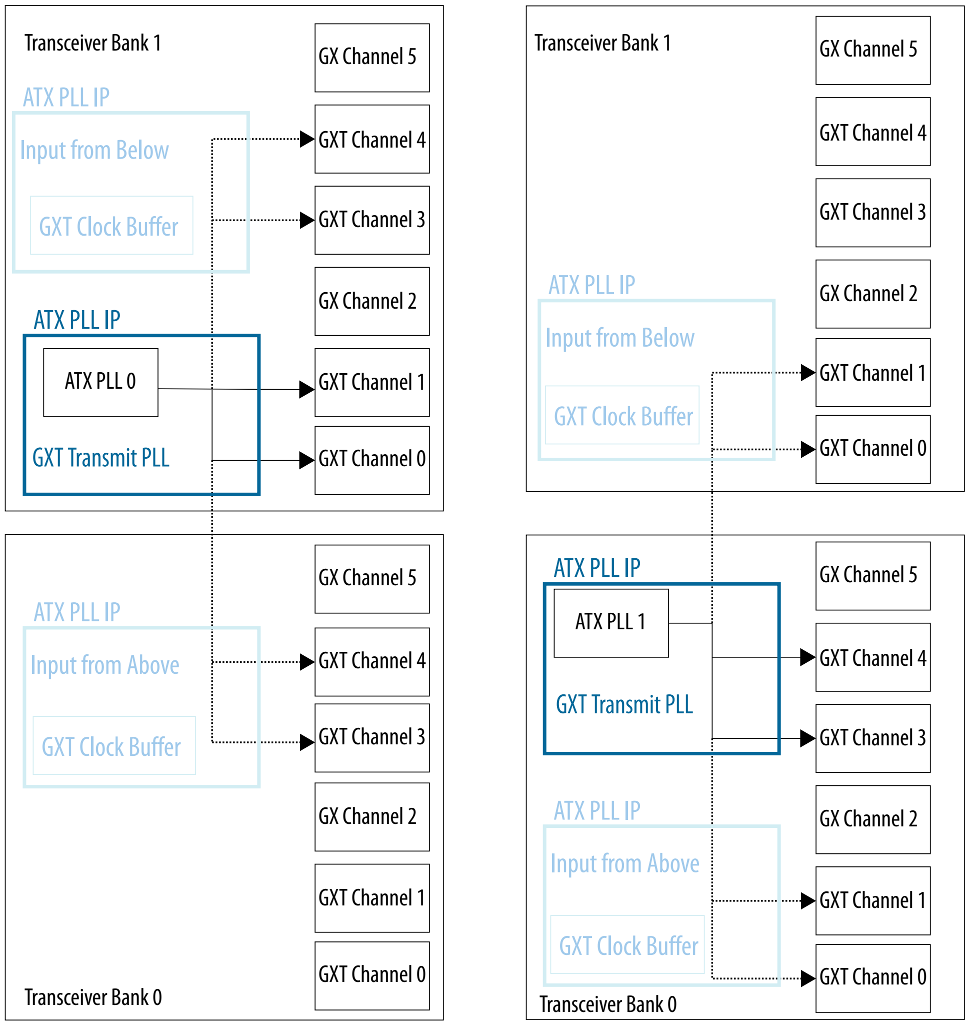
The ATX PLL can be configured in the following GXT modes:

- GXT transmit PLL with GXT clocks to adjacent GXT channels
- GXT transmit PLL with GXT clocks to GXT clock buffer ATX PLLs
- GXT transmit PLL with GXT clocks to adjacent GXT channels and GXT clock buffer ATX PLLs
- GXT clock buffer ATX PLL

To configure an ATX PLL as a GXT transmit PLL with GXT clocks to adjacent GXT channels:

1. Set the **ATX PLL operation mode** drop-down as **GXT mode**.
2. Select the **Enable GXT local clock output port (tx_serial_clk_gxt)** checkbox.
3. Set the **GXT output clock source** drop-down as **Local ATX PLL.**
4. Set the ATX PLL input reference clock and datarate parameters.

Figure 140. Main and Adjacent ATX PLL IP Instances to Drive 6 GXT Channels



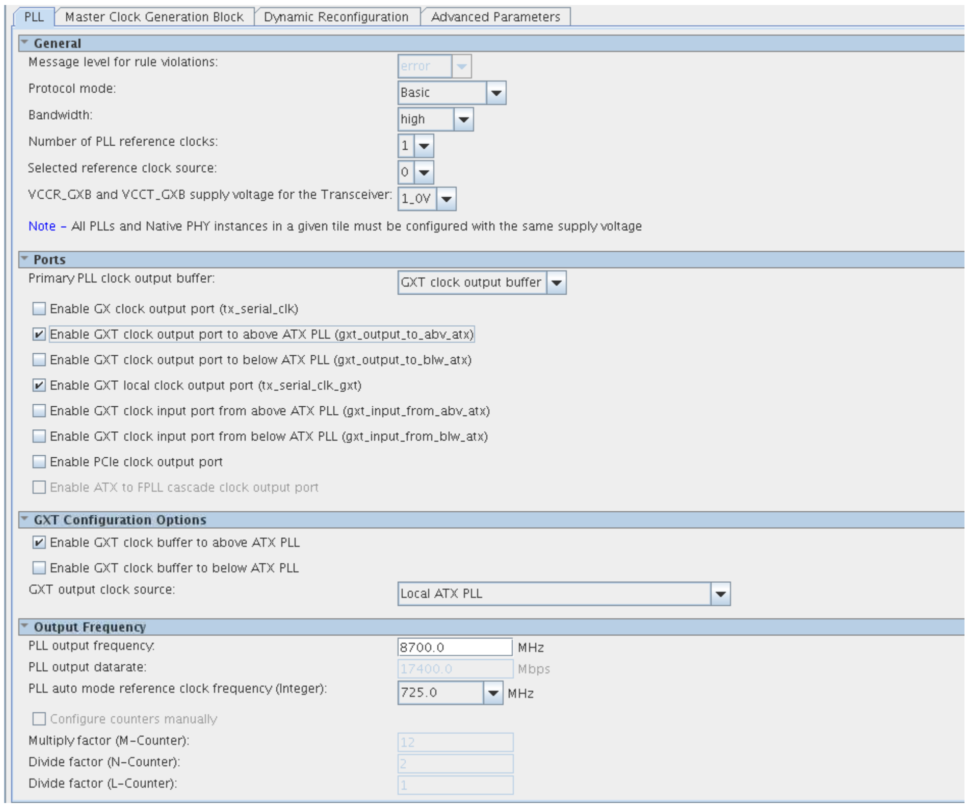
To configure an ATX PLL as a GXT transmit PLL with GXT clocks to GXT clock buffer ATX PLLs:

1. Set the **ATX PLL operation mode** drop-down as **GXT mode**.
2. Select the **Enable GXT clock output port to above ATX PLL (gxt_output_to_abv_atx)**, **Enable GXT clock output port to below ATX PLL (gxt_output_to_blw_atx)**, or both checkbox.
3. Select the **Enable GXT clock buffer to above ATX PLL**, **Enable GXT clock buffer to above ATX PLL**, or both checkbox.
4. Set the **GXT output clock source** drop-down as **Disabled**.
5. Set the ATX PLL input reference clock and datarate parameters.

To configure an ATX PLL as a GXT transmit PLL with GXT clocks to adjacent GXT channels and GXT clock buffer ATX PLLs:

1. Set the **ATX PLL operation mode** drop-down as **GXT mode**.
2. Select the **Enable GXT local clock output port (tx_serial_clk_gxt)** checkbox.
3. Set the **GXT output clock source** drop-down as **Local ATX PLL.**
4. Select the **Enable GXT output port** to **above ATX PLL (gxt_output_to_abv_atx)**, **Enable GXT output port**, or both to **below ATX PLL (gxt_output_to_blw_atx)** checkbox.
5. Select the **Enable GXT clock buffer** to **above ATX PLL**, **Enable GXT clock buffer**, or both to **above ATX PLL** checkbox.
6. Set the ATX PLL input reference clock and datarate parameters.

Figure 141. ATX PLL IP Parameter Details for Main ATX PLL IP



General

Message level for rule violations: error
Protocol mode: Basic
Bandwidth: high
Number of PLL reference clocks: 1
Selected reference clock source: 0
VCCR_GXB and VCCT_GXB supply voltage for the Transceiver: 1_0V

Note – All PLLs and Native PHY instances in a given tile must be configured with the same supply voltage

Ports

Primary PLL clock output buffer: GXT clock output buffer

- [ ] Enable GX clock output port (tx_serial_clk)
- [✓] Enable GXT clock output port to above ATX PLL (gxt_output_to_abv_atx)
- [ ] Enable GXT clock output port to below ATX PLL (gxt_output_to_blw_atx)
- [✓] Enable GXT local clock output port (tx_serial_clk_gxt)
- [ ] Enable GXT clock input port from above ATX PLL (gxt_input_from_abv_atx)
- [ ] Enable GXT clock input port from below ATX PLL (gxt_input_from_blw_atx)
- [ ] Enable PCIe clock output port
- [ ] Enable ATX to FPLL cascade clock output port

GXT Configuration Options

- [✓] Enable GXT clock buffer to above ATX PLL
- [ ] Enable GXT clock buffer to below ATX PLL

GXT output clock source: Local ATX PLL

Output Frequency

PLL output frequency: 8700.0 MHz
PLL output datarate: 17400.0 Mbps
PLL auto mode reference clock frequency (Integer): 725.0 MHz

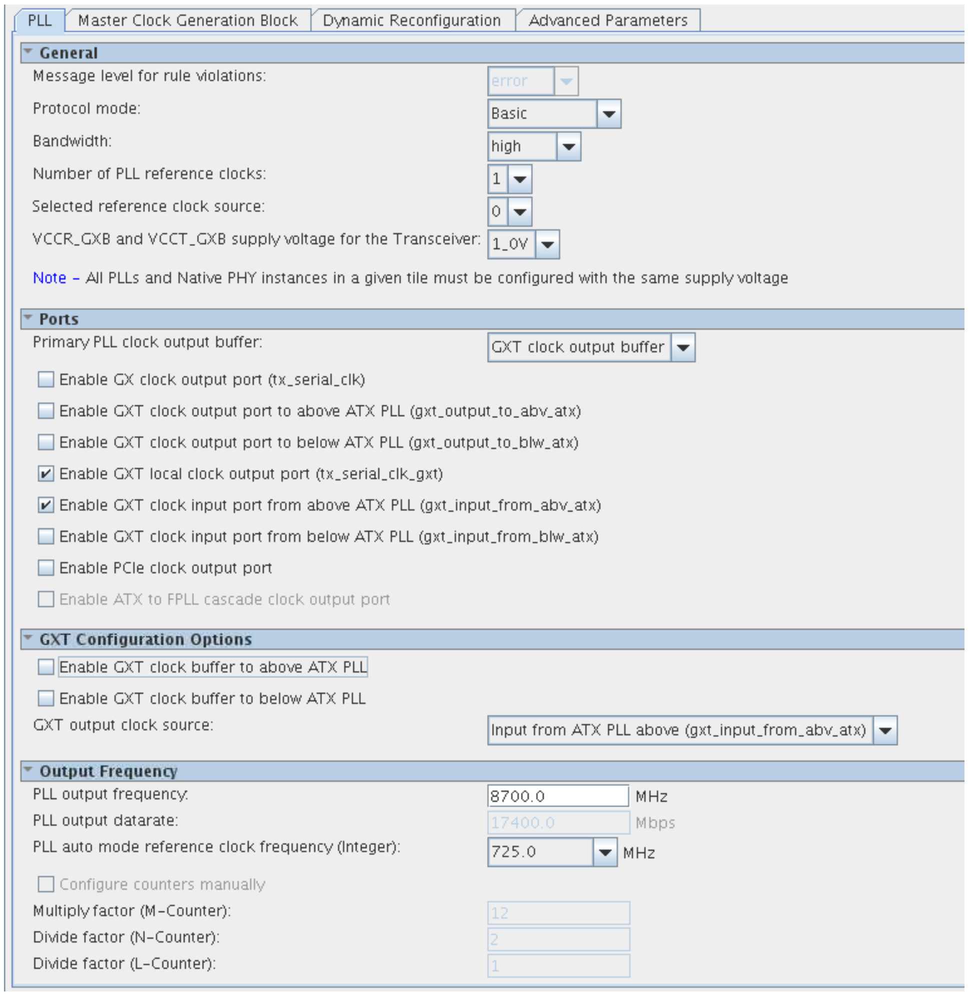
- [ ] Configure counters manually

Multiply factor (M-Counter): 12
Divide factor (N-Counter): 2
Divide factor (L-Counter): 1

To configure an ATX PLL as a GXT clock buffer ATX PLL:

1. Set the **ATX PLL operation mode** drop-down as **GXT mode.**
2. Select the **Enable GXT local clock output port (tx_serial_clk_gxt)** checkbox.
3. Set the **GXT output clock source** drop-down as **Input from ATX PLL above (gxt_input_from_abv_atx)** or **Input from ATX PLL below (gxt_input_from_blw_atx)**.
4. Tie off the `pll_refclk0` pin to REFCLK pin, if the GXT clock buffer ATX PLL is not reconfigured to a GXT transmit PLL or GX transmit PLL.

Figure 142. ATX PLL IP Parameter Details for Clock Buffer ATX PLL IP



General

Message level for rule violations:	error
Protocol mode:	Basic
Bandwidth:	high
Number of PLL reference clocks:	1
Selected reference clock source:	0
VCCR_GXB and VCCT_GXB supply voltage for the Transceiver:	1_0V

Note – All PLLs and Native PHY instances in a given tile must be configured with the same supply voltage

Ports

Primary PLL clock output buffer: GXT clock output buffer

- [ ] Enable GX clock output port (tx_serial_clk)
- [ ] Enable GXT clock output port to above ATX PLL (gxt_output_to_abv_atx)
- [ ] Enable GXT clock output port to below ATX PLL (gxt_output_to_blw_atx)
- [✓] Enable GXT local clock output port (tx_serial_clk_gxt)
- [✓] Enable GXT clock input port from above ATX PLL (gxt_input_from_abv_atx)
- [ ] Enable GXT clock input port from below ATX PLL (gxt_input_from_blw_atx)
- [ ] Enable PCIe clock output port
- [ ] Enable ATX to FPLL cascade clock output port

GXT Configuration Options

- [ ] Enable GXT clock buffer to above ATX PLL
- [ ] Enable GXT clock buffer to below ATX PLL

GXT output clock source: Input from ATX PLL above (gxt_input_from_abv_atx)

Output Frequency

PLL output frequency:	8700.0 MHz
PLL output datarate:	17400.0 Mbps
PLL auto mode reference clock frequency (Integer):	725.0 MHz

- [ ] Configure counters manually

Multiply factor (M-Counter):	12
Divide factor (N-Counter):	2
Divide factor (L-Counter):	1

An ATX PLL can be reconfigured between modes, but all needed ports must be enabled in the instance.

Note: An ATX PLL cannot be reconfigured from GX to GXT mode if the adjacent master CGB is being used.

Related Information

Steps to Perform Dynamic Reconfiguration on page 405

3.1.1.3. GXT Implementation Usage Restrictions for ATX PLL GX & MCGB

If configured as a Clock Buffer ATX PLL (selecting **Input from ATX PLL below/ above**), the ATX PLL cannot be used to drive GX serial output.

If configured as the Main ATX PLL (**Local ATX PLL output**) the ATX PLL MCGB cannot be used.

Neither of the MCGB's can be used when the Main ATX PLL and the Clock Buffer ATX PLL are utilized to drive GXT channels.

3.1.1.4. Instantiating the ATX PLL IP Core

The Intel Stratix 10 transceiver ATX PLL IP core provides access to the ATX PLLs in the hardware. One instance of the PLL IP core represents one ATX PLL in the hardware.

1. Open the Intel Quartus Prime Pro Edition software.
2. Click **Tools ➤ IP Catalog**.
3. In **IP Catalog**, under **Library ➤ Interface Protocol ➤ Transceiver PLL ➤** , select **Intel Stratix 10 L-Tile/H-Tile Transceiver ATX PLL** and click **Add**.
4. In the **New IP Instance** dialog box, provide the IP instance name.
5. Select the **Intel Stratix 10** device family.
6. Select the appropriate device and click **OK**.

The ATX PLL IP core **Parameter Editor** window opens.

3.1.1.5. ATX PLL IP Core - Parameters, Settings, and Ports

Table 129. ATX PLL IP Core - Configuration Options, Parameters, and Settings

Parameter	Range	Description
Message level for rule violations	**Error**	Specifies the messaging level to use for parameter rule violations. • Error—Causes all rule violations to prevent IP generation.
Protocol mode	**Basic** **PCIe Gen1** **PCIe Gen2** **PCIe Gen3** **SDI_cascade** **OTN_cascade**	Governs the internal setting rules for the VCO. This parameter is not a preset. You must set all other parameters for your protocol. **SDI_cascade** and **OTN_cascade** are supported cascade mode configurations and enables "ATX to FPLL cascade output port", "manual configuration of counters" and "fractional mode". Protocol mode **SDI_cascade** enables **SDI cascade** rule checks and **OTN_cascade** enables **OTN cascade** rule checks.
Bandwidth	**Low** **Medium** **High**	Specifies the VCO bandwidth. Higher bandwidth reduces PLL lock time, at the expense of decreased jitter rejection.
Number of PLL reference clocks	**1 to 5**	Specifies the number of input reference clocks for the ATX PLL. You can use this parameter for data rate reconfiguration.

continued...

Parameter	Range	Description
Selected reference clock source	**0 to 4**	Specifies the initially selected reference clock input to the ATX PLL.
VCCR_GXB and VCCT_GXB supply voltage for the Transceiver	**1_0V, and 1_1V** (43)	Selects the VCCR_GXB and VCCT_GXB supply voltage for the Transceiver.
Primary PLL clock output buffer	**GX clock output buffer/GXT clock output buffer**	Specifies which PLL output is active initially. If GX is selected "Enable PLL GX clock output port" should be enabled. If GXT is selected "Enable PLL GXT clock output port" should be enabled.
Enable GX clock output port (tx_serial_clk)	**On/Off**	GX clock output port feeds x1 clock lines. Must be selected for PLL output frequency smaller than 8.7 GHz. If GX is selected in "Primary PLL clock output buffer", the port should be enabled as well.
Enable GXT clock output port to above ATX PLL (gxt_output_to_abv_atx)	**On/Off**	GXT clock output to above ATX PLL to feed the dedicated high speed clock lines. Must be selected for PLL output frequency greater than 8.7 GHz. If GXT is selected in "Primary PLL clock output buffer", the port should be enabled as well.
Enable GXT clock output port to below ATX PLL (gxt_output_to_blw_atx)	**On/Off**	GXT clock output to below ATX PLL to feed the dedicated high speed clock lines. Must be selected for PLL output frequency greater than 8.7 GHz. If GXT is selected in "Primary PLL clock output buffer", the port should be enabled as well.
Enable GXT local clock output port (tx_serial_clk_gxt)	**Off**	GXT local clock output port to feed the dedicated high speed clock lines. Must be selected for PLL output frequency greater than 8.7 GHz. If GXT is selected in "Primary PLL clock output buffer", the port should be enabled as well.
Enable GXT clock input port from above ATX PLL (gxt_input_from_abv_atx)	**On/Off**	GXT clock input port from Above ATX PLL port to drive the dedicated high speed clock lines. Must be selected for PLL input frequency greater than 8.7 GHz. If GXT is selected in "Primary PLL clock input buffer", the port should be enabled as well.
Enable GXT clock input port from below ATX PLL (gxt_input_from_blw_atx)	**On/Off**	GXT clock input port from Below ATX PLL port to drive the dedicated high speed clock lines. Must be selected for PLL input frequency greater than 8.7 GHz. If GXT is selected in "Primary PLL clock input buffer", the port should be enabled as well.
Enable PCIe clock output port	**On/Off**	This is the 500 MHz fixed PCIe clock output port and is intended for PIPE mode. The port should be connected to "pipe_hclk_in" port of the Native PHY IP.
Enable ATX to FPLL cascade clock output port	**On/Off**	Enables the ATX to FPLL cascade clock output port. This option selects Fractional mode and "Configure counters manually" option. OTN_cascade protocol mode enables OTN rule checks and SDI_cascade mode enables SDI rule checks
Enable GXT clock buffer to above ATX PLL	**On/Off**	GXT clock output port to drive the above ATX PLL. Must be selected for output frequency greater than 8.7 GHz. If GXT is selected in "Primary PLL clock input buffer", the port should be enabled as well.

continued...

(43) Refer to the *Intel Stratix 10 Device Datasheet* for details about the minimum, typical, and maximum supply voltage specifications.

Parameter	Range	Description
Enable GXT clock buffer to below ATX PLL	**On/Off**	GXT clock output port to drive the below ATX PLL. Must be selected for output frequency greater than 8.7 GHz. If GXT is selected in "Primary PLL clock input buffer", the port should be enabled as well.
GXT output clock source	**Local ATX PLL** **Input from ATX PLL above (gxt_input_from _abv_atx)** **Input from ATX PLL above (gxt_input_from _blw_atx)** **Disabled**	Specifies which GXT clock output is active based on GXT 3:1 mux selection. The possible options are input from above/ below ATX PLLs OR local ATX PLL.
PLL output frequency	Refer to the Transceiver Performance Specifications section of the Intel Stratix 10 Device Datasheet	Use this parameter to specify the target output frequency for the PLL.
PLL output datarate	Refer to the GUI	Specifies the target datarate for which the PLL is used.
PLL auto mode reference clock frequency (Integer)	Refer to the GUI	Selects the auto mode input reference clock frequency for the PLL (Integer).
Configure counters manually	**On/Off**	Enables manual control of PLL counters. Available only in ATX to FPLL cascade configuration
Multiply factor (M-Counter)	**Read only** For OTN_cascade or SDI_cascade, refer to the GUI	Displays the M-counter value. Specifies the M-counter value (In SDI_cascade or OTN_cascade Protocol mode only).
Divide factor (N-Counter)	**Read only** For SDI_cascade or OTN_cascade, refer to the GUI	Displays the N-counter value. For SDI_cascade or OTN_cascade, refer to the GUI.
Divide factor (L-Counter)	**Read only**	Displays the L-counter value.

Table 130. ATX PLL IP Core - Master Clock Generation Block Parameters and Settings

Parameter	Range	Description
Include Master Clock Generation Block (44)	**On/Off**	When enabled, includes a master CGB as a part of the ATX PLL IP core. The PLL output drives the Master CGB. This is used for x6/x24 bonded and non-bonded modes.
Clock division factor	**1, 2, 4, 8**	Divides the master CGB clock input before generating bonding clocks.
Enable x24 non-bonded high-speed clock output port	**On/Off**	Enables the master CGB serial clock output port used for x24 non-bonded modes.
Enable PCIe clock switch interface	**On/Off**	Enables the control signals for the PCIe clock switch circuitry. Used for PCIe clock rate switching.

continued...

(44) Manually enable the MCGB for bonding applications.

Parameter	Range	Description
Enable mcgb_rst and mcgb_rst_stat ports	**On/Off**	Enables the `mcgb_rst` and `mcgb_rst_stat` ports. These ports must be disabled for all PCIe configurations when using L-Tile or H-Tile devices.
Number of auxiliary MCGB clock input ports	**0, 1**	Auxiliary input is used to implement the PCIe Gen3 PIPE protocol. It is not available in fPLL.
MCGB input clock frequency	**Read only**	Displays the master CGB's input clock frequency. This parameter is not settable by user.
MCGB output data rate.	**Read only**	Displays the master CGB's output data rate. This parameter is not settable by user. The value is calculated based on "MCGB input clock frequency" and "Master CGB clock division factor".
Enable bonding clock output ports	**On/Off**	Enables the **tx_bonding_clocks** output ports of the master CGB used for channel bonding. This option should be turned ON for bonded designs.
PMA interface width	**8, 10, 16, 20, 32, 40, 64**	Specifies PMA-PCS interface width. Match this value with the PMA interface width selected for the Native PHY IP core. You must select a proper value for generating bonding clocks for the Native PHY IP core.

Table 131. ATX PLL IP Core - Dynamic Reconfiguration

Parameter	Range	Description
Enable dynamic reconfiguration	**On/Off**	Enables the dynamic reconfiguration interface.
Enable Native PHY Debug Master Endpoint	**On/Off**	When enabled, the PLL IP includes an embedded Native PHY Debug Master Endpoint that connects internally Avalon memory-mapped interface slave. The NPDME can access the reconfiguration space of the transceiver. It can perform certain test and debug functions via JTAG using the System Console. This option requires you to enable the "Share reconfiguration interface" option for configurations using more than 1 channel and may also require that a `jtag_debug` link be included in the system.
Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE	**On/Off**	When enabled, the **reconfig_waitrequest** does not indicate the status of Avalon memory-mapped interface arbitration with PreSICE. The Avalon memory-mapped interface arbitration status is reflected in a soft status register bit. This feature requires that the "Enable control and status registers" feature under "Optional Reconfiguration Logic" be enabled.
Enable capability registers	**On/Off**	Enables capability registers, which provide high level information about the transceiver PLL's configuration.
Set user-defined IP identifier	**0 to 255**	Sets a user-defined numeric identifier that can be read from the user_identifer offset when the capability registers are enabled.
Enable control and status registers	**On/Off**	Enables soft registers for reading status signals and writing control signals on the phy interface through the embedded debug. Available signals include `pll_cal_busy`, **pll_locked** and **pll_powerdown**.
Configuration file prefix	**altera_xcvr_atx _pll_s10**	Specifies the file prefix to use for generated configuration files when enabled. Each variant of the IP should use a unique prefix for configuration files.

continued...

Parameter	Range	Description
Generate SystemVerilog package file	**On/Off**	When enabled, the IP generates a SystemVerilog package file named "(Configuration file prefix)_reconfig_parameters.sv" containing parameters defined with the attribute values needed for reconfiguration.
Generate C header file	**On/Off**	When enabled, the IP generates a C header file named "(Configuration file prefix)_reconfig_parameters.h" containing macros defined with the attribute values needed for reconfiguration.
Generate MIF (Memory Initialize File)	**On/Off**	When enabled, the IP generates an MIF (Memory Initialization File) named "(Configuration file prefix)_reconfig_parameters.mif". The MIF file contains the attribute values needed for reconfiguration in a data format.
Enable multiple reconfiguration profiles	**On/Off**	When enabled, you can use the GUI to store multiple configurations. The IP generates reconfiguration files for all of the stored profiles. The IP also checks your multiple reconfiguration profiles for consistency to ensure you can reconfigure between them.
Enable embedded reconfiguration streamer	**On/Off**	Enables the embedded reconfiguration streamer, which automates the dynamic reconfiguration process between multiple predefined configuration profiles.
Generate reduced reconfiguration files	**On/Off**	When enabled, the Native PHY generates reconfiguration report files containing only the attributes or RAM data that are different between the multiple configured profiles.
Number of reconfiguration profiles	**1 to 8**	Specifies the number of reconfiguration profiles to support when multiple reconfiguration profiles are enabled.
Store current configuration to profile:	**0 to 7**	Selects which reconfiguration profile to store when clicking the "Store profile" button.

Table 132. ATX PLL IP Core - Ports

Port	Direction	Clock Domain	Description
`pll_refclk0`	Input	N/A	Reference clock input port 0. There are a total of five reference clock input ports. The number of reference clock ports available depends on the **Number of PLL reference clocks** parameter.
`pll_refclk1`	Input	N/A	Reference clock input port 1.
`pll_refclk2`	Input	N/A	Reference clock input port 2.
`pll_refclk3`	Input	N/A	Reference clock input port 3.
`pll_refclk4`	Input	N/A	Reference clock input port 4.
`mcgb_aux_clk0`	Input	N/A	Used for PCIe implementation to switch between fPLL and ATX PLL during link speed negotiation.
`pcie_sw[1:0]`	Input	Asynchronous	2-bit rate switch control input used for PCIe protocol implementation.
`gxt_input_from_abv_atx`	Input	N/A	GXT clock input from above ATX PLL to drive the dedicated high speed clock lines.
`gxt_input_from_blw_atx`	Input	N/A	GXT clock input from below ATX PLL to drive the dedicated high speed clock lines.

continued...

Port	Direction	Clock Domain	Description
`mcgb_rst`	Input	N/A	Resets the master CGB. This port must be disabled for all PCIe configurations when using L-Tile or H-Tile devices.
`tx_serial_clk`	Output	N/A	High speed serial clock output port for GX channels. Represents the x1 clock network.
`pll_locked`	Output	Asynchronous	Active high status signal which indicates if the PLL is locked.
`pll_pcie_clk`	Output	N/A	Used for PCIe.
`pll_cal_busy`	Output	Asynchronous	Status signal which is asserted high when PLL calibration is in progress. OR this signal with **tx_cal_busy** port before connecting to the reset controller IP.
`tx_bonding_clocks[5:0]`	Output	N/A	Optional 6-bit bus which carries the low speed parallel clock outputs from the master CGB. Each transceiver channel in a bonded group has this 6-bit bus. Used for channel bonding, and represents the x6/x24 clock network.
`mcgb_serial_clk`	Output	N/A	High speed serial clock output for x6/x24 non-bonded configurations.
`pcie_sw_done[1:0]`	Output	Asynchronous	2-bit rate switch status output used for PCIe protocol implementation.
`atx_to_fpll_cascade_clk`	Output	N/A	The ATX PLL output clock is used to drive fPLL reference clock input (only available in SDI_cascade or OTN_cascade protocol mode).
`tx_serial_clk_gxt`	Output	N/A	GXT clock output to drive the dedicated high speed clock lines.
`gxt_output_to_abv_atx`	Output	N/A	GXT clock output to above ATX PLL to drive the dedicated high speed clock lines.
`gxt_output_to_blw_atx`	Output	N/A	GXT clock output to below ATX PLL to drive the dedicated high speed clock lines.
`mcgb_rst_stat`	Output	N/A	Status signal for the master CGB. This port must be disabled for all PCIe configurations when using L-Tile or H-Tile devices.

Related Information

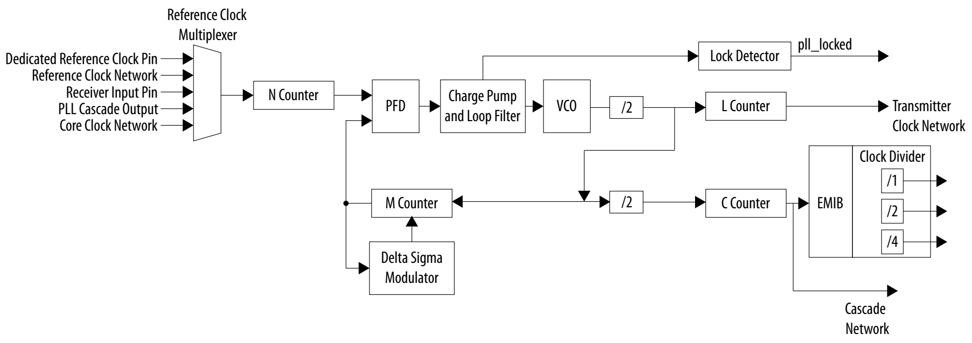
- Calibration on page 433
- Avalon Interface Specifications
  The ports related to reconfiguration are compliant with the Avalon Specification. Refer to the Avalon Specification for more details about these ports.
- Intel Stratix 10 Device Datasheet

3.1.2. fPLL

The fractional PLL (fPLL) is used for generating clock frequencies for data rates up to 12.5 Gbps. It can support both integer and fractional frequency synthesis. The fPLL can be used as a transmit PLL for transceiver applications. The fPLL can be cascaded from either the ATX or another fPLL, or it can be used to drive the core clock network. There are two fPLLs in each transceiver bank.

PLL cascading enables additional flexibility in terms of reference clock selection.

Figure 143. fPLL Block Diagram



Input Reference Clock

This is the dedicated input reference clock source for the fPLL.

The input reference clock can be driven from one of the following sources. The sources are listed in order of performance, with the first choice giving the best jitter performance.

- Dedicated reference clock pin
- Reference clock network
- Receiver input pin
- PLL cascade output
- Core clock network

Note: Each core clock network reference clock pin can only drive fPLLs located on a single tile.

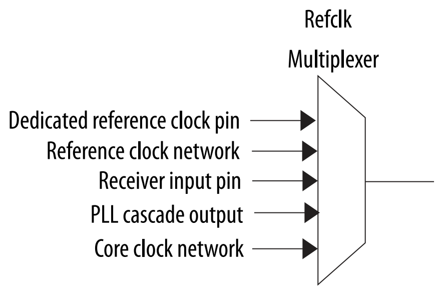
The input reference clock is a differential signal. Intel recommends using the dedicated reference clock pin as the input reference clock source for best jitter performance. The input reference clock must be stable and free-running at device power-up for proper PLL operation. If the reference clock is not available at device power-up, then you must recalibrate the PLL when the reference clock is available.

Note: The fPLL calibration is clocked by the `OSC_CLK_1` clock, which must be stable and available for the calibration to proceed. Refer to the *Calibration* section for details about PLL calibration and `OSC_CLK_1` clock.

fPLL Reference Clock Multiplexer

The reference clock (`refclk`) mux selects the reference clock to the PLL from the various available reference clock sources.

Figure 144. Reference Clock Multiplexer



N Counter

The N counter divides the reference clock (`refclk`) mux's output. The N counter division helps lower the loop bandwidth or reduce the frequency within the phase frequency detector's (PFD) operating range. The N counter supports division factors from 1 to 32.

Phase Frequency Detector

The reference clock (`refclk`) signal at the output of the N counter block and the feedback clock (`fbclk`) signal at the output of the M counter block are supplied as inputs to the PFD. The output of the PFD is proportional to the phase difference between the `refclk` and `fbclk` inputs. The PFD aligns the `fbclk` to the `refclk`. The PFD generates an "Up" signal when the reference clock's falling edge occurs before the feedback clock's falling edge. Conversely, the PFD generates a "Down" signal when the feedback clock's falling edge occurs before the reference clock's falling edge.

Charge Pump and Loop Filter (CP + LF)

The PFD output is used by the charge pump and loop filter to generate a control voltage for the VCO. The charge pump translates the "Up"/"Down" pulses from the PFD into current pulses. The current pulses are filtered through a low pass filter into a control voltage that drives the VCO frequency.

Voltage Controlled Oscillator

The fPLL has a ring oscillator based VCO. The VCO uses the following equation to transform the input control voltage into an adjustable frequency clock:

VCO freq = 2 * M * input reference clock/N

N and M are the N counter and M counter division factors.

L Counter

The L counter divides the VCO's clock output. When the fPLL acts as a transmit PLL, the output of the L counter drives the clock generation block (CGB), x1 clock lines and TX PMA.

M Counter

The M counter divides the VCO's clock output. The outputs of the M counter and N counter have same frequency. M counter range is 8 to 127 in integer mode and 11 to 124 in fractional mode.

Lock Detector

The lock detector block indicates when the reference clock and the feedback clock are phase aligned in integer mode, and frequency aligned in fractional mode. The lock detector generates an active high `pll_locked` signal to indicate that the PLL is locked to its input reference clock.

Delta Sigma Modulator

The delta sigma modulator is used in fractional mode. It modulates the M counter divide value over time so that the PLL can perform fractional frequency synthesis. In fractional mode, the M value is as follows:

M (integer) + K/2^32, where K is the Fractional multiply factor (K) in the fPLL IP Core Parameter Editor.

The legal values of K are greater than 1% and less than 99% of the full range of 2^{32} and can only be manually entered in the fPLL IP Core Parameter Editor in the Intel Quartus Prime Pro Edition.

The output frequency resolution when the fPLL is configured in fractional mode varies with VCO frequency. A 7 GHz VCO frequency results in 1.63 Hz step per K value LSB.

C Counter

The fPLL C counter division factors range from 1 to 512.

Related Information

- Calibration on page 433
- fPLL IP Core - Parameters, Settings, and Ports on page 269

3.1.2.1. Instantiating the fPLL IP Core

The fPLL IP core for Intel Stratix 10 transceivers provides access to fPLLs in hardware. One instance of the fPLL IP core represents one fPLL in the hardware.

1. Open the Intel Quartus Prime Pro Edition.
2. Click **Tools ➤ IP Catalog**.
3. In **IP Catalog**, under **Library ➤ Interface Protocols ➤ Intel Stratix 10 L-Tile/H-Tile fPLL**, select **Intel Stratix 10 Transceiver fPLL** IP core and click **Add**.
4. In the **New IP Instance** dialog box, provide the IP instance name.
5. Select the **Intel Stratix 10** device family.
6. Select the appropriate device and click **OK**.

The fPLL IP core **Parameter Editor** window opens.

3.1.2.2. fPLL IP Core Constraints

To implement the fPLL IP core, you must adhere to the following constraints:

- You must use `create_clock` constraints on fPLL reference clocks on the project's top-level SDC file.
- Any SDC design constraints referring to transceiver clocks must be listed after the transceiver Native PHY SDC file constraints.
- fPLL output clocks have no phase relationship to the reference clock when utilizing the fPLL output clocks for core usage. The fPLL output clocks of the clock divider are still in phase with each other, however.

3.1.2.3. fPLL IP Core - Parameters, Settings, and Ports

Table 133. fPLL IP Core - Configuration Options, Parameters, and Settings

Parameters	Range	Description
fPLL Mode	**Core** **Cascade Source** **Transceiver**	Specifies the fPLL mode of operation. Select **Core** to use fPLL as a general purpose PLL to drive the FPGA core clock network. fPLL in **Core** mode does not support the dynamic reconfiguration feature. Select **Cascade Source** to connect an fPLL to another PLL as a cascading source. Select **Transceiver** to use an fPLL as a transmit PLL for the transceiver block.
Message level for rule violations	**Error/Warning**	Sets rule checking level Selecting "error" causes all rule violations to prevent IP generation. Selecting "warning" displays all rule violations as warnings and allows IP generation in spite of violations.
Protocol Mode	**Basic** **PCIe Gen1** **PCIe Gen2** **PCIe Gen3** **SDI_cascade** **OTN_cascade** **SATA GEN3** **HDMI**	Governs the internal setting rules for the VCO. This parameter is not a preset. You must set all parameters for your protocol.
Bandwidth	**Low** **Medium** **High**	Specifies the VCO bandwidth. Higher bandwidth reduces PLL lock time, at the expense of decreased jitter rejection.
Number of PLL reference clocks	**1 to 5**	Specify the number of input reference clocks for the fPLL.
Selected reference clock source	**0 to 4**	Specifies the initially selected reference clock input to the fPLL.
Enable fractional mode	**On/Off**	Enables the fractional frequency mode. This enables the PLL to output frequencies which are not integral multiples of the input reference clock.
VCCR_GXB and VCCT_GXB supply voltage for the Transceiver	**1_0V, and 1_1V** (45)	Specifies the Transceiver supply voltage.

continued...

(45) Refer to the *Intel Stratix 10Device Datasheet* for details about the minimum, typical, and maximum supply voltage specifications.

Parameters	Range	Description
PLL output frequency	**User defined**	Displays the target output frequency for the PLL.
PLL output datarate	**Read-only**	Displays the PLL datarate.
PLL integer reference clock frequency	**User defined**	Set the fPLL's reference clock frequency for clock synthesis.
Configure counters manually	**On/Off**	Selecting this option allows you to manually specify M, N, C and L counter values.
Multiply factor (M-counter)	**8 to 127 (integer mode)** **11 to 123 (fractional mode)**	Specifies the multiply factor (M-counter).
Divide factor (N-counter)	**1 to 31**	Specifies the divide factor (N-counter).
Divide factor (L-counter)	**1, 2, 4, 8**	Specifies the divide factor (L-counter).
Divide factor (C-counter)	**1** to **512**	Specifies the fPLL output clock frequency to the core when configured in core mode.

Table 134. fPLL—Master Clock Generation Block Parameters and Settings

Parameters	Range	Description
Include Master Clock Generation Block	**On/Off**	When enabled, includes a master CGB as a part of the fPLL IP core. The PLL output drives the master CGB. This is used for x6/x24 bonded and non-bonded modes.
Clock division factor	**1, 2, 4, 8**	Divides the master CGB clock input before generating bonding clocks.
Enable x24 non-bonded high-speed clock output port	**On/Off**	Enables the master CGB serial clock output port used for x6/xN non-bonded modes.
Enable PCIe clock switch interface	**On/Off**	Enables the control signals used for PCIe clock switch circuitry.
Enable mcgb_rst and mcgb_rst_stat ports	**On/Off**	The `mcgb_rst` and `mcgb_rst_stat` ports are required when the transceivers are configured in PCIE Gen 3 x2/x4/x8/x16 PIPE mode .
Number of auxiliary MCGB clock input ports	**0-1**	The number should be set to 1 when the transceivers are configured in PCIE Gen 3 x2/x4/x8/x16 PIPE mode and 0 for all other modes.
MCGB input clock frequency	**Read only**	Displays the master CGB's required input clock frequency. You cannot set this parameter.
MCGB output data rate	**Read only**	Displays the master CGB's output data rate. You cannot set this parameter. This value is calculated based on MCGB input clock frequency and MCGB clock division factor.
Enable bonding clock output ports	**On/Off**	Enables the `tx_bonding_clocks` output ports of the Master CGB used for channel bonding. You must enable this parameter for bonded designs.
PMA interface width	**8, 10, 16, 20, 32, 40, 64**	Specifies the PMA-PCS interface width. Match this value with the PMA interface width selected for the Native PHY IP core. You must select a proper value for generating bonding clocks for the Native PHY IP core.

Table 135. fPLL IP Core - Dynamic Reconfiguration

Parameters	Range	Description
Enable dynamic reconfiguration	**On/Off**	Enables the dynamic reconfiguration interface.
Enable Native PHY Debug Master Endpoint	**On/Off**	When enabled, the PLL IP includes an embedded Native PHY Debug Master Endpoint that connects internally Avalon memory-mapped interface slave. The NPDME can access the reconfiguration space of the transceiver. It can perform certain test and debug functions via JTAG using the System Console. This option requires you to enable the "Share reconfiguration interface" option for configurations using more than 1 channel and may also require that a `jtag_debug` link be included in the system.
Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE	**On/Off**	When enabled, the `reconfig_waitrequest` does not indicate the status of Avalon memory-mapped interface arbitration with PreSICE. The Avalon memory-mapped interface arbitration status is reflected in a soft status register bit. This feature requires that the "Enable control and status registers" feature under "Optional Reconfiguration Logic" be enabled.
Enable capability registers	**On/Off**	Enables capability registers, which provide high level information about the transceiver PLL's configuration
Set user-defined IP identifier	**1 to 5**	Sets a user-defined numeric identifier that can be read from the user_identifer offset when the capability registers are enabled
Enable control and status registers	**On/Off**	Enables soft registers for reading status signals and writing control signals on the phy interface through the embedded debug. Available signals include `pll_cal_busy`, `pll_locked` and `pll_powerdown`.
Configuration file prefix	**On/Off**	Specifies the file prefix to use for generated configuration files when enabled. Each variant of the IP should use a unique prefix for configuration files.
Generate SystemVerilog package file	**On/Off**	When enabled, The IP generates a SystemVerilog package file named "(Configuration file prefix)_reconfig_parameters.sv" containing parameters defined with the attribute values needed for reconfiguration.
Generate C header file	**On/Off**	When enabled, The IP generates a C header file named "(Configuration file prefix)_reconfig_parameters.h" containing macros defined with the attribute values needed for reconfiguration.
Generate MIF (Memory Initialize File)	**On/Off**	When enabled The IP generates an MIF (Memory Initialization File) named "(Configuration file prefix)_reconfig_parameters.mif". The MIF file contains the attribute values needed for reconfiguration in a data format.
Enable multiple reconfiguration profiles	**On/Off**	When enabled, you can use the GUI to store multiple configurations. The IP generates reconfiguration files for all of the stored profiles. The IP also checks your multiple reconfiguration profiles for consistency to ensure you can reconfigure between them.
Enable embedded reconfiguration streamer	**On/Off**	Enables the embedded reconfiguration streamer, which automates the dynamic reconfiguration process between multiple predefined configuration profiles.

continued...

Parameters	Range	Description
Generate reduced reconfiguration files	**On/Off**	When enabled, The Native PHY generates reconfiguration report files containing only the attributes or RAM data that are different between the multiple configured profiles.
Number of reconfiguration profiles	**1 to 31**	Specifies the number of reconfiguration profiles to support when multiple reconfiguration profiles are enabled.
Store current configuration to profile:	**1, 2, 4, 8**	Selects which reconfiguration profile to store when clicking the "Store profile" button.

Table 136. fPLL IP Core - Ports

Port	Direction	Clock Domain	Description
`pll_refclk0`	Input	N/A	Reference clock input port 0. There are five reference clock input ports. The number of reference clock ports available depends on the **Number of PLL reference clocks** parameter.
`pll_refclk1`	Input	N/A	Reference clock input port 1.
`pll_refclk2`	Input	N/A	Reference clock input port 2.
`pll_refclk3`	Input	N/A	Reference clock input port 3.
`pll_refclk4`	Input	N/A	Reference clock input port 4.
`mcgb_aux_clk0`	Input	N/A	Used for PCIe to switch between fPLL/ATX PLL during link speed negotiation.
`pcie_sw[1:0]`	Input	Asynchronous	2-bit rate switch control input used for PCIe protocol implementation.
`mcgb_rst`	Input	N/A	Resets the master CGB. This port should be used when implementing PCI Express Gen 3 PIPE only.
`tx_serial_clk`	Output	N/A	High speed serial clock output port for GX channels. Represents the x1 clock network.
`pll_locked`	Output	Asynchronous	Active high status signal which indicates if PLL is locked.
`pll_cascade_clk`	Output	N/A	fPLL cascade clock output port
`pll_pcie_clk`	Output	N/A	Used for PCIe.
`pll_cal_busy`	Output	Asynchronous	Status signal which is asserted high when PLL calibration is in progress. Perform logical OR with this signal and the `tx_cal_busy` port on the reset controller IP.
`tx_bonding_clocks[5:0]`	Output	N/A	Optional 6-bit bus which carries the low speed parallel clock outputs from the Master CGB. Used for channel bonding, and represents the x6/x24 clock network.
`mcgb_serial_clk`	Output	N/A	High speed serial clock output for x6/x24 non-bonded configurations.

continued...

Send Feedback

Port	Direction	Clock Domain	Description
`pcie_sw_done[1:0]`	Output	Asynchronous	2-bit rate switch status output used for PCIe protocol implementation.
`pll_cascade_clk`	Output	N/A	fPLL cascade output port
`outclk_div1`	Output	N/A	Core output clock (only in Core mode). The frequency is the PLL output frequency. No phase relationship to refclk.
`outclk_div2`	Output	N/A	Core output clock (only in Core mode). The frequency is half of the `outclk_div1` frequency. Phase aligned to `outclk_div1`.
`outclk_div4`	Output	N/A	Core output clock (only in Core mode). The frequency is quarter of the `outclk_div1` frequency. Phase aligned to `outclk_div1`.
`mcgb_rst_stat`	Output	N/A	Status signal for the master CGB. This port should be used when implementing PCI Express Gen 3 PIPE only

Related Information

- Calibration on page 433
- Reconfiguration Interface and Dynamic Reconfiguration on page 394
- Avalon Interface Specifications
  The ports related to reconfiguration are compliant with the Avalon Specification. Refer to the Avalon Specification for more details about these ports.
- Intel Stratix 10 Device Datasheet

3.1.3. CMU PLL

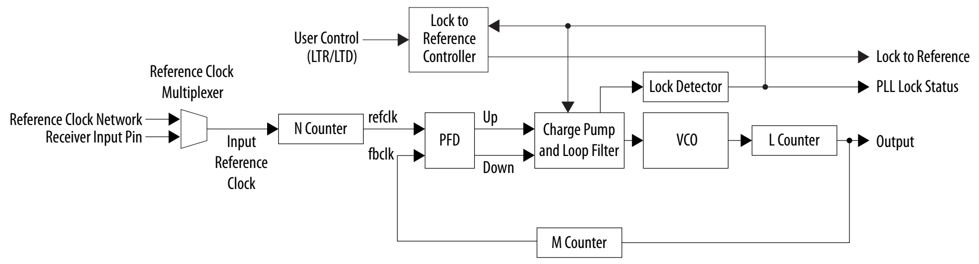
The clock multiplier unit (CMU) PLL resides locally within each transceiver channel. The channel PLL's primary function is to recover the receiver clock and data in the transceiver channel. In this case the PLL is used in clock and data recovery (CDR) mode.

When the channel PLL of channel 1 or channel 4 is configured in the CMU mode, the channel PLL can drive the local clock generation block (CGB) of its own channel. However, when the channel PLL is used as a CMU PLL, the channel can only be used as a transmitter channel because the CDR block is not available to recover the received clock and data.

The CMU PLL from transceiver channel 1 and channel 4 can also be used to drive other transceiver channels within the same transceiver bank. The CDR of channels 0, 2, 3, and 5 cannot be configured as a CMU PLL.

For data rates lower than 6 Gbps, the local CGB divider has to be engaged (TX local division factor in transceiver PHY IP under the TX PMA tab).

Figure 145. CMU PLL Block Diagram



Input Reference Clock

The input reference clock for a CMU PLL can be sourced from either the reference clock network or a receiver input pin. The input reference clock is a differential signal. The input reference clock must be stable and free-running at device power-up for proper PLL operation. If the reference clock is not available at device power-up, then you must recalibrate the PLL when the reference clock is available.

Note: The CMU PLL calibration is clocked by the `OSC_CLK_1` clock which must be stable and available for calibration to proceed.

Reference Clock Multiplexer (Refclk Mux)

The reference clock (`refclk`) mux selects the input reference clock to the PLL from the various reference clock sources available.

N Counter

The N counter divides the refclk mux's output. The N counter division helps lower the loop bandwidth or reduce the frequency to within the phase frequency detector's (PFD) operating range. Possible divide ratios are 1 (bypass), 2, 4, and 8.

Phase Frequency Detector (PFD)

The reference clock (`refclk`) signal at the output of the N counter block and the feedback clock (`fbclk`) signal at the output of the M counter block is supplied as an input to the PFD. The PFD output is proportional to the phase difference between the two inputs. It aligns the input reference clock (`refclk`) to the feedback clock (`fbclk`). The PFD generates an "Up" signal when the reference clock's falling edge occurs before the feedback clock's falling edge. Conversely, the PFD generates a "Down" signal when feedback clock's falling edge occurs before the reference clock's falling edge.

Charge Pump and Loop Filter (CP + LF)

The PFD output is used by the charge pump and loop filter to generate a control voltage for the VCO. The charge pump translates the "Up"/"Down" pulses from the PFD into current pulses. The current pulses are filtered through a low pass filter into a control voltage which drives the VCO frequency.

Voltage Controlled Oscillator (VCO)

The CMU PLL has a ring oscillator based VCO.

L Counter

The L counter divides the differential clocks generated by the CMU PLL. The division factors supported are 1 and 2.

M Counter

The M counter is used in the PFD's feedback path. The output of the L counter is connected to the M counter. The combined division ratios of the L counter and the M counter determine the overall division factor in the PFD's feedback path.

Lock Detector (LD)

The lock detector indicates when the CMU PLL is locked to the desired output's phase and frequency. The lock detector XORs the Up/Down pulses and indicates when the M counter's output and N counter's output are phase-aligned.

Related Information

Calibration on page 433

3.1.3.1. Instantiating CMU PLL IP Core

The CMU PLL IP core for Intel Stratix 10 transceivers provides access to the CMU PLLs in hardware. One instance of the CMU PLL IP core represents one CMU PLL in hardware.

1. Open the Intel Quartus Prime Pro Edition.
2. Click **Tools ➤ IP Catalog**.
3. In **IP Catalog**, under **Library ➤ Interface Protocols ➤ Transceiver PLL** , select **Intel Stratix 10 L-Tile/H-Tile Transceiver CMU PLL** and click **Add**.
4. In the **New IP Instance Dialog Box**, provide the IP instance name.
5. Select **Intel Stratix 10** device family.
6. Select the appropriate device and click **OK**.

The CMU PLL IP core **Parameter Editor** window opens.

3.1.3.2. CMU PLL IP Core - Parameters, Settings, and Ports

Table 137. CMU PLL IP Core - Parameters and Settings

Parameters	Range	Description
Number of PLL reference clocks	**1 to 4**	Specifies the number of input reference clocks for the CMU PLL. You can use this parameter for data rate reconfiguration.
Selected reference clock source	**0 to 3**	Specifies the initially selected reference clock input to the CMU PLL.
Bandwidth	**Low** **Medium** **High**	PLL bandwidth specifies the ability of the PLL to track the input clock and jitter. PLL with "Low" bandwidth setting indicates better jitter rejection but a slower lock time. PLL with a "High" bandwidth has a faster lock time but tracks more jitter. A "Medium" bandwidth offers a balance between lock time and jitter rejection

continued...

Parameters	Range	Description
VCCR_GXB and VCCT_GXB supply voltage for the Transceiver	**1_0V, and 1_1V** [46]	Selects the VCCR_GXB and VCCT_GXB supply voltage for the Transceiver.
PLL reference clock frequency	Refer to the GUI	Selects the input reference clock frequency (MHz) for the PLL.
PLL output frequency	Refer to the GUI	Specify the target output frequency (MHz) for the PLL.

Table 138. CMU PLL IP Core - Dynamic Reconfiguration

Parameters	Range	Description
Enable dynamic reconfiguration	**On/Off**	Enables the dynamic reconfiguration interface.
Enable Native PHY Debug Master Endpoint	**On/Off**	When enabled, the PLL IP includes an embedded Native PHY Debug Master Endpoint that connects internally Avalon memory-mapped interface slave. The NPDME can access the reconfiguration space of the transceiver. It can perform certain test and debug functions via JTAG using the System Console. This option requires you to enable the "Share reconfiguration interface" option for configurations using more than 1 channel and may also require that a `jtag_debug` link be included in the system.
Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE	**On/Off**	When enabled, the `reconfig_waitrequest` does not indicate the status of Avalon memory-mapped interface arbitration with PreSICE. The Avalon memory-mapped interface arbitration status is reflected in a soft status register bit. This feature requires that the "Enable control and status registers" feature under "Optional Reconfiguration Logic" be enabled.
Enable capability registers	**On/Off**	Enables capability registers, which provide high level information about the transceiver PLL's configuration
Set user-defined IP identifier	**0 to 255**	Sets a user-defined numeric identifier that can be read from the user_identifer offset when the capability registers are enabled
Enable control and status registers	**On/Off**	Enables soft registers for reading status signals and writing control signals on the phy interface through the embedded debug. Available signals include pll_cal_busy, pll_locked and pll_powerdown.
Configuration file prefix	**altera_xcvr_cdr_pll_s10**	Specifies the file prefix to use for generated configuration files when enabled. Each variant of the IP should use a unique prefix for configuration files.
Generate SystemVerilog package file	**On/Off**	When enabled, The IP generates a SystemVerilog package file named "(Configuration file prefix)_reconfig_parameters.sv" containing parameters defined with the attribute values needed for reconfiguration.
Generate C header file	**On/Off**	When enabled, The IP generates a C header file named "(Configuration file prefix)_reconfig_parameters.h" containing macros defined with the attribute values needed for reconfiguration.
Generate MIF (Memory Initialize File)	**On/Off**	When enabled The IP generates an MIF (Memory Initialization File) named "(Configuration file prefix)_reconfig_parameters.mif". The MIF file contains the attribute values needed for reconfiguration in a data format.

[46] Refer to the *Intel Stratix 10 Device Datasheet* for details about the minimum, typical, and maximum supply voltage specifications.

Table 139. CMU PLL IP Core - Parameter Summary

Parameters	Range	Description
Multiply factor (M-Counter)	**1 to 5**	Specifies the value for the feedback multiplier counter (M counter)
Divide factor (N-Counter)	**0 to 4**	Specifies the value for the pre-divider counter (N counter)
Divide factor (L-Counter)	**Low** **Medium** **High**	Specifies the value for the phase-frequency detector (PFD) circuit

Table 140. CMU PLL IP Core - Ports

Port	Range	Clock Domain	Description
`pll_refclk0`	input	N/A	Reference clock input port 0. There are 5 reference clock input ports. The number of reference clock ports available depends on the **Number of PLL reference clocks** parameter.
`pll_refclk1`	input	N/A	Reference clock input port 1.
`pll_refclk2`	input	N/A	Reference clock input port 2.
`pll_refclk3`	input	N/A	Reference clock input port 3.
`tx_serial_clk`	output	N/A	High speed serial clock output port for GX channels. Represents the x1 clock network.
`pll_locked`	output	Asynchronous	Active high status signal which indicates if PLL is locked.
`pll_cal_busy`	output	Asynchronous	Status signal that is asserted high when PLL calibration is in progress. Perform logical OR with this signal and the `tx_cal_busy` port on the reset controller IP.

Related Information

- Reconfiguration Interface and Dynamic Reconfiguration on page 394
- Calibration on page 433
- Avalon Interface Specifications
  The ports related to reconfiguration are compliant with the Avalon Specification. Refer to the Avalon Specification for more details about these ports.
- Intel Stratix 10 Device Datasheet

3.2. Input Reference Clock Sources

The transmitter PLL and the clock data recovery (CDR) block need an input reference clock source to generate the clocks required for transceiver operation. The input reference clock must be stable and free-running at device power-up for proper PLL calibrations.

Intel Stratix 10 transceiver PLLs have five possible input reference clock sources, depending on jitter requirements:

- Dedicated reference clock pins
- Receiver input pins
- Reference clock network (with two new high quality reference clock lines)
- PLL cascade output (fPLL only)
- Core clock network (fPLL only)

Note: The reference clock sources are distributed to all the banks in the same tile via the reference clock network, even if the banks are at different transceiver voltages. However, the reference clock sources cannot drive across tiles.

Intel recommends using the dedicated reference clock pins and the reference clock network for the best jitter performance.

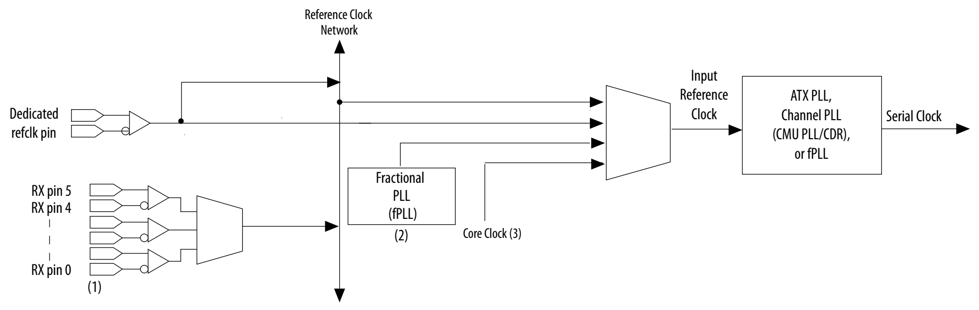
The following protocols require you to place the reference clock in the same bank as the transmit PLL:

For the best jitter performance, Intel recommends placing the reference clock as close as possible, to the transmit PLL. The following protocols require the reference clock to be placed in same bank as the transmit PLL:

- OTU2e, OTU2, OC-192 and 10G PON
- 6G and 12G SDI

Note: For optimum performance of GXT channel, the reference clock of transmit PLL is recommended to be from a dedicated reference clock pin in the same triplet.

Figure 146. Input Reference Clock Sources



Note : (1) Any RX pin in the same bank can be used as an input reference clock.
(2) The output of another PLL can be used as an input reference clock source during PLL cascading. Intel Stratix 10 transceivers support fPLL to fPLL and ATX PLL to fPLL cascading. Refer to "PLL Cascading Clock Network" for more details on PLL cascading.
(3) Core Clock present only for fPLL.

Note: In Intel Stratix 10 devices, the FPGA fabric core clock network can be used as an input reference source for fPLL only.

Related Information

- PLL Cascading Clock Network on page 302
- PLL Cascading as an Input Reference Clock Source on page 282

3.2.1. Dedicated Reference Clock Pins

To minimize the jitter, the advanced transmit (ATX) PLL and the fractional PLL (fPLL) can source the input reference clock directly from the reference clock buffer without passing through the reference clock network. The input reference clock is also fed into the reference clock network. It can also drive the core fabric.

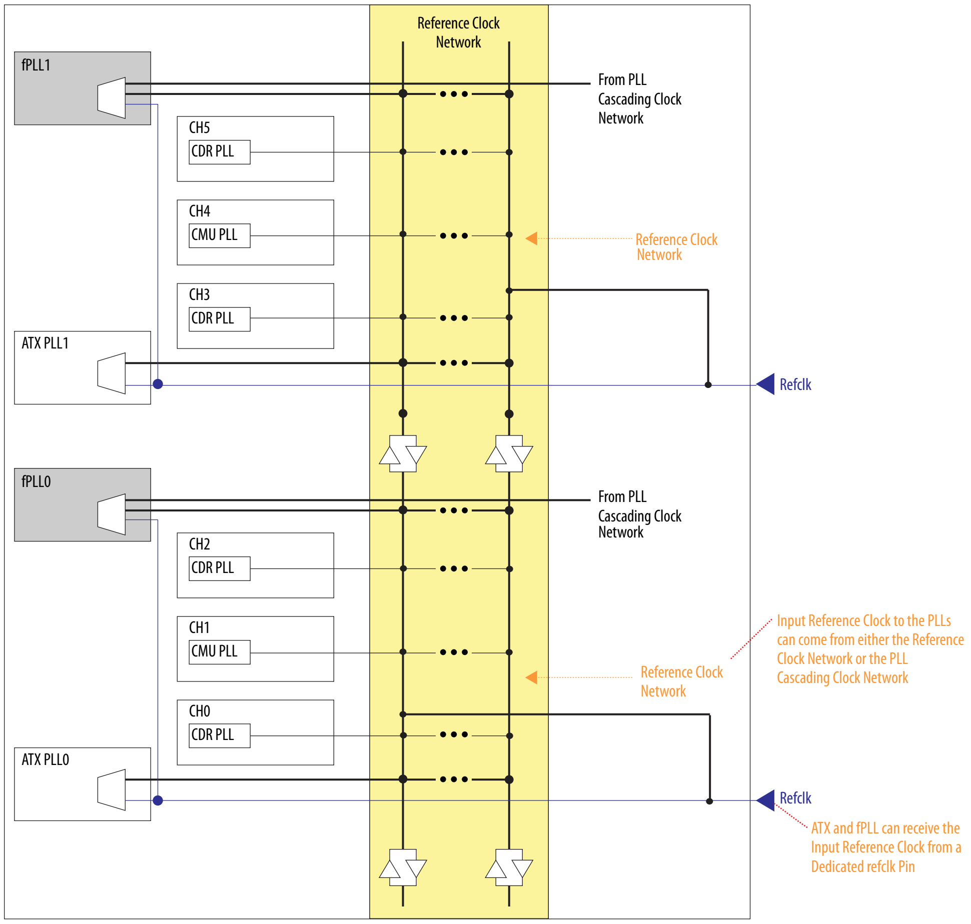
Note: The reference clock pins use thick oxide and are thus safe from damage due to hot swapping.

Set the following assignments to the dedicated reference clock pins through the **Assignment Editor** of the Intel Quartus Prime Pro Edition software. Since the reference clock is a direct input to the Native PHY IP core and not an analog parameter you cannot set it through the GUI.

Use the **XCVR_S10_REFCLK_TERM_TRISTATE** QSF assignment to set the `refclk` tristate termination setting. All other assignments like **INPUT_TERMINATION DIFFERENTIAL**, **XCVR_REFCLK_PIN_TERMINATION AC_COUPLING**, **XCVR_REFCLK_PIN_TERMINATION DC_COUPLING_EXTERNAL_RESISTOR**, and **XCVR_REFCLK_PIN_TERMINATION DC_COUPLING_INTERNAL_100_OHMS** are for older device families and will be ignored when used for Intel Stratix 10 devices.

Figure 147. Dedicated Reference Clock Pins

There are two dedicated reference clock (`refclk`) pins available in each transceiver bank. The bottom `refclk` pin feeds the bottom ATX PLL and fPLL. The top `refclk` pin feeds the top ATX PLL, fPLL, and CMU PLL via the reference clock network. The dedicated reference clock pins can also drive the reference clock network and the core fabric.


Reference Clock Network
fPLL1
From PLL Cascading Clock Network
CH5
CDR PLL
CH4
CMU PLL
Reference Clock Network
CH3
CDR PLL
ATX PLL1
Refclk
fPLL0
From PLL Cascading Clock Network
CH2
CDR PLL
CH1
CMU PLL
Input Reference Clock to the PLLs can come from either the Reference Clock Network or the PLL Cascading Clock Network
Reference Clock Network
CH0
CDR PLL
ATX PLL0
Refclk
ATX and fPLL can receive the Input Reference Clock from a Dedicated refclk Pin

3.2.1.1. Reference Clock I/O Standard

Pin Planner or Assignment Editor Name: I/O Standard

Description: The I/O Standard dictates the type of interface standard used on the pins.

Syntax for QSF Setting:

set_instance_assignment -name IO_STANDARD <value> -to <dedicated refclk pin name>

Table 141. Available Options

Value	Description
High Speed Current Steering Logic (HCSL)	High Speed Current Steering Logic (HCSL) is the recommended differential I/O standard for PCI Express applications. It is an open emitter output with a 15 mA

continued...

Value	Description
	current source and requiring 50 Ω external resistor to ground for the output to be switching. In the PCI Express configurations, DC-coupling is allowed on the REFCLK if the selected REFCLK I/O standard is HCSL.
Current Mode Logic (CML)	Current mode logic (CML), or source-coupled logic (SCL), is the recommended differential I/O standard intended for data transmission at speeds between 312.5 Mbps & 3.124 Gbps across standard printed circuit boards. The data transmission is point-to-point, unidirectional, and is usually terminated at the destination with 50 Ω on both differential lines. CML is frequently used in interfaces to fiber optic components, connections between modules, HDMI video etc.,
Low Voltage Differential Signaling (LVDS)	Low-voltage differential signaling, or LVDS, also known as TIA/EIA-644, is the low power and high speeds as it uses inexpensive twisted-pair copper cables.
Low-Voltage Positive/Pseudo Emitter–Coupled Logic (LVPECL)	The LVPECL electrical specification is similar to LVDS, but operates with a larger differential voltage swing. LVPECL is less power efficient than LVDS due to its ECL origins and larger swings.

3.2.1.2. Dedicated Reference Clock Pin Termination (XCVR_S10_REFCLK_TERM_TRISTATE)

Pin Planner or Assignment Editor Name: Dedicated Reference Clock Pin Termination

Description: Specifies if the termination for dedicated reference clock pin is tri-stated. It defaults to **TRISTATE_OFF** for non-HCSL cases.

Syntax for QSF Setting:

set_instance_assignment -name XCVR_S10_REFCLK_TERM_TRISTATE <value> -to <dedicated refclk pin name>

Table 142. Available Options

Value	Description
TRISTATE_OFF	Internal termination enabled and on-chip biasing circuitry enabled
TRISTATE_ON	Internal termination tri-stated. Off-chip termination and biasing circuitry must be implemented

Assign To

Reference clock pin.

3.2.2. Receiver Input Pins

Receiver input pins can be used as an input reference clock source to transceiver PLLs. However, they cannot be used to drive core fabric.

The receiver input pin drives the reference clock network, which can then feed any number of transmitter PLLs on the same tile. When a receiver input pin is used as an input reference clock source, the clock data recovery (CDR) block of that channel is not available. As indicated in Input Reference Clock Sources on page 277, only one RX differential pin pair per three channels can be used as an input reference clock source at any given time.

3.2.3. PLL Cascading as an Input Reference Clock Source

In PLL cascading, PLL outputs are connected to the cascading clock network. In this mode, the output of one PLL drives the reference clock input of another PLL. PLL cascading can generate frequency outputs not normally possible with a single PLL solution. The transceivers in Intel Stratix 10 devices support fPLL to fPLL cascading. ATX PLL to fPLL cascading is available to OTN and SDI protocols only.

Note:

- To successfully complete the calibration process, the reference clocks driving the PLLs (ATX PLL, fPLL) must be stable and free running at start of FPGA configuration. Otherwise, recalibration will be necessary.
- When the fPLL is used as a cascaded fPLL (downstream fPLL), a user recalibration on the fPLL is required. Refer to "User Recalibration" section in "Calibration" chapter for more information.

Related Information

PLL Cascading Clock Network on page 302

3.2.4. Reference Clock Network

The reference clock network distributes a reference clock source to the entire transceiver tile even if the VCCR_GXB and VCCT_GXB operating voltages of the banks in the tile are different. This allows any reference clock pin to drive any transmitter PLL on the same side of the device. Designs using multiple transmitter PLLs which require the same reference clock frequency and are located in the same tile, can share the same dedicated reference clock (`refclk`) pin.

In addition, two high quality reference clock line with dedicated voltage regulator are available. In a tile, one line is driven by the bottom reference clocks, and the other line is driven by the top reference clocks. Use the high quality reference clock line for OTN, SDI or GXT implementation.

To assign a reference clock source to a high quality reference clock line, please use the following QSF assignment:

```
set_instance_assignment -name XCVR_USE_HQ_REFCLK ON -to pin_name
```

Reference clock and transmitter PLL location awareness is critical when using high quality reference clock lines. For example, there are fitter errors if you use the bottom reference clock in Bank 0 to drive the transmitter PLL in Bank 2 and bottom reference clock in Bank 1 to drive the transmitter PLL in Bank 3.

3.2.5. Core Clock as an Input Reference Clock

The core clock can be used as an input reference clock for fPLL only.

The core clock network routes the clock directly to the PLL. For best performance, use the dedicated reference clock pins or the reference clock network.

3.3. Transmitter Clock Network

The transmitter clock network routes the clock from the transmitter PLL to the transmitter channel. It provides two types of clocks to the transmitter channel:

- High-Speed Serial Clock—high-speed clock for the serializer.
- Low-Speed Parallel Clock—low-speed clock for the serializer and the PCS.

In a bonded channel configuration, both the serial clock and the parallel clock are routed from the transmitter PLL to the transmitter channel. In a non-bonded channel configuration, only the serial clock is routed to the transmitter channel, and the parallel clock is generated locally within the channel. To support various bonded and non-bonded clocking configurations, four types of transmitter clock network lines are available:

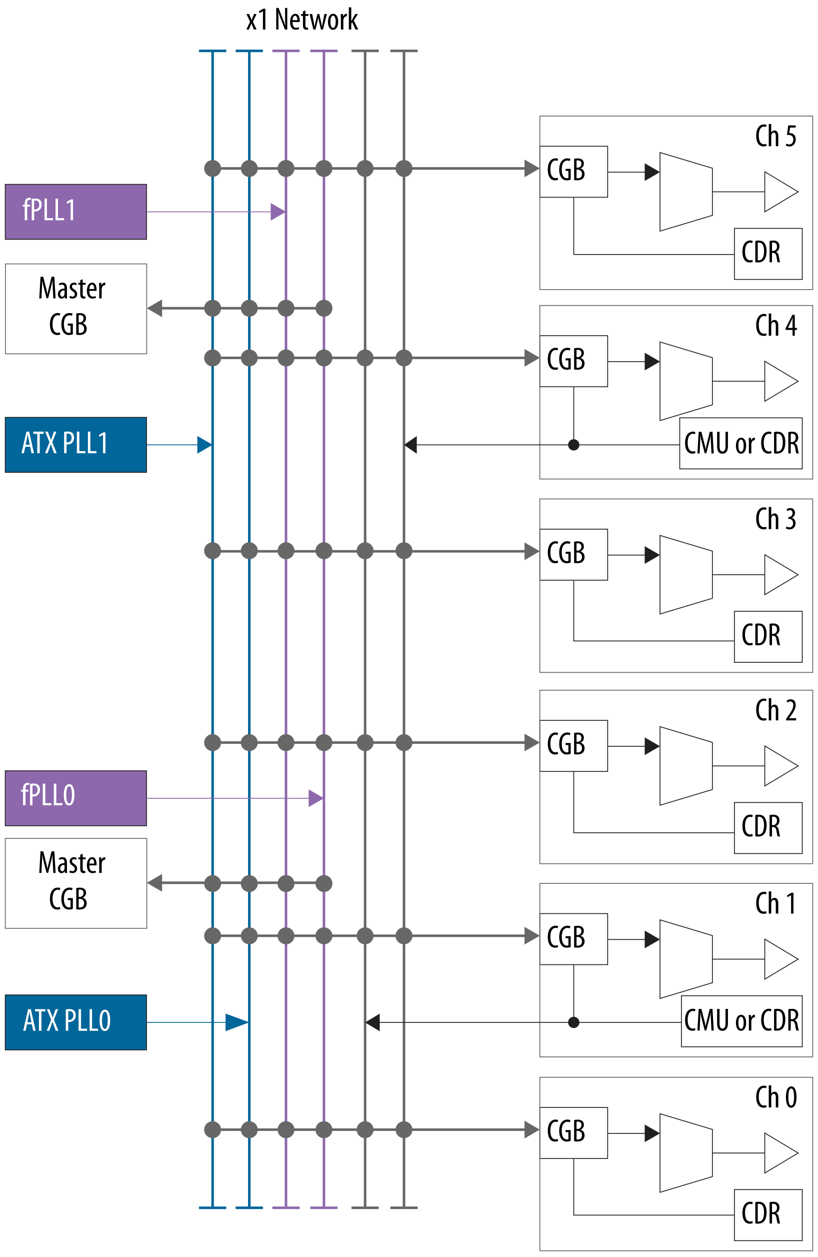
- x1 clock lines
- x6 clock lines
- x24 clock lines
- GXT Clock Network

3.3.1. x1 Clock Lines

The x1 clock lines route the high speed serial clock output of a PLL to any channel within a transceiver bank. The low speed parallel clock is then generated by that particular channel's local clock generation block (CGB). Non-bonded channel configurations use the x1 clock network.

The x1 clock lines can be driven by the ATX PLL, fPLL, or by either one of the two channel PLLs (channel 1 and 4 when used as a CMU PLL) within a transceiver bank.

Figure 148. x1 Clock Lines



3.3.2. x6 Clock Lines

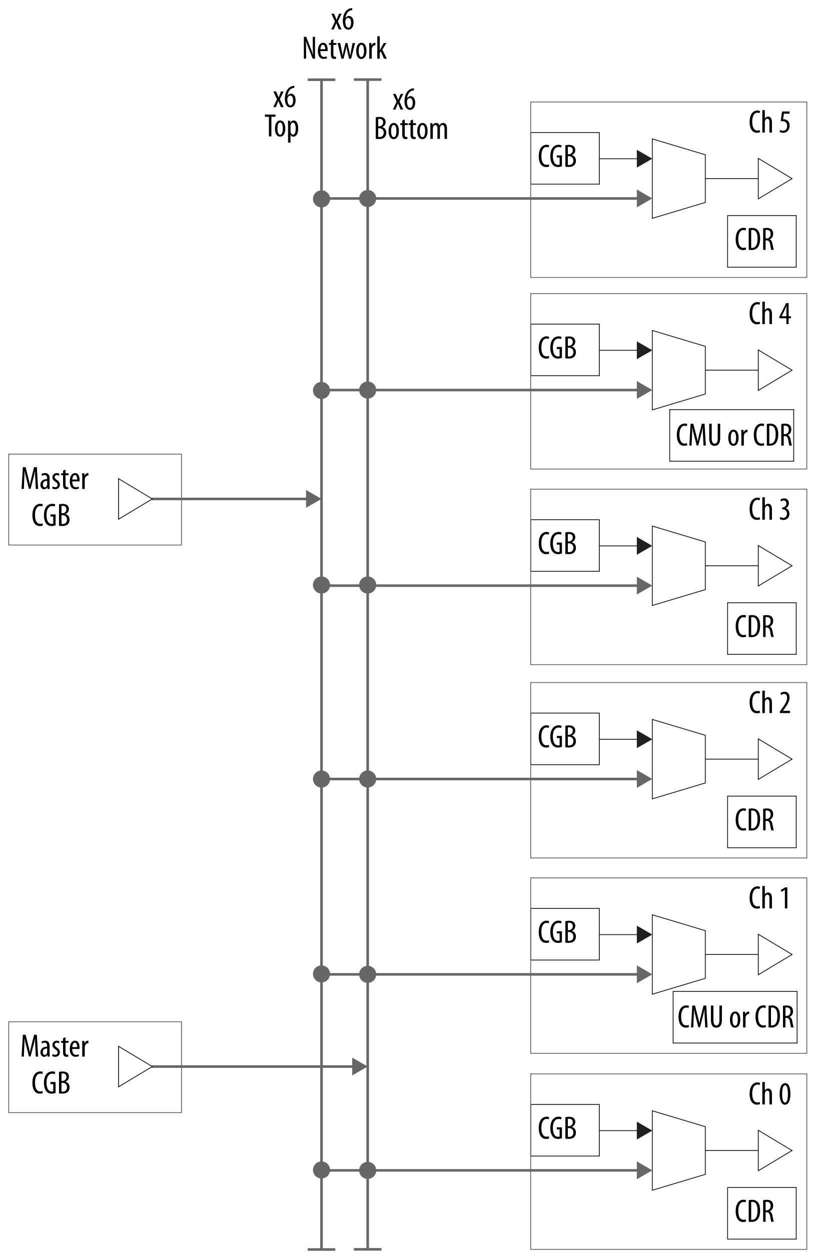
The x6 clock lines route the clock within a transceiver bank. The x6 clock lines are driven by the master CGB. The master CGB can only be driven by the ATX PLL or the fPLL. Because the CMU PLLs cannot drive the master CGB, the CMU PLLs cannot be used for bonding purposes. There are two x6 clock lines per transceiver bank, one for each master CGB. Any channel within a transceiver bank can be driven by the x6 clock lines.

For bonded configuration mode, the low speed parallel clock output of the master CGB is used and the local CGB within each channel is bypassed. For non-bonded configurations, use the master CGB to provide a high-speed serial clock output to each channel, in case you have multiple channels driven by the same ATX/fPLL, and if the non-bonded channels span across a transceiver bank.

The x6 clock lines also drive the x24 clock lines which route the clocks to the neighboring transceiver banks.

Note: For more information about bonded configurations, refer to the "Channel Bonding" chapter.

Figure 149. x6 Clock Lines



Related Information

Channel Bonding on page 299

3.3.3. x24 Clock Lines

The x24 clock lines route the transceiver clocks across multiple transceiver banks within the same transceiver tile.

The master CGB drives the x6 clock lines and the x6 clock lines drive the x24 clock lines. There are two x24 clock lines: x24 Up and x24 Down. x24 Up clock lines route the clocks to transceiver banks located above the current bank. x24 Down clock lines route the clocks to transceiver banks located below the current bank.

The x24 clock lines can be used in both bonded and non-bonded configurations. For bonded configurations, the low-speed parallel clock output of the master CGB is used, while the local CGB within each channel is bypassed. For non-bonded configurations, use the master CGB to provide a high-speed serial clock output to each channel, in case you have multiple channels driven by the same ATX/fPLL, and if the non-bonded channels span across a transceiver bank. A maximum of 24 channels can be used in a single bonded or non-bonded x24 group.

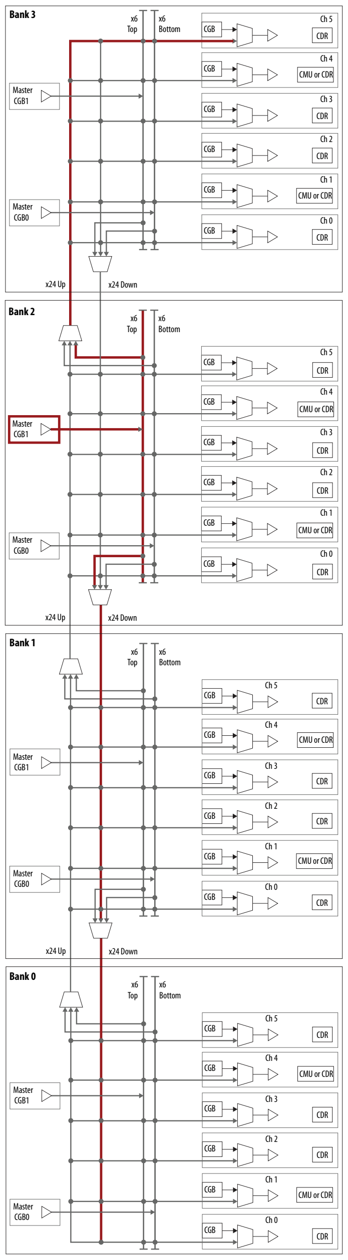
When the banks within a transceiver tile are powered at different voltages (for example, some banks are operating at 1.03 V while other banks are operating at 1.12 V), the x24 clock lines are only allowed to traverse between contiguous banks operating at the same VCCR_GXB and VCCT_GXB voltages. The x24 clock lines crossing boundaries of banks operating at different voltages is not allowed. See the Intel Stratix 10 Device Family Pin Connection Guidelines for a description of the transceiver power connection guidelines.

Note: The VCCR_GXB and VCCT_GXB per bank option is not enabled in the Intel Quartus Prime software by default. Use the following QSF assignment to enable this option:

```
set_global_assignment -name ALLOW_VCCR_VCCT_PER_BANK ON
```

For more information about bonded configurations, refer to *Channel Bonding*.

Figure 150. x24 Clock Network



Related Information

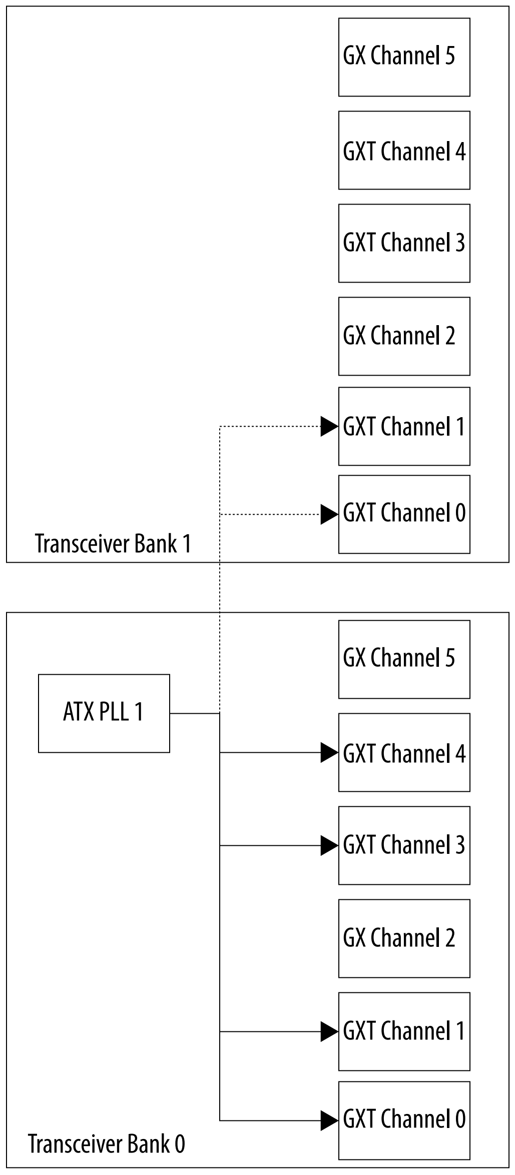
- Channel Bonding on page 299
- Intel Stratix 10 Device Datasheet
  Refer to the "L-Tile/H-Tile Transceiver Clock Network Maximum Data Rate Specifications—Preliminary" table in the Intel Stratix 10 Device Datasheet.
- Intel Stratix 10 Device Family Pin Connection Guidelines
  Refer to the Intel Stratix 10 Device Family Pin Connection Guidelines for a description of the transceiver power connection guidelines.

3.3.4. GXT Clock Network

ATX PLLs can access the GXT Clock Network. The clock network allows a single ATX PLL to drive up to 6 GXT channels in non-bonded mode on H-Tile and up to 4 GXT channels in non-bonded mode on L-Tile.

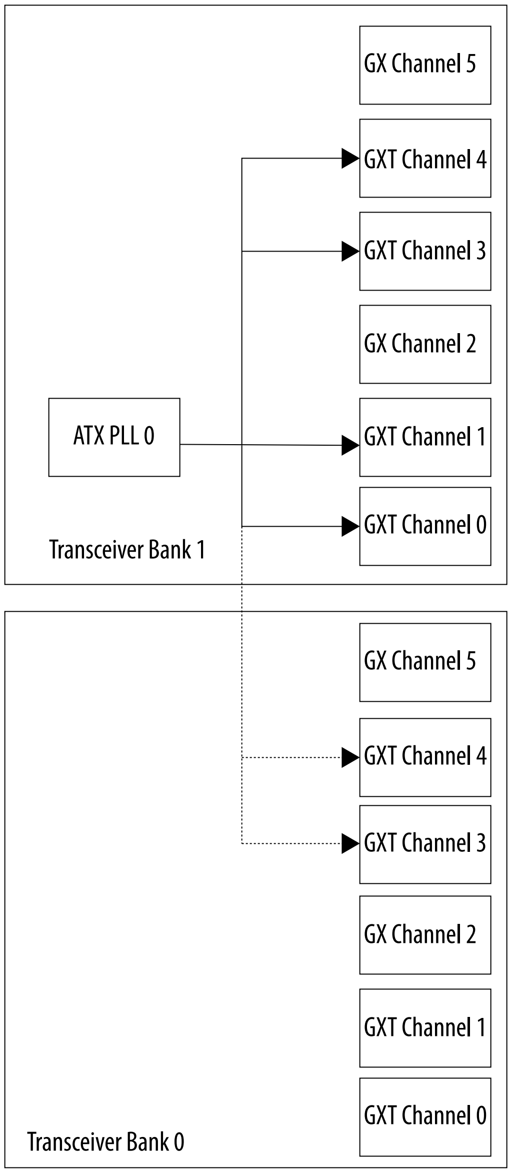
- Each ATX PLL has a 3:1 mux that can select whether to select its own, above ATX PLL, or below ATX PLL to drive two adjacent GXT channels.
  - — Intel Quartus Prime Pro Edition does not infer the 3:1 mux based on your design. Instead, you need to instantiate up to 3 ATX PLL IP cores. One instance is configured as a PLL, while the other two instances have the 3:1 mux drop-down set to select the first ATX PLL as the adjacent GXT channels source.
    - Use the additional drop-down to select source based on the device selected in the project.
  - — The top ATX PLL in a bank can drive the following GXT channels:
    - Channels 0,1, 3, 4 in the bank
    - Channels 0, 1 in the bank above in the same H-Tile
  - — The bottom ATX PLL in a bank can drive the following GXT channels:
    - Channels 0, 1, 3, 4 in the bank
    - Channels 3, 4 in the bank below in the same H-Tile

Figure 151. Top ATX PLL in a Transceiver Bank Driving GXT Channels



Refer also to Figure 140 on page 257.

Figure 152. Bottom ATX PLL in a Transceiver Bank Driving GXT Channels

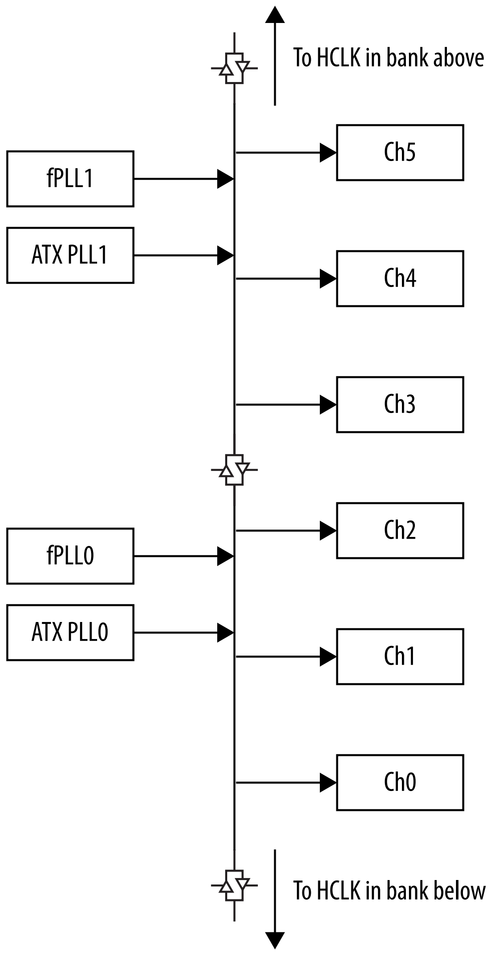


Refer also to Figure 140 on page 257.

3.3.5. HCLK Network

In PCIe mode, the HCLK network drives the PIPE interface by driving `pll_pcie_clk` from the fPLL or ATX PLL. The HCLK network is separated by a triplet (3 channels in a bank). The three channels in a triplet share the same `pll_pcie_clk` from either the ATX or fPLL. This means that two independent PCIe x1 or x2 links cannot be fitted in the same triplet.

Figure 153. HCLK Network



3.4. Clock Generation Block

In Intel Stratix 10 devices, there are two types of clock generation blocks (CGBs)

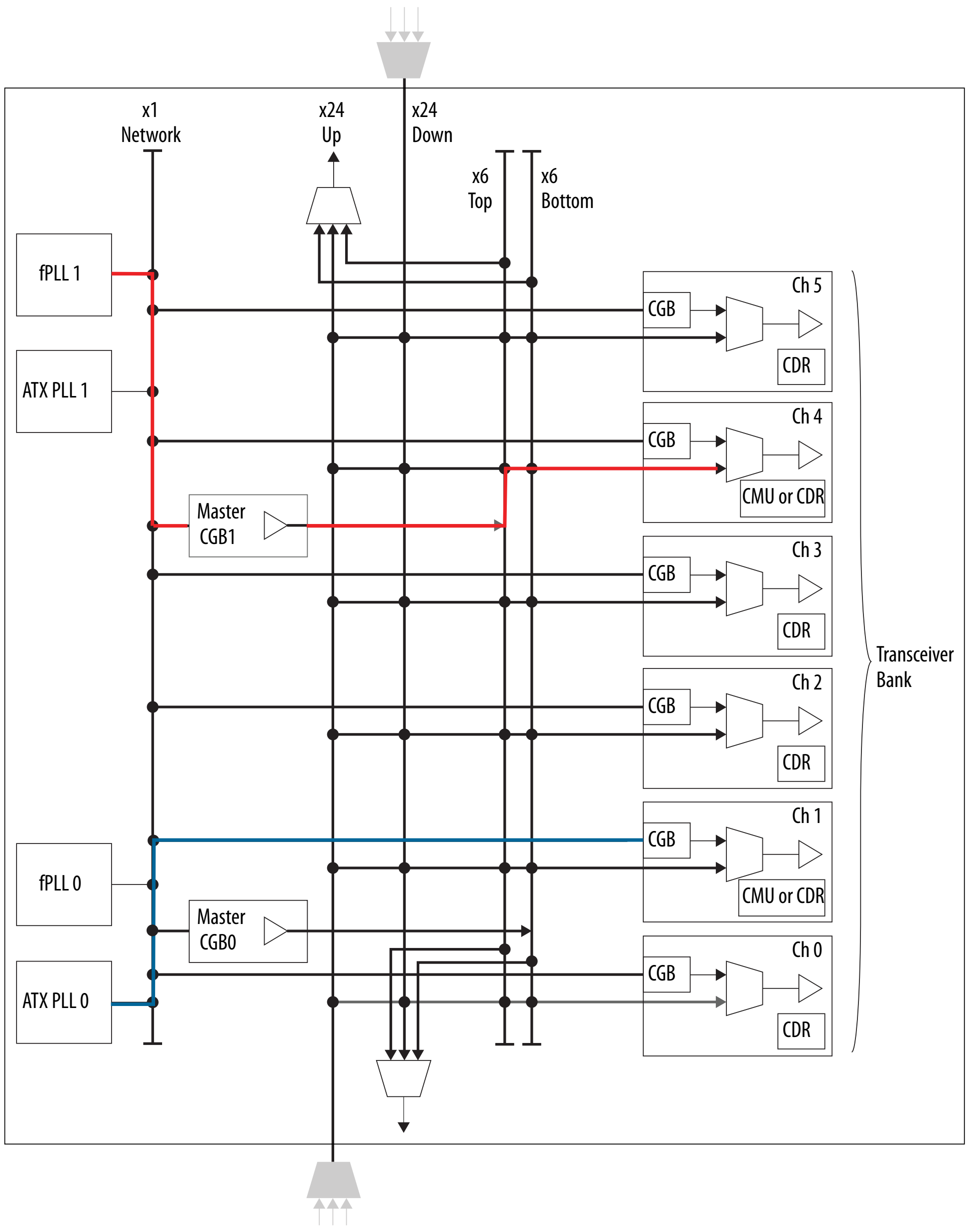
- Local clock generation block (local CGB)
- Master clock generation block (master CGB)

Each transmitter channel has a local clock generation block (CGB). For non-bonded channel configurations, the serial clock generated by the transmit PLL drives the local CGB of each channel. The local CGB generates the parallel clock used by the serializer and the PCS.

There are two standalone master CGBs within each transceiver bank. The master CGB provides the same functionality as the local CGB within each transceiver channel. The output of the master CGB can be routed to other channels within a transceiver bank using the x6 clock lines. The output of the master CGB can also be routed to channels in other transceiver banks using the x24 clock lines. Each transmitter channel has a multiplexer to select its clock source from either the local CGB or the master CGB.

Note: If you are using a master CGB, do not configure the adjacent ATX PLL from a GX to a GXT mode.

Figure 154. Clock Generation Block and Clock Network



The local clock for each transceiver channel can be sourced from either the local CGB via the x1 network, or the master CGB via the x6/x24 network. For example, as shown by the red highlighted path, the fPLL 1 drives the x1 network which in turn drives the master CGB. The master CGB then drives the x6 clock network which routes the clocks to the local channels. As shown by the blue highlighted path, the ATX PLL 0 can also drive the x1 clock network which can directly feed a channel's local CGB. In this case, the low speed parallel clock is generated by the local CGB.

Related Information

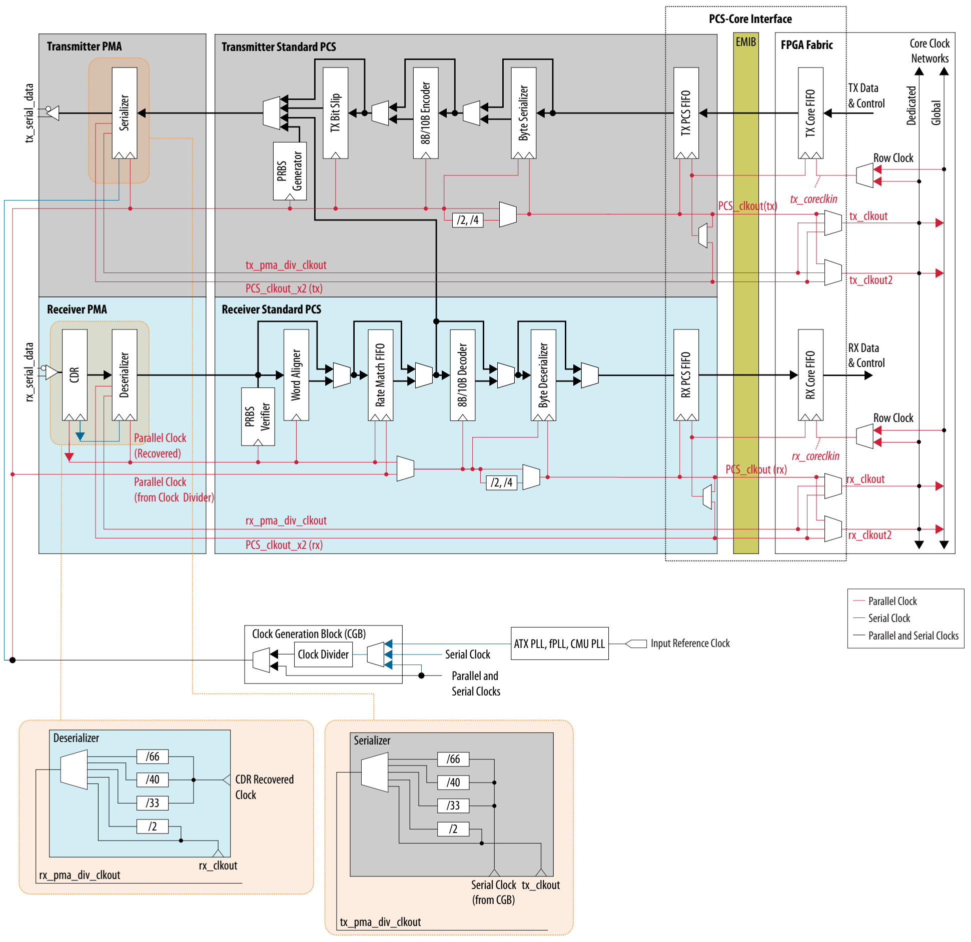
Clock Generation Block (CGB) on page 20

3.5. FPGA Fabric-Transceiver Interface Clocking

The FPGA fabric-transceiver interface consists of clock signals from the FPGA fabric into the transceiver and clock signals from the transceiver into the FPGA fabric.

The transmitter channel forwards a parallel output clock `tx_clkout` to the FPGA fabric to clock the transmitter data and control signals into the transmitter. The receiver channel forwards a parallel output clock `rx_clkout` to the FPGA fabric to clock the data and status signals from the receiver into the FPGA fabric. Based on the receiver channel configuration, the parallel output clock is recovered from either the receiver serial data or the `rx_clkout` clock (in configurations without the rate matcher) or the `tx_clkout` clock (in configurations with the rate matcher).

Figure 155. FPGA Fabric—Transceiver Interface Clocking (Standard PCS Example)



Send Feedback

The Standard PCS and Enhanced PCS `tx_clkout` and `tx_clkout2` outputs can be driven from the following sources:

- PCS clkout (tx)
- PCS clkout x2 (tx)
- pma_div_clkout (tx)

The Standard PCS and Enhanced PCS `rx_clkout` and `rx_clkout2` outputs can be driven from the following sources:

- PCS clkout (RX)
- PCS clkout x2 (RX)
- pma_div_clkout (RX)

For example, if you use the Enhanced PCS Gearbox with a 66:40 ratio, then you can use `tx_pma_div_clkout` with a divide-by-33 ratio to clock the write side of the TX FIFO, instead of using a PLL to generate the required clock frequency, or using an external clock source.

Related Information

- PMA Parameters on page 39
- PCS-Core Interface Parameters on page 42
- PCS-Core Interface Ports on page 66

3.6. Double Rate Transfer Mode

The double rate transfer mode is a new mode introduced in Intel Stratix 10 device to reduce data latency. Enabling double-rate transfer mode splits the PCS parallel data into two words and each word is transferred to or from the transceiver PCS at twice the parallel clock frequency.

Related Information

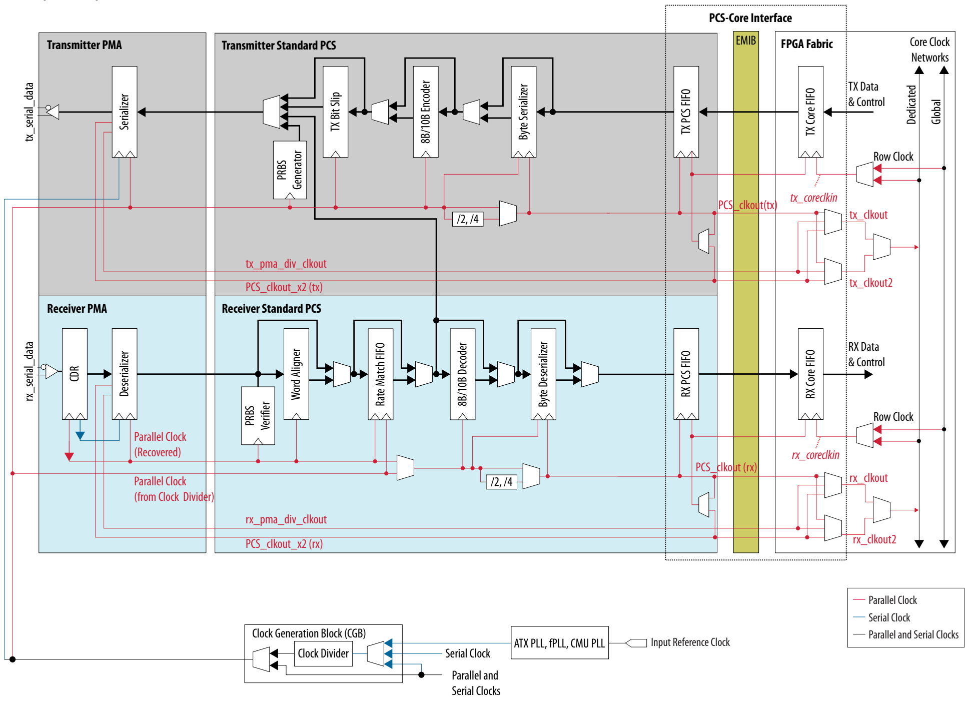
How to Implement Double Rate Transfer Mode on page 143

3.7. Transmitter Data Path Interface Clocking

The clocks generated by the PLLs are used to clock the channel PMA and PCS blocks. The clocking architecture is different for the Standard PCS and the Enhanced PCS. For PCS Direct, the clocking architecture remains the same as Standard PCS.

Figure 156. Transmitter Standard PCS and PMA Clocking

The master or the local CGB provides the high speed serial clock to the serializer of the transmitter PMA, and the low speed parallel clock to the transmitter PCS.



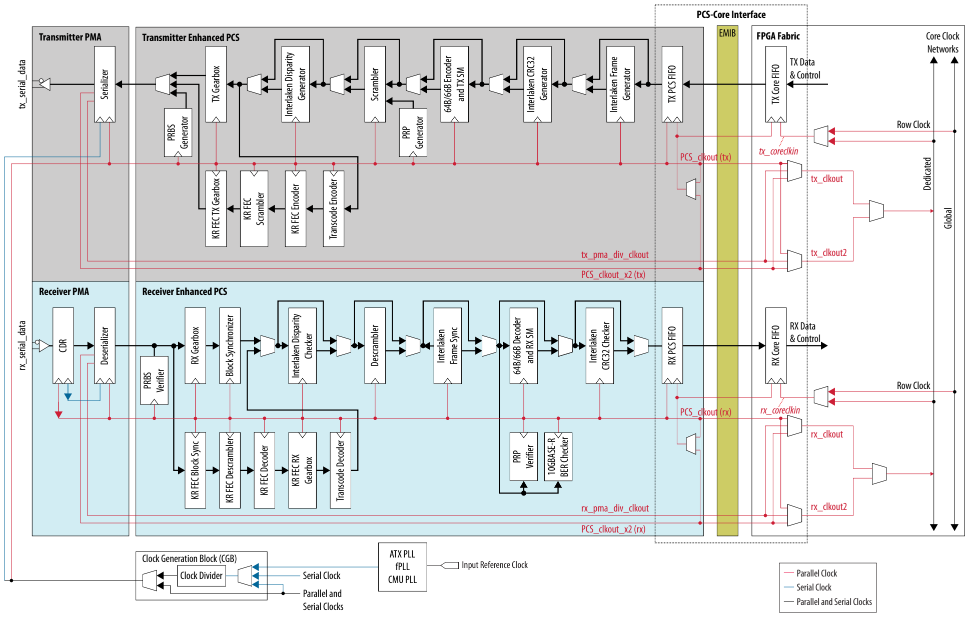
In the Standard PCS, for configurations that do not use the byte serializer, the parallel clock is used by all the blocks up to the read side of the TX PCS FIFO. For configurations that use the byte serializer block, the clock divided by 2 or 4 is used by the byte serializer and the read side of the TX PCS FIFO. The clock used to clock the read side of the TX PCS FIFO is also forwarded to the FPGA fabric to provide an interface between the FPGA fabric and the transceiver.

If the `tx_clkout` that is forwarded to the FPGA fabric is used to clock the write side of the phase compensation FIFO, then both sides of the FIFO have 0 ppm frequency difference because it is the same clock that is used.

If you use a different clock than the `tx_clkout` to clock the write side of the phase compensation FIFO, then you must ensure that the clock provided has a 0 ppm frequency difference with respect to the `tx_clkout`.

Figure 157. Transmitter Enhanced PCS and PMA Clocking

The master or local CGB provides the serial clock to the serializer of the transmitter PMA, and the parallel clock to the transmitter PCS.



In the Enhanced PCS, the parallel clock is used by all the blocks up to the read side of the TX PCS FIFO. The clocks of all channels in bonded configuration are forwarded. For example, you can pick `tx_clkout[0]` as the source for clocking their TX logic in the core.

For the Enhanced PCS, the transmitter PCS forwards the following clocks to the FPGA fabric:

- `tx_clkout` for each transmitter channel in non-bonded and bonded configuration. In bonded configuration, any `tx_clkout` can be used depending on your core timing requirements.

You can clock the transmitter datapath interface using one of the following methods:

- Quartus Prime selected transmitter datapath interface clock
- User-selected transmitter datapath interface clock

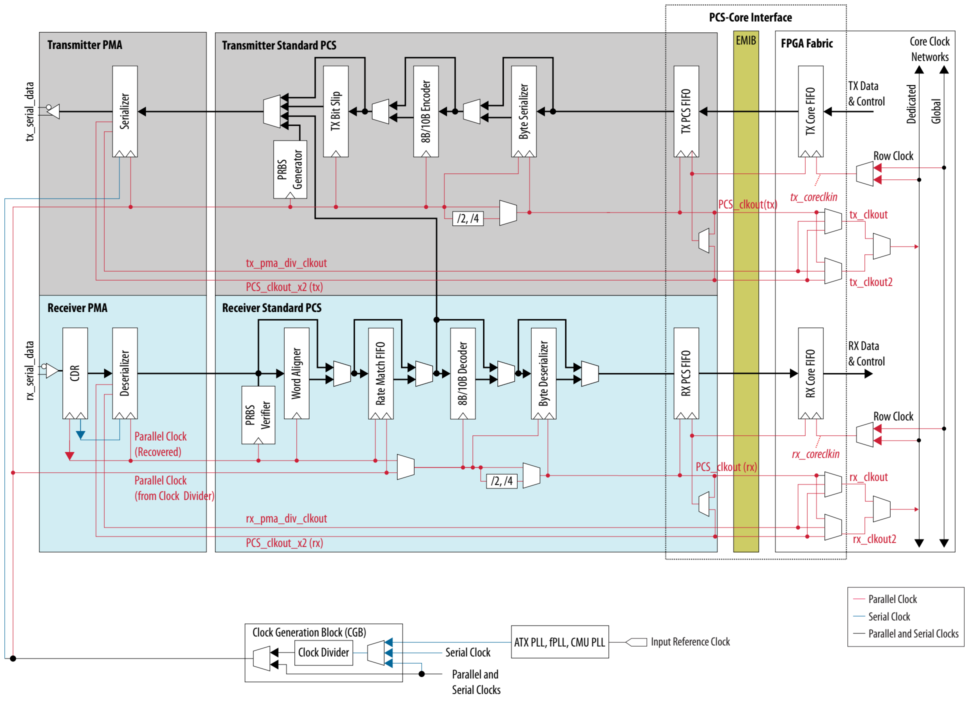
3.8. Receiver Data Path Interface Clocking

The CDR block present in the PMA of each channel recovers the serial clock from the incoming data. The CDR block also divides the recovered serial clock to generate the recovered parallel clock. Both the recovered serial and the recovered parallel clocks are used by the deserializer. The receiver PCS can use the following clocks based on the configuration of the receiver channel:

- Recovered parallel clock from the CDR in the PMA.
- Parallel clock from the clock divider used by the transmitter PCS (if enabled) for that channel.
- The Enhanced PCS receiver parallel clock (`rx_clkout`).

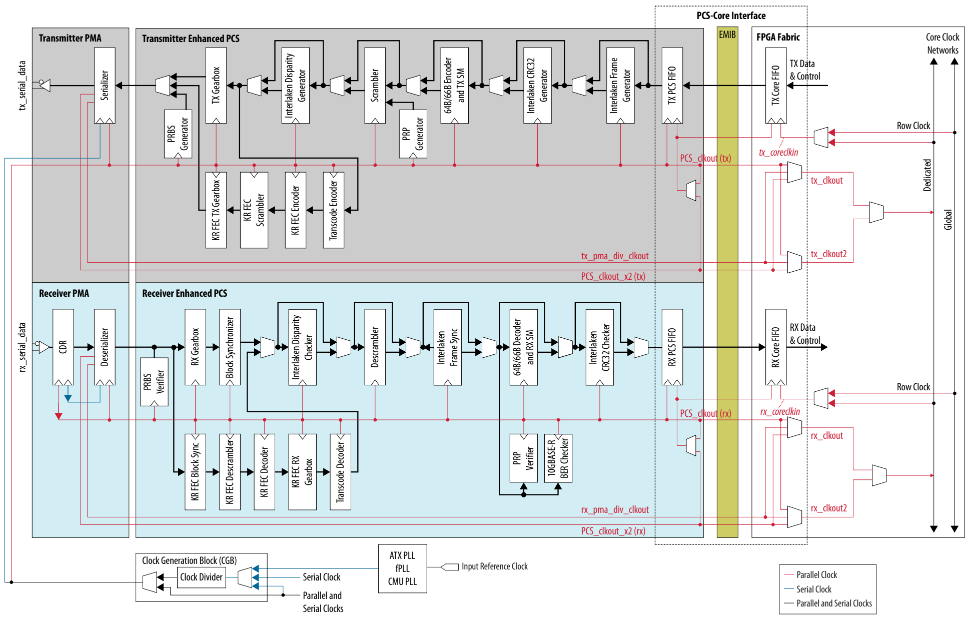
For configurations that use the byte deserializer block, the clock divided by 2 or 4 is used by the byte deserializer and the write side of the RX phase compensation FIFO.

Figure 158. Receiver Standard PCS and PMA Clocking



All configurations that use the Standard PCS channel must have a 0 ppm phase difference between the receiver datapath interface clock and the read side clock of the RX phase compensation FIFO.

Figure 159. Receiver Enhanced PCS and PMA Clocking



The receiver PCS forwards the following clocks to the FPGA fabric:

- `rx_clkout`—for each receiver channel when the rate matcher is not used.
- `tx_clkout`—for each receiver channel when the rate matcher is used.
- `rx_clkout`—from Standard PCS.

3.9. Channel Bonding

For Intel Stratix 10 devices, two types of bonding modes are available:

- PMA bonding
- PMA and PCS bonding

3.9.1. PMA Bonding

PMA bonding reduces skew between PMA channels. In PMA bonding, only the PMA portion of the transceiver datapath is skew compensated. The PCS is not skew compensated.

In Intel Stratix 10 devices, there is a single PMA bonding scheme:

- x6/x24 bonding

The channels in the bonded group do not have to be placed contiguously.

3.9.1.1. x6/x24 Bonding

In x6/x24 bonding mode, a single transmit PLL is used to drive multiple channels.

The steps below explain the x6/24 bonding process:

1. The ATX PLL or the fPLL generates a high speed serial clock.
2. The PLL drives the high speed serial clock to the master CGB via the x1 clock network.
3. The master CGB drives the high speed serial and the low speed parallel clock into the x6 clock network.
4. The x6 clock network feeds the TX clock multiplexer for the transceiver channels within the same transceiver bank. The local CGB in each transceiver channel is bypassed.
5. To drive the channels in adjacent transceiver banks, the x6 clock network drives the x24 clock network. The x24 clock network feeds the TX clock multiplexer for the transceiver channels in these adjacent transceiver banks.

Note: The x24 clock lines are only allowed to traverse between contiguous banks operating at the same VCCR_GXB/VCCT_GXB voltages. The x24 clock lines crossing boundaries of banks operating at different voltages is not allowed.

For more information about the transceiver power connection guidelines, refer to the *Intel Stratix 10 Device Family Pin Connection Guidelines*.

Related Information

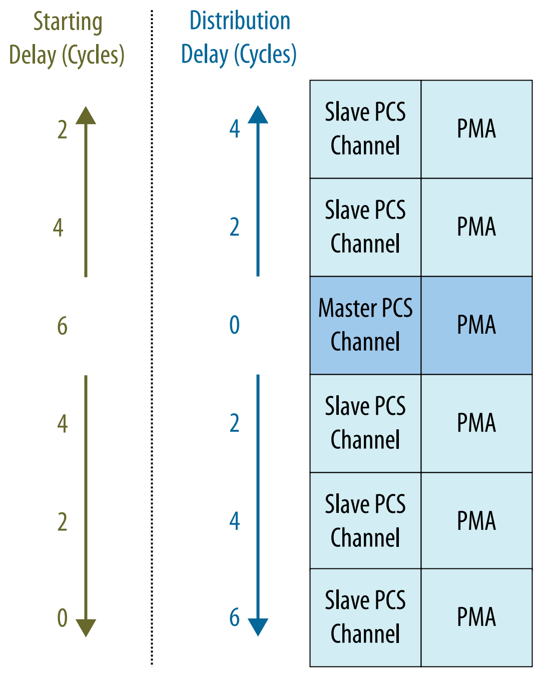
- x6 Clock Lines on page 284
- x24 Clock Lines on page 286
- Intel Stratix 10 Device Family Pin Connection Guidelines

3.9.2. PMA and PCS Bonding

PMA and PCS bonding reduces skew between both the PMA and PCS outputs within a group of channels.

For PMA bonding, either x6 or x24 is used. For PMA and PCS bonding, some of the PCS control signals within the bonded group are skew aligned using dedicated hardware inside the PCS.

Figure 160. PMA and PCS Bonding



PMA and PCS bonding use master and slave channels. One PCS channel in the bonded group is selected as the master channel and all others are slave channels. To ensure that all channels start transmitting data at the same time and in the same state, the master channel generates a start condition. This condition is transmitted to all slave channels. The signal distribution of this start condition incurs a two parallel clock cycle delay. Because this signal travels sequentially through each PCS channel, this delay is added per channel. The start condition used by each slave channel is delay compensated based on the slave channel's distance from the master channel. This results in all channels starting on the same clock cycle.

The transceiver PHY IP automatically selects the center channel to be the master PCS channel. This minimizes the total starting delay for the bonded group.

Note: Use the `tx_clkout` from the master channel as the source clock to drive the `tx_coreclkin` port for all other channels in the bonded interface.

Note: Because the PMA and PCS bonding signals travel through each PCS block, the PMA and PCS bonded groups must be contiguously placed. The channel order needs to be maintained when doing the pin assignments to the dedicated RX serial inputs and TX serial outputs (for example: PIN_BC7 and PIN_BC8 for GXBR4D_TX_CH0p and GXBR4D_TX_CH0n TX serial outputs). Channels need to be placed in an ascending order from bottom to top. Swapping of channels, when doing pin assignments, leads to errors.

3.9.3. Selecting Channel Bonding Schemes

In Intel Stratix 10 devices, select PMA and PCS bonding for bonded protocols that are explicitly supported by the hard PCS blocks. For example, PCI-Express, SFI-S, and 40GBASE-KR.

Select PMA-only bonding when a bonded protocol is not explicitly supported by the hard PCS blocks. For example, for Interlaken protocol, PMA-only bonding is used and a soft PCS bonding IP is implemented in the FPGA fabric.

3.9.4. Skew Calculations

To calculate the maximum skew between the channels, the following parameters are used:

- PMA to PCS datapath interface width (S)
- Maximum difference in number of parallel clock cycles between deassertion of each channel's FIFO reset (N).

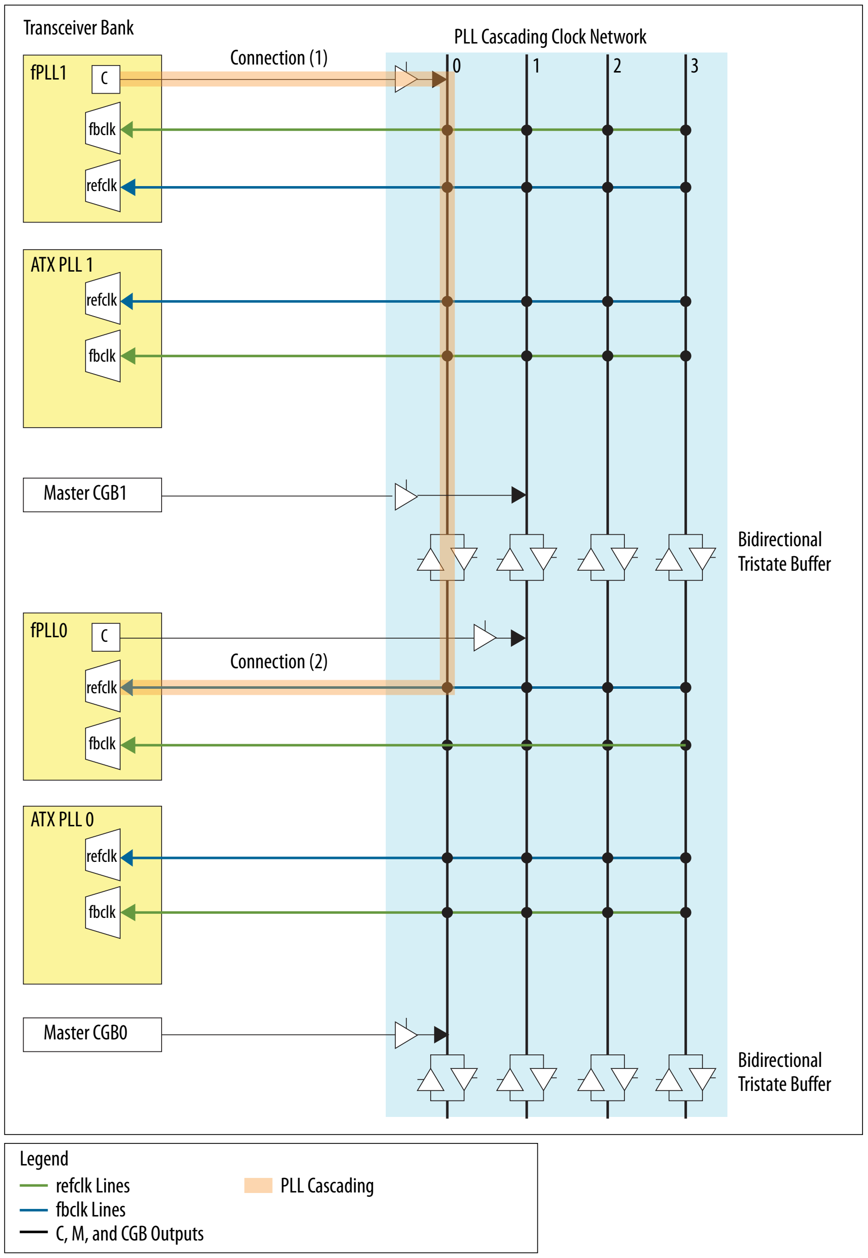
To calculate the channel skew, the following three scenarios are considered:

- Non-bonded—Both the PMA and PCS are non-bonded. Skew ranges from 0 UI to [(S-1) + N*S] UI.
- PMA bonding using x6 / x24 clock network—The PCS is non-bonded. Skew ranges from [0 to (N*S)] UI + x6/x24 clock skew.
- PMA and PCS bonding using the x6 / x24 clock network—Skew = x6 / x24 clock skew.

3.10. PLL Cascading Clock Network

The PLL cascading clock network spans the entire tile and is used for PLL cascading.

Figure 161. PLL Cascading Clock Network



To support PLL cascading, the following connections are present:

1. The C counter output of the fPLL drives the **cascading clock** network.
2. The **cascading clock** network drives the **reference clock** input of all PLLs.

For PLL cascading, connections (1) and (2) are used to connect the output of one PLL to the reference clock input of another PLL.

The transceivers in Intel Stratix 10 devices support fPLL to fPLL and ATX PLL to fPLL (via dedicated ATX PLL to fPLL cascade path) cascading.

In x24 bonding configurations, one PLL is used for each bonded group.

3.11. Using PLLs and Clock Networks

For L-Tile and H-Tile, PLLs are not integrated in the Native PHY IP core. You must instantiate the PLL IP cores separately. Unlike in some previous device families, PLL merging is no longer performed by the Intel Quartus Prime Pro Edition. This gives you more control, transparency, and flexibility in the design process. You can specify the channel configuration and PLL usage.

3.11.1. Non-bonded Configurations

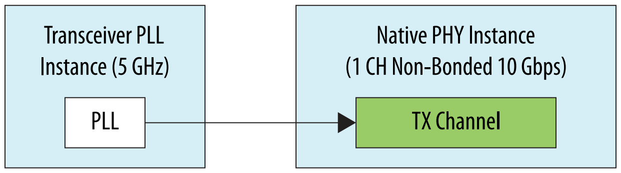
In a non-bonded configuration, only the high speed serial clock is routed from the transmitter PLL to the transmitter channel. The low speed parallel clock is generated by the local clock generation block (CGB) present in the transceiver channel. For non-bonded configurations, because the channels are not related to each other and the feedback path is local to the PLL, the skew between channels cannot be calculated. Also, the skew introduced by the clock network is not compensated.

3.11.1.1. Implementing Single Channel x1 Non-Bonded Configuration

In x1 non-bonded configuration, the PLL source is local to the transceiver bank and the x1 clock network is used to distribute the clock from the PLL to the transmitter channel.

For a single channel design, a PLL is used to provide the clock to a transceiver channel.

Figure 162. PHY IP Core and PLL IP Core Connection for Single Channel x1 Non-Bonded Configuration Example


Transceiver PLL Instance (5 GHz)
PLL
Native PHY Instance (1 CH Non-Bonded 10 Gbps)
TX Channel

Steps to implement a Single-Channel x1 Non-Bonded Configuration

1. Choose the PLL IP core (ATX PLL, fPLL, or CMU PLL) you want to instantiate in your design and instantiate the PLL IP core.
   - Refer to *Instantiating the ATX PLL IP Core*, or *Instantiating the fPLL IP Core*, or *Instantiating the CMU PLL IP Core* for detailed steps.
2. Configure the PLL IP core using the **IP Parameter Editor**
   - For the ATX PLL IP core or the fPLL IP core do not include the Master CGB.
   - For the CMU PLL IP core, specify the reference clock and the data rate. No special configuration rule is required.
3. Configure the Native PHY IP core using the **IP Parameter Editor**
   - Set the **Native PHY IP core TX Channel bonding mode** to **Non-Bonded**.
   - Set the number of channels as per your design requirement. In this example, the number of channels is set to 1.
4. Create a top level wrapper to connect the PLL IP core to the Native PHY IP core.

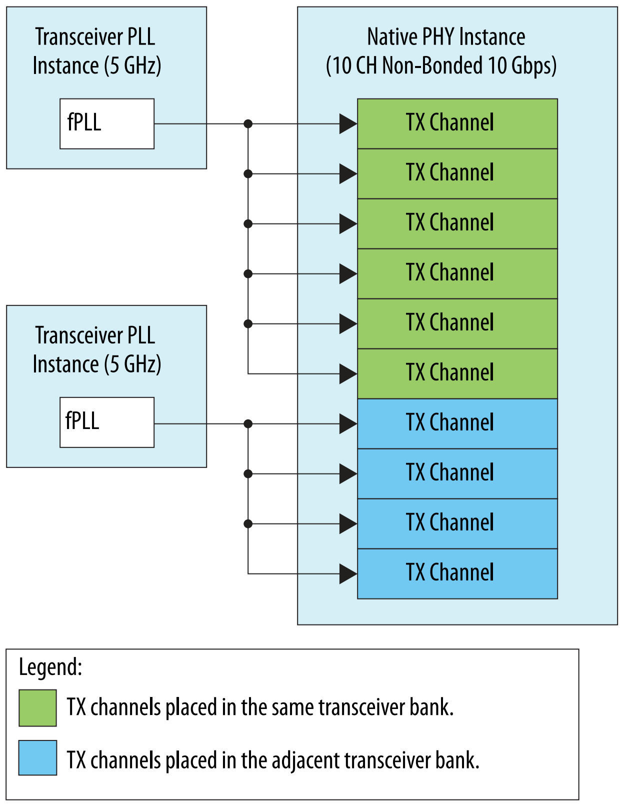
- The `tx_serial_clk output` port of the PLL IP core represents the high speed serial clock.
- The Native PHY IP core has 1 (for this example) `tx_serial_clk input` ports.
- As shown in the figure above, connect the `tx_serial_clk input` to the transceiver PLL instance.

3.11.1.2. Implementing Multi-Channel x1 Non-Bonded Configuration

This configuration is an extension of the x1 non-bonded case. In the following example, 10 channels are connected to two instances of the PLL IP core. Two PLL instances are required because PLLs using the x1 clock network can only span the 6 channels within the same transceiver bank. A second PLL instance is required to provide the clock to the remaining 4 channels.

Because 10 channels are not bonded and are unrelated, you can use a different PLL type for the second PLL instance. It is also possible to use more than two PLL IP cores and have different PLLs driving different channels. If some channels are running at different data rates, then you need different PLLs driving different channels.

Figure 163. PHY IP Core and PLL IP Core Connection for Multi-Channel x1 Non-Bonded Configuration


Transceiver PLL Instance (5 GHz)
fPLL
Transceiver PLL Instance (5 GHz)
fPLL
Native PHY Instance (10 CH Non-Bonded 10 Gbps)
TX Channel
Legend:
TX channels placed in the same transceiver bank.
TX channels placed in the adjacent transceiver bank.

Steps to implement a Multi-Channel x1 Non-Bonded Configuration:

1. Choose the PLL IP core (ATX PLL, fPLL, or CMU PLL) you want to instantiate in your design and instantiate the PLL IP core.

- Refer to *Instantiating the ATX PLL IP Core*, or *Instantiating the fPLL IP Core*, or *Instantiating the CMU PLL IP Core* for detailed steps.

2. Configure the PLL IP core using the **IP Parameter Editor**
    - For the ATX PLL IP core do not include the Master CGB. If your design uses the ATX PLL IP core and more than 6 channels, the x1 Non-Bonded Configuration is not a suitable option. Multi-channel x24 Non-Bonded are the required configurations when using the ATX PLL IP core and more than 6 channels in the Native PHY IP core.
    - For the CMU PLL IP core, specify the reference clock and the data rate. No special configuration rule is required.
3. Configure the Native PHY IP core using the **IP Parameter Editor**
    - Set the **Native PHY IP core TX Channel bonding mode** to **Non-Bonded**.
    - Set the number of channels as per your design requirement. In this example, the number of channels is set to 10.
4. Create a top level wrapper to connect the PLL IP core to the Native PHY IP core.
    - The `tx_serial_clk output` port of the PLL IP core represents the high speed serial clock.
    - The Native PHY IP core has 10 (for this example) `tx_serial_clk input` ports. Each port corresponds to the input of the local CGB of the transceiver channel.
    - As shown in the figure above, connect the first 6 `tx_serial_clk input` to the first transceiver PLL instance.
    - Connect the remaining 4 `tx_serial_clk input` to the second transceiver PLL instance.

Related Information

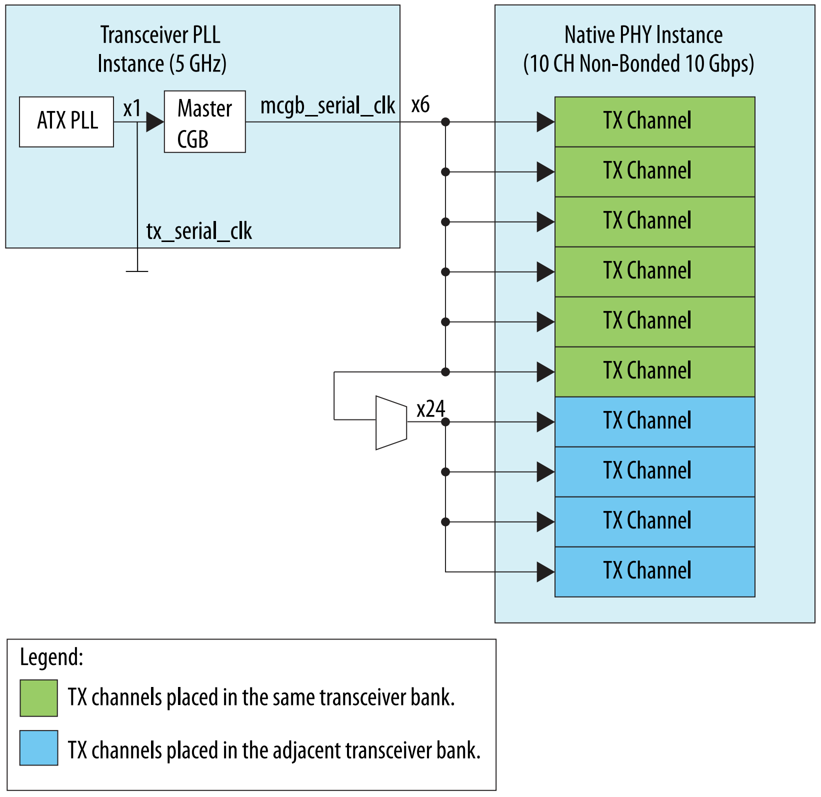
- Instantiating the ATX PLL IP Core on page 260
- Instantiating the fPLL IP Core on page 268
- Instantiating CMU PLL IP Core on page 275

3.11.1.3. Implementing Multi-Channel x24 Non-Bonded Configuration

Using the x24 non-bonded configuration reduces the number of PLL resources and the reference clock sources used.

Figure 164. PHY IP Core and PLL IP Core Connection for Multi-Channel x24 Non-Bonded Configuration

In this example, the same PLL is used to drive 10 channels across two transceiver banks.


Transceiver PLL Instance (5 GHz)
ATX PLL
x1
Master CGB
mcgb_serial_clk
x6
tx_serial_clk
x24
Native PHY Instance (10 CH Non-Bonded 10 Gbps)
TX Channel
Legend:
TX channels placed in the same transceiver bank.
TX channels placed in the adjacent transceiver bank.

Steps to implement a multi-channel x24 non-bonded configuration

1. You can use either the ATX PLL or fPLL for multi-channel x24 non-bonded configuration.
   - Refer to *Instantiating the ATX PLL IP Core* or *Instantiating the fPLL IP Core* for detailed steps.
   - Only the ATX PLL or fPLL can be used for this example, because the CMU PLL cannot drive the master CGB.
2. Configure the PLL IP core using the **IP Parameter Editor**. Enable **Include Master Clock Generation Block** .
3. Configure the Native PHY IP core using the **IP Parameter Editor**
   - Set the **Native PHY IP core TX Channel bonding mode** to **Non-Bonded** .
   - Set the number of channels as per your design requirement. In this example, the number of channels is set to 10.
4. Create a top level wrapper to connect the PLL IP core to the Native PHY IP core.

- In this case, the PLL IP core has `mcgb_serial_clk` output port. This represents the x24 clock line.
- The Native PHY IP core has 10 (for this example) `tx_serial_clk input` ports. Each port corresponds to the input of the local CGB of the transceiver channel.
- As shown in the figure above, connect the `mcgb_serial_clk` output port of the PLL IP core to the 10 `tx_serial_clk input` ports of the Native PHY IP core.
- Leave the PLL IP's `tx_serial_clk output` port unconnected.

Related Information

- Instantiating the ATX PLL IP Core on page 260
- Instantiating the fPLL IP Core on page 268

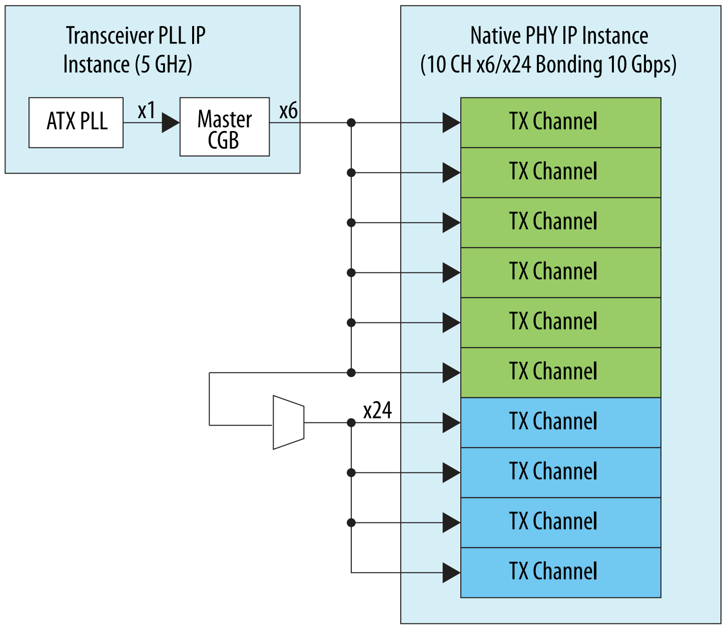
3.11.2. Bonded Configurations

In a bonded configuration, both the high speed serial and low speed parallel clocks are routed from the transmitter PLL to the transmitter channel. In this case, the local CGB in each channel is bypassed and the parallel clocks generated by the master CGB are used to clock the network.

In bonded configurations, the transceiver clock skew between the channels is minimized. Use bonded configurations for channel bonding to implement protocols such as PCIe and Interlaken.

3.11.2.1. Implementing x6/x24 Bonding Mode

Figure 165. Connection between Native PHY IP and PLL IP Cores for x6/x24 Bonding Mode



Legend:

🟩 TX channels placed in the same transceiver bank.

🟦 TX channels placed in the adjacent transceiver bank.

Note: Although the above diagram looks similar to the "Multi-Channel x1/x24 Non-Bonded Example", the clock input ports on the transceiver channels bypass the local CGB in x6/x24 bonding configuration. This internal connection is taken care of when the **Native PHY channel bonding mode** is set to **Bonded**.

Steps to implement a x6/x24 bonded configuration

1. You can instantiate either the ATX PLL or the fPLL for x6/x24 bonded configuration.
   - Refer to *Instantiating the ATX PLL IP Core* or *Instantiating the fPLL IP Core* for detailed steps.
   - Only the ATX PLL or fPLL can be used for bonded configurations, because the CMU PLL cannot drive the Master CGB.
2. Configure the PLL IP core using the **IP Parameter Editor**. Enable **Include Master Clock Generation Block** and **Enable bonding** clock output ports.
3. Configure the Native PHY IP core using the **IP Parameter Editor**.

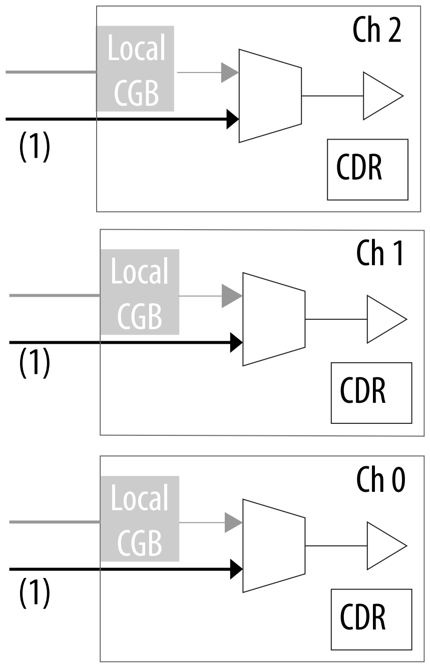
- Set the **Native PHY IP core TX Channel bonding mode** to either **PMA bonding** or **PMA/PCS bonding**.

  Note: All channels must be contiguously placed when using PMA/PCS bonding. Refer to the "Channel Bonding" section for more details.
- Set the number of channels required by your design. In this example, the number of channels is set to 10.

4. Create a top level wrapper to connect the PLL IP core to Native PHY IP core.
   - In this case, the PLL IP core has `tx_bonding_clocks` output bus with width [5:0].
   - The Native PHY IP core has `tx_bonding_clocks` input bus with width [5:0] multiplied by the number of transceiver channels (10 in this case). For 10 channels, the bus width is [59:0].

     Note: While connecting `tx_bonding_clocks`, leave `tx_serial_clk` open to avoid any Intel Quartus Prime Pro Edition software fitter errors.
   - Connect the PLL IP core to the PHY IP core by duplicating the output of the PLL[5:0] for the number of channels. For 10 channels, the Verilog syntax for the input port connection is `.tx_bonding_clocks ({10{tx_bonding_clocks_output}})`.

Figure 166. x6/x24 Bonding Mode —Internal Channel Connections



Note: (1) The local CGB is bypassed by the clock input ports in bonded mode.

Related Information

- x24 Clock Lines on page 286
- Instantiating the ATX PLL IP Core on page 260

- Instantiating the fPLL IP Core on page 268
- Implementing Multi-Channel x24 Non-Bonded Configuration on page 306
- Channel Bonding on page 299

3.11.3. Implementing PLL Cascading

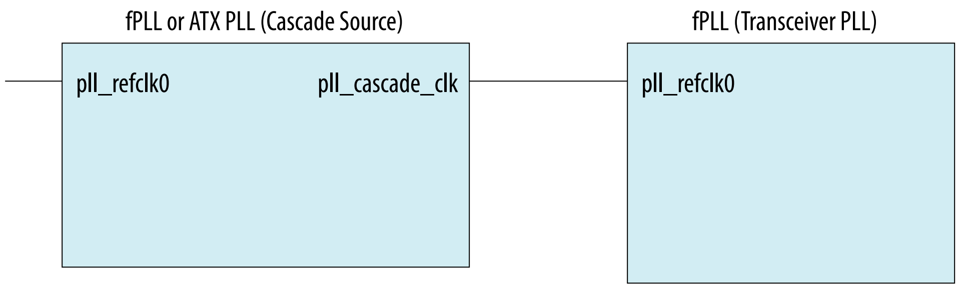
In PLL cascading, the output of the first PLL feeds the input reference clock to the second PLL.

For example, if the input reference clock has a fixed frequency, and the desired data rate was not an integer multiple of the input reference clock, the first PLL can be used to generate the correct reference clock frequency. This output is fed as the input reference clock to the second PLL. The second PLL generates the clock frequency required for the desired data rate.

The transceivers in Intel Stratix 10 devices support fPLL to fPLL and ATX PLL to fPLL cascading. The first PLL (cascade source) and second PLL (downstream PLL) have to be in the same 24-channel tile. For OTN and SDI applications, there is a dedicated clock path for cascading ATX PLL to fPLL.

Note: When the fPLL is used as a cascaded fPLL (downstream fPLL), a user recalibration on the fPLL is required. Refer to *User Recalibration* section for more information.

Figure 167. PLL Cascading



Steps to implement fPLL to fPLL cascading:

1. Instantiate the fPLL IP core.
2. Set the following configuration settings for the fPLL IP core in the **Parameter Editor**:
   - Set the **fPLL Mode** to **Cascade Source**.
   - Set the **Desired output clock frequency**.
3. Instantiate the fPLL IP core (the second PLL in PLL cascading configuration). Refer to *Instantiating the fPLL IP Core* for detailed steps.
4. Configure the second fPLL IP core for the desired data rate and the reference clock frequency. Set reference clock frequency for the second fPLL same as the output frequency of the first fPLL.
5. Connect the fPLL IP core (cascade source) to fPLL IP core (transceiver PLL) as shown in the above figure. Ensure the following connections:

- The fPLL has an output port `pll_cascade_clk`. Connect this port to the second fPLL's `pll_refclk0` port.

6. If the input reference clock is available at device power-up, the first PLL will be calibrated during the power-up calibration. The second PLL needs to be recalibrated. If the input reference clock is not available at device power-up, then re-run the calibration for the first PLL. After the first PLL has been properly calibrated, re-calibrate the second PLL.

Notes:

- No special configuration is required for the Native PHY instance.
- ATX PLL to fPLL cascading mode is added to address the OTN and SDI jitter requirement. In this mode, ATX PLL generates a relatively high and clean reference frequency in fractional mode. The reference is driving the fPLL, which is running in integer mode. Overall cascaded two PLLs, synthesize a needed frequency for a given data rate.
- You may use this configuration to generate clock frequencies that cannot be generated by a single PLL. It is most commonly used for OTN/SDI applications.

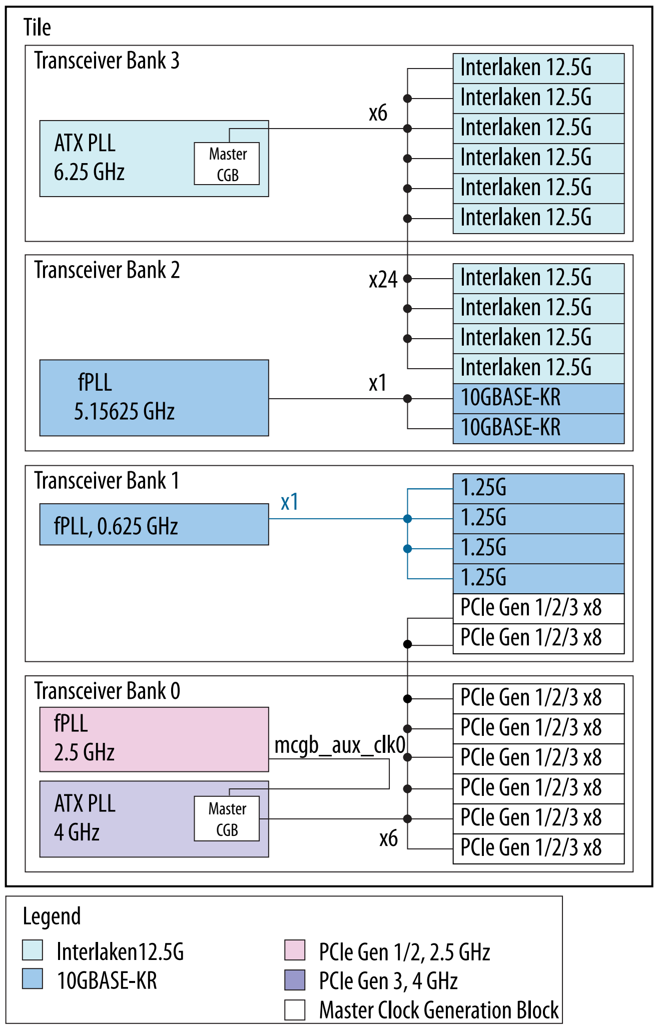
Related Information

- User Recalibration on page 444
- Instantiating the fPLL IP Core on page 268

3.11.4. Mix and Match Example

In the Intel Stratix 10 transceiver architecture, the separate Native PHY IP core and the PLL IP core scheme allows great flexibility. It is easy to share PLLs and reconfigure data rates. The following design example illustrates PLL sharing and both bonded and non-bonded clocking configurations.

Figure 168. Mix and Match Design Example



PLL Instances

In this example, two ATX PLL instances and three fPLL instances are used. Choose an appropriate reference clock for each PLL instance. The **IP Catalog** lists the available PLLs.

Use the following data rates and configuration settings for PLL IP cores:

- Transceiver PLL Instance 0: ATX PLL with output clock frequency of 6.25 GHz
  - Enable the Master CGB and bonding output clocks.
- Transceiver PLL instance 1: fPLL with output clock frequency of 5.1625 GHz
  - Select the **Use as Transceiver PLL** option.
- Transceiver PLL instance 2: fPLL with output clock frequency of 0.625 GHz
- Transceiver PLL instance 3: fPLL with output clock frequency of 2.5 GHz
  - Select **Enable PCIe clock output port** option.
  - Select **Use as Transceiver PLL** option.
    - Set **Protocol Mode** to **PCIe Gen2**.
- Transceiver PLL instance 4: ATX PLL with output clock frequency of 4 GHz
  - Enable Master CGB and bonding output clocks.
  - Select **Enable PCIe clock switch interface** option.
  - Set **Number of Auxiliary MCGB Clock Input ports** to 1.

Native PHY IP Core Instances

In this example, three Transceiver Native PHY IP core instances and two 10GBASE-KR PHY IP instances are used. Use the following data rates and configuration settings for the PHY IPs:

- 12.5 Gbps Interlaken with a bonded group of 10 channels
  - Select the **Interlaken 10x12.5 Gbps** preset from the Intel Stratix 10 Transceiver Native PHY IP core GUI.
- 1.25 Gbps Gigabit Ethernet with a non-bonded group of four channels
  - Select the **GIGE-1.25Gbps** preset from the Intel Stratix 10 Transceiver Native PHY IP core GUI.
  - Change the **Number of data channels** to 2.
- PCIe Gen3 with a bonded group of 8 channels
  - Select the **PCIe PIPE Gen3x8** preset from the Intel Stratix 10 Transceiver Native PHY IP core GUI.
  - Under **TX Bonding options**, set the **PCS TX channel bonding master** to channel 5.

    Note: The PCS TX channel bonding master must be physically placed in channel 1 or channel 4 within a transceiver bank. In this example, the 5th channel of the bonded group is physically placed at channel 1 in the transceiver bank.
  - Refer to *PCI Express* (PIPE) for more details.
- 10.3125 Gbps 10GBASE-KR non-bonded group of 2 channels
  - Instantiate the Intel Stratix 10 1G/10GbE and 10GBASE-KR PHY IP two times, with one instance for each channel.
  - Refer to *10GBASE-KR PHY IP Core* for more details.

Connection Guidelines for PLL and Clock Networks

- For 12.5 Gbps Interlaken with a bonded group of 10 channels, connect the `tx_bonding_clocks` to the transceiver PLL's `tx_bonding_clocks` output port. Make this connection for all 10 bonded channels. This connection uses a master CGB and the x6 / x24 clock line to reach all the channels in the bonded group.
- Connect the `tx_serial_clk` port of the two instances of the 10GBASE-KR PHY IP to the `tx_serial_clk` port of PLL instance 1 (fPLL at 5.1625 GHz). This connection uses the x1 clock line within the transceiver bank.
- Connect the 1.25 Gbps Gigabit Ethernet non-bonded PHY IP instance to the `tx_serial_clk` port of the PLL instance 2. Make this connection twice, one for each channel. This connection uses the x1 clock line within the transceiver bank.
- Connect the PCIe Gen3 bonded group of 8 channels as follows:
  - Connect the `tx_bonding_clocks` of the PHY IP to the `tx_bonding_clocks` port of the Transceiver PLL Instance 4. Make this connection for each of the 8 bonded channels.
  - Connect the `pipe_sw_done` of the PHY IP to the `pipe_sw` port of the transceiver PLL instance 4.
  - Connect the `pll_pcie_clk` port of the PLL instance 3 to the PHY IP's `pipe_hclk_in` port.
  - Connect `tx_serial_clk` port of the PLL instance 3 to the `mcgb_aux_clk0` port of the PLL instance 4. This connection is required as a part of the PCIe speed negotiation protocol.

Related Information

PCI Express (PIPE) on page 164

3.12. PLLs and Clock Networks Revision History

Document Version	Changes
2020.03.03	Made the following changes: • Updated the following figures to make it clear that `rx_clkout` is driven by CDR. — FPGA Fabric—Transceiver Interface Clocking (Standard PCS Example) — Transmitter Standard PCS and PMA Clocking — Transmitter Enhanced PCS and PMA Clocking — Receiver Standard PCS and PMA Clocking — Receiver Enhanced PCS and PMA Clocking • Clarified *ATX PLL to fPLL Spacing Requirements*. • Clarified that the reference clock for GXT channels must be located in the same triplet as the master ATX PLL. • Added this note, "Use the `tx_clkout` from the master channel as the source clock to drive the `tx_coreclkin` port for all other channels in the bonded interface." • Added *HCLK Network*. • In *fPLL IP Core - Parameters, Settings, and Ports*, clarified that fPLL in **Core** mode does not support the dynamic reconfiguration feature.
2019.03.22	Made the following change: • Added SATA GEN3 and HDMI to the Protocol Mode Range in the "fPLL IP Core - Configuration Options, Parameters, and Settings" table.
2019.01.23	Made the following changes:

continued...

Document Version	Changes
	• Added a new requirement: The reference clock for GXT channels must be located in the same triplet as the master ATX PLL.
2018.10.05	Made the following changes: • Added GXT Clock buffers to the "Main and Adjacent ATX PLL IP Instances to Drive 6 GXT Channels" figure. • Removed the ATX PLL to ATX PLL spacing requirement from the "ATX PLL Spacing Requirements" table. • Updated the note in the "Input Reference Clock Sources" section. • Added a note to the "x6/x24 Bonding" section.
2018.10.04	Made the following changes: • Added "even if the VCCR_GXB and VCCT_GXB operating voltages of the banks in the tile are different" to *Reference Clock Network*. • Added the following to *x24 Clock Lines*: A maximum of 24 channels can be used in a single bonded or non-bonded x24 group. When the banks within a transceiver tile are powered at different voltages (for example, some banks are operating at 1.03 V while other banks are operating at 1.12 V), the x24 clock lines are only allowed to traverse between contiguous banks operating at the same VCCR_GXB and VCCT_GXB voltages. The x24 clock lines crossing boundaries of banks operating at different voltages is not allowed. See the Intel Stratix 10 Device Family Pin Connection Guidelines for a description of the transceiver power connection guidelines.
2018.07.06	Made the following changes: • Added a note to *Using the ATX PLL for GXT Channels*: An ATX PLL cannot be reconfigured from GX to GXT mode if the adjacent master CGB is being used. • Clarified reference clock and transmitter PLL location awareness in *Reference Clock Network*. • Added note to *Clock Generation Block*: If you are using a master CGB, do not configure the adjacent ATX PLL from a GX to a GXT mode. • For ATX PLL, fPLL, and CMU PLL IP Core - Parameters, Settings, and Ports, added "The ports related to reconfiguration are compliant with the *Avalon Specification*. Refer to the *Avalon Specification* for more details about these ports" to the *Avalon Specification* link.
2018.03.16	Made the following changes: • Grayed the CGB blocks in "x6/x24 Bonding Mode —Internal Channel Connections" figure in "Implementing x6/x24 Bonding Mode" topic. • Updated "five fPLL instances" to 3 in the description of "PLL Instances" section. • Changed the description of the following parameters and ports in the "ATX PLL IP Core - Parameters, Settings, and Ports" section: — **Enable mcgb_rst and mcgb_rst_stat ports** — `mcgb_rst` — `mcgb_rst_stat` • Updated the data rates and configuration settings for PLL IP Cores and PHY IPs in "Mix and Match Example" section. Also updated from "four Transceiver Native PHY IP core instances and four 10GBASE-KR PHY IP instances" to 3 Native PHY IP core instances and 2 10GBASE-KR PHY IP instances. • Added "Lock Detector" block description for fPLL. • Note "All channels must be contiguously placed when using PMA/PCS bonding. Refer to the "Channel Bonding" section for more details." has been added in the "Implementing x6/x24 Bonding Mode" section. • Removed Clock Cascading Inputs for "Dedicated Reference Clock Pins" figure. • Updated term "TX phase compensation FIFO" to "TX PCS FIFO" in "Transmitter Data Path Interface Clocking" topic. • Added steps to implement a Single-Channel x1 Non-Bonded Configuration. • Added a note for PLL cascading "You may use this configuration to generate clock frequencies that cannot be generated by a single PLL. It is most commonly used for OTN/SDI applications." • Added a note "Each core clock network reference clock pin cannot drive fPLLs located on multiple L/H-Tiles". • Title of the figure changed from "FPGA Fabric—Transceiver Interface Clocking" to "FPGA Fabric—Transceiver Interface Clocking (Standard PCS Example)".

continued...

Document Version	Changes
	• Added the sentence "For PCS Direct, the clocking architecture remains the same as Standard PCS" in "Transmitter Data Path Interface Clocking". • Changed the note in the "ATX PLL Spacing Requirements" section. • Re-organized "Dedicated Reference Clock Pins" and updated to current Intel Quartus Prime Pro Edition names and made QSF edits for the transceiver `refclk`. • Added "Leave the PLL IP's tx_serial_clk output port unconnected" to "Implementing Multi-Channel x24 Non-Bonded Configuration." • Clarified the "Multi-Channel x1/x24 Non-Bonded Example" figure. • Changed L-Tile max data rate to 26.6 in "Transmit PLL Recommendation Based on Data Rates." • Clarified the "ATX PLL Spacing Requirements" table. • Changed GT to GXT in "ATX PLL GXT Clock Connection." • Added signal names to "PHY IP Core and PLL IP Core Connection for Multi-Channel x24 Non-Bonded Configuration." • Changed "adjacent" to "clock buffer" in "Using the ATX PLL for GXT Channels" and "GXT Implementation Usage Restrictions for ATX PLL GX & MCGB." • Updated "ATX PLL IP Parameter Details for Clock Buffer ATX PLL IP" and "ATX PLL IP Parameter Details for Main ATX PLL IP" figures. • Removed L-Tile from "GXT Clock Network." • Made rx_clkout and tx_clkout able to drive both Dedicated and Global Core Clock Networks for the "FPGA Fabric—Transceiver Interface Clocking (Standard PCS Example)" figure. • Changed support to fPLL to fPLL and ATX PLL to fPLL only for "PLL Cascading Clock Network."
2017.08.11	Made the following changes: • Added a note in "Dedicated Reference Clock Pins" topic stating "The reference clock pins use thick oxide and are thus safe from damage due to hot swapping". • Added a paragraph in the Reference Clock Network section : "You can only use the two high quality reference clock lines for one bottom and one top reference clock in a tile. There are fitter errors if you try to use both lines for two bottom reference clocks in a tile." • Added a note "For L-Tile, you can only have 4 GXT per tile, all in the same bank".
2017.06.06	Made the following changes: • Feedback compensation bonding is not supported. • Updated "ATX PLL Spacing Requirements" table. • Added a new section "GXT Implementation Usage Restrictions for ATX PLL GX & MCGB". • Updated "Reference Clock Network" section. • Added a new figure "Main and Adjacent ATX PLL IP Instances to Drive 6 GXT Channels". • Updated topic "Timing Closure Recommendations".
2017.03.08	Made the following changes: • Changed all the notes in the "Using the ATX PLL for GXT Channels" section.
2017.02.17	Made the following changes: • Updated the ATX PLL description to "ATX PLL only supports fractional mode". • Updated the L Counter description to "The division factor supported are 1 and 2". • Updated the Receiver Input Pins description to "Receiver input pins can be used as an input reference clock source to transceiver PLLs. However, they cannot be used to drive core fabric". • Added new section "ATX PLL Spacing Requirements". • Added new section "Using the ATX PLL for GXT Channels". • Added the following note in the relevant topics: "When the fPLL is used as a cascaded fPLL (downstream fPLL), a user recalibration on the fPLL is required. Refer to "User Recalibration" section in "Calibration" chapter for more information." • Following parameters are added in the fPLL IP Core parameters table: "Message level for rule violations", "Enable /1 output clock", "Enable /2 output clock", "Enable /4 output clock", "PLL integer/fractional reference clock frequency" and "Enable mcgb_rst and mcgb_rst_stat ports".
2016.12.21	Initial release

4. Resetting Transceiver Channels

To ensure that transceiver channels are ready to transmit and receive data, you must properly reset the transceiver PHY. Intel recommends a reset sequence that ensures the physical coding sublayer (PCS) and physical medium attachment (PMA) in each transceiver channel initialize and function correctly. You can either use the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP or create your own reset controller.

4.1. When Is Reset Required?

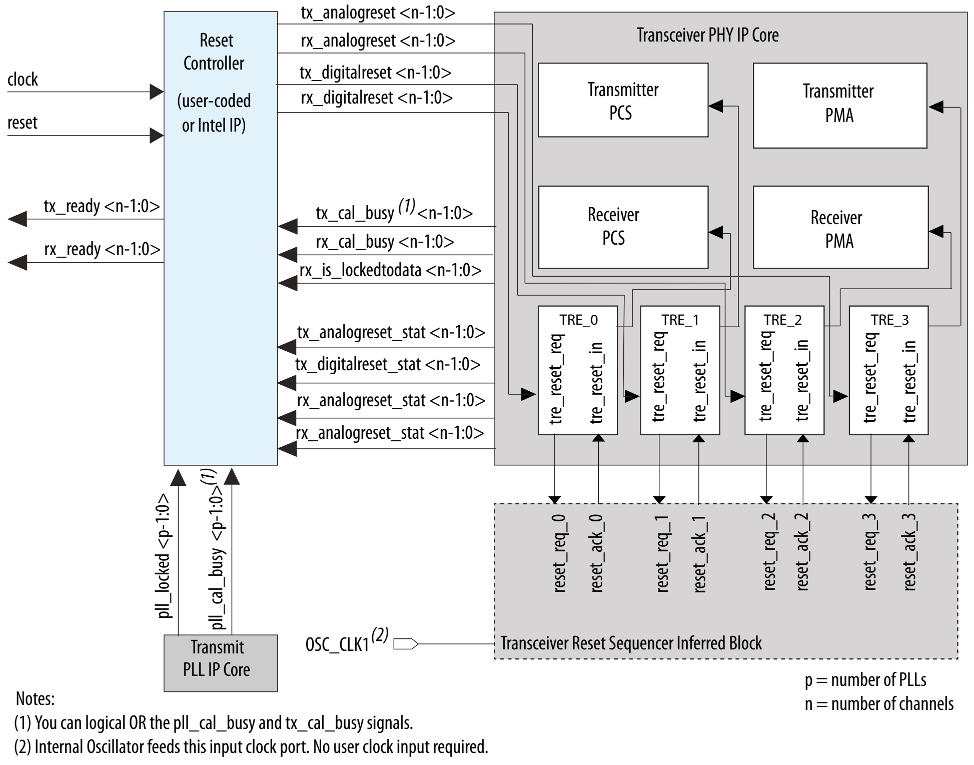
You can reset the transmitter (TX) and receiver (RX) data paths independently or together. The recommended reset sequence requires reset and initialization of the PLL driving the TX or RX channels, as well as the TX and RX datapaths. A reset is required after any of the following events:

Table 143. Reset Conditions

Event	Reset Requirement
Device power up and configuration	Requires reset to the transceiver PHY.
PLL reconfiguration	Requires reset to the PHY. The transmitter channel must be held in reset before performing PLL reconfiguration.
PLL reference clock frequency change	Requires reset to the PHY.
PLL recalibration	The transmitter channel must be held in reset before performing PLL recalibration.
PLL lock loss or recovery	Requires reset to the PHY.
Channel dynamic reconfiguration	Requires holding the channel in reset before performing a dynamic reconfiguration that causes rate change.
Optical module connection	Requires reset of RX to ensure lock of incoming data.
RX CDR lock mode change	Requires reset of the RX channel any time the RX clock and data recovery (CDR) block switches from lock-to-reference to lock-to-data RX channel.

4.2. Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP Implementation

Figure 169. Reset Controller, Transceiver PHY and TX PLL IP Cores Interaction



Transceiver Reset Endpoints—The Transceiver PHY IP core contains Transceiver Reset Endpoints (TREs). The analog and digital reset ports (both TX/RX) of the Transceiver Native PHY IP core are connected to the input of the TX TRE and RX TRE, respectively.

Transceiver Reset Sequencer—The Intel Quartus Prime Pro Edition software detects the presence of TREs and automatically inserts only one Transceiver Reset Sequencer (TRS)[47]. The `tx_digitalreset`, `rx_digitalreset`, `tx_analogreset` and `rx_analogreset` requests from the reset controller (user-coded or Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP) are received by the TREs. The TRE sends the reset request to the TRS for scheduling. TRS schedules all the requested PCS/PMA resets and provides acknowledgment for each request. You can use either Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP or your own reset controller. However, for the TRS to work correctly, the required timing duration must be followed.

Related Information

Resetting Transceiver Channels on page 318

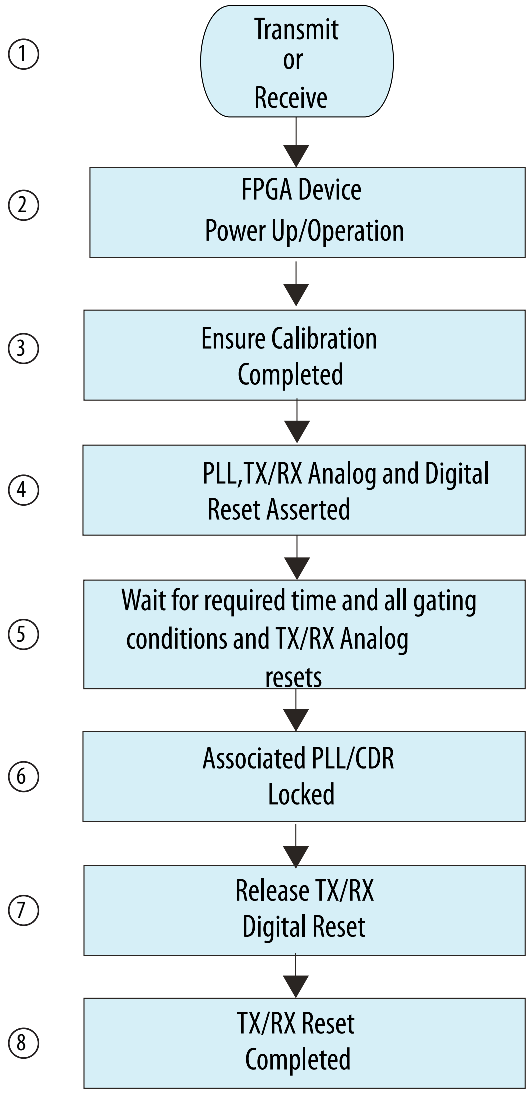
[47] There is only one centralized TRS instantiated for one or more Native PHY. The TRS IP is an inferred block and is not visible in the RTL. You have no control over this block.

4.3. How Do I Reset?

You reset a transceiver PHY by integrating a reset controller in your system design to initialize the PCS and PMA blocks. You can save time by using the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP, or you can implement your own reset controller that follows the recommended reset sequence. You can design your own reset controller if you require individual control of each signal for reset or need additional control or status signals as part of the reset functionality.

4.3.1. Recommended Reset Sequence

Figure 170. Transmitter and Receiver Reset Sequence


1
Transmit or Receive
2
FPGA Device Power Up/Operation
3
Ensure Calibration Completed
4
PLL,TX/RX Analog and Digital Reset Asserted
5
Wait for required time and all gating conditions and TX/RX Analog resets
6
Associated PLL/CDR Locked
7
Release TX/RX Digital Reset
8
TX/RX Reset Completed

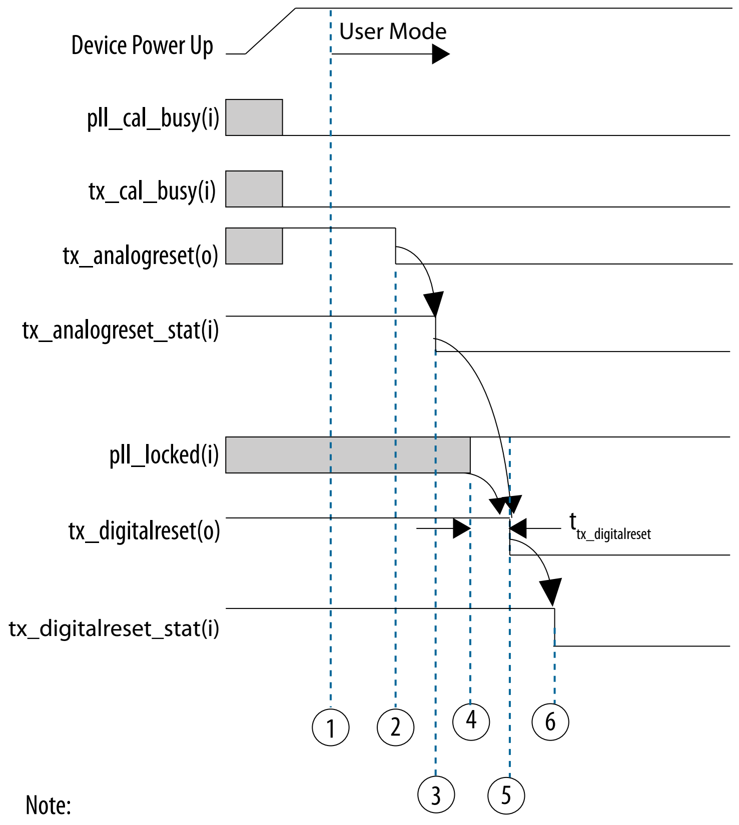
4.3.1.1. Resetting the Transmitter After Power Up

The FPGA automatically calibrates the PLL at every power-up before entering user-mode. Perform a reset sequence after the device enters the user-mode. Your user coded Reset Controller must comply with the reset sequence below to ensure a reliable transmitter initialization after the initial power-up calibration.

The step numbers in this list correspond to the numbers in the following figure.

1. Deassert `tx_analogreset` after the device enters user mode for a minimum duration of 2 ms. The **CONF_DONE** pin is asserted when the device enters user mode.
2. Wait for `tx_analogreset_stat` signal from the PHY to deassert, to ensure that `tx_analogreset` deasserts successfully.
3. Wait for `pll_locked` to assert.
4. Deassert `tx_digitalreset` after the `pll_locked` stays asserted for a minimum duration of $t_{tx\_digitaltreset}$.
5. Wait for `tx_digitalreset_stat` signal from the PHY, to deassert, to ensure that `tx_digitalreset` deasserts successfully.

Figure 171. Transmitter Power Up Sequence During device Operation



Note:

(1) Area in gray is don't care zone.

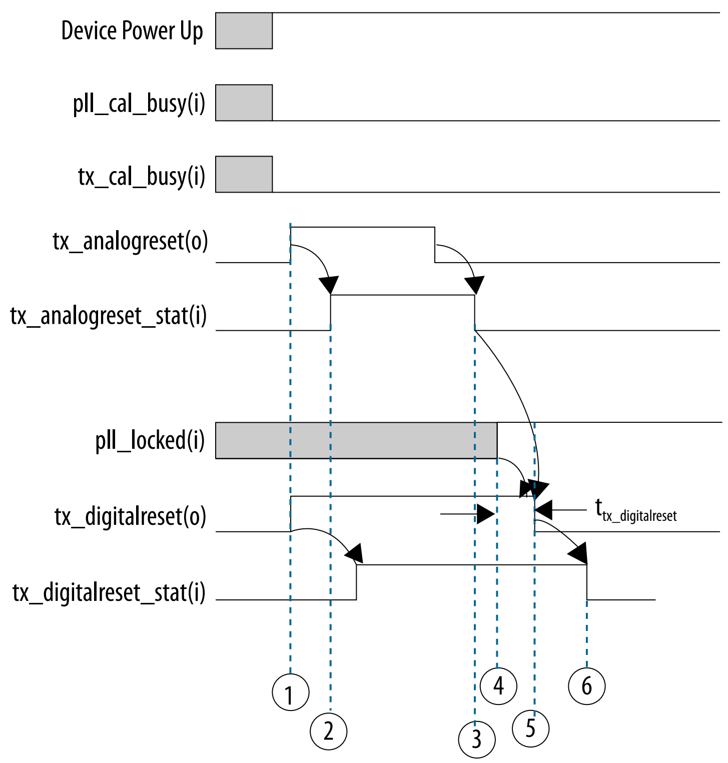
4.3.1.2. Resetting the Transmitter During Device Operation

Follow this reset sequence to reset the analog or digital blocks of the transmitter at any point during the device operation. Use this reset to re-establish a link. Your user coded Reset Controller must comply with the reset sequence below to ensure a reliable transmitter operation.

The step numbers in this list correspond to the numbers in the following figure.

1. Assert `tx_analogreset` and `tx_digitalreset`, while `pll_cal_busy` and `tx_cal_busy` are low.
2. Wait for `tx_analogreset_stat` from the PHY to assert, to ensure that `tx_analogreset` asserts successfully. `tx_analogreset_stat` goes high when TX PMA has been successfully held in reset.
    a. Deassert `tx_analogreset`.
3. Wait for `tx_analogreset_stat` from the PHY, to deassert, to ensure that `tx_analogreset` deasserts successfully. `tx_analogreset_stat` goes low when TX PMA has been successfully released out of reset.
4. The `pll_locked` signal goes high after the TX PLL acquires lock. Wait for `tx_analogreset_stat` to deassert before monitoring the `pll_locked` signal.
5. Deassert `tx_digitalreset` a minimum $\texttt{t}_{\texttt{tx\_digitalreset}}$ time after `pll_locked` goes high.
6. Wait for `tx_digitalreset_stat` from the PHY, to deassert, to ensure that `tx_digitalreset` deasserts successfully in the PCS.

Figure 172. Transmitter Reset Sequence During Device Operation



Note:

(1) Area in gray is don't care zone.

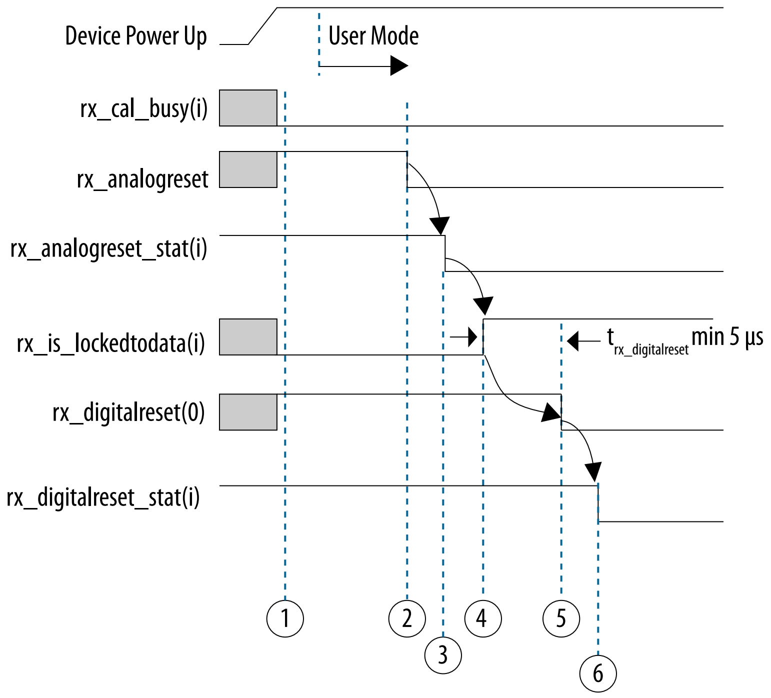
4.3.1.3. Resetting the Receiver After Power UP

The FPGA automatically calibrates the PLL at every power-up before entering user-mode. Perform a reset sequence after the device enters the user-mode. Your user coded Reset Controller must comply with the reset sequence below to ensure a reliable transmitter initialization after the initial power-up calibration.

The step numbers in this list correspond to the numbers in the following figure.

1. Deassert `rx_analogreset` after a minimum duration of $t_{rx\_analogreset}$ after the device enters user mode. The **CONF_DONE** pin is asserted when the device enters user mode.
2. Wait for `rx_analogreset_stat` signal from the PHY, to deassert, to ensure that `rx_analogreset` deasserts successfully.
3. Wait for `rx_is_lockedtodata` to assert.
4. Deassert `rx_digitalreset` after the `rx_is_lockedtodata` stays asserted for a minimum duration of t_{LTD} of 5us. If the `rx_is_lockedtodata` is asserted and toggles, you must wait another additional t_{LTD} duration before deasserting `rx_digitalreset`.
5. Wait for `rx_digitalreset_stat` signal from the PHY, to deassert, to ensure that `rx_digitalreset` deasserts successfully.

Figure 173. Resetting the Receiver After Power Up



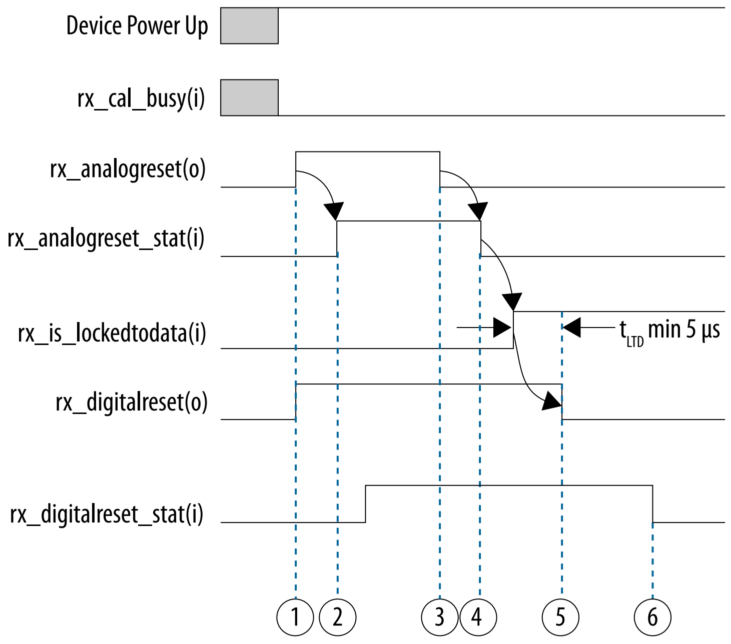
4.3.1.4. Resetting the Receiver During Device Operation (Auto Mode)

Follow this reset sequence to reset the analog or digital blocks of the receiver at any point during the device operation. Use this reset to re-establish a link. Your user coded Reset Controller must comply with the reset sequence below to ensure a reliable receiver operation.

The step numbers in this list correspond to the numbers in the following figure.

1. Assert `rx_analogreset` and `rx_digitalreset` while `rx_cal_busy` is low.
2. Wait for `rx_analogreset_stat` to assert, to ensure that `rx_analogreset` asserts successfully. `rx_analogreset_stat` goes high when RX PMA has been successfully held in reset.
   a. Deassert `rx_analogreset`.
3. Wait for `rx_analogreset_stat` to deassert, to ensure that `rx_analogreset` deasserts successfully. `rx_analogreset_stat` goes low when RX PMA has been successfully released out of reset.
4. The `rx_is_lockedtodata` signal goes high after the CDR acquires lock.
5. Ensure `rx_is_lockedtodata` is asserted for t_{LTD} (minimum of 5 µs) before deasserting `rx_digitalreset`.
6. Wait for `rx_digitalreset_stat` from the PHY, to deassert, to ensure that `rx_digitalreset` deasserts successfully in the PCS.

Figure 174. Receiver Reset Sequence During Device Operation



4.3.1.5. Clock Data Recovery in Manual Lock Mode

Use the clock data recovery (CDR) manual lock mode to override the default CDR automatic lock mode depending on your design requirements.

4.3.1.5.1. Control Settings for CDR Manual Lock Mode

Use the following control settings to set the CDR lock mode:

Table 144. Control Settings for the CDR in Manual Lock Mode

rx_set_locktoref	rx_set_locktodata	CDR Lock Mode
0	0	Automatic
1	0	Manual LTR mode
X	1	Manual LTD mode

4.3.1.5.2. Resetting the Transceiver in CDR Manual Lock Mode

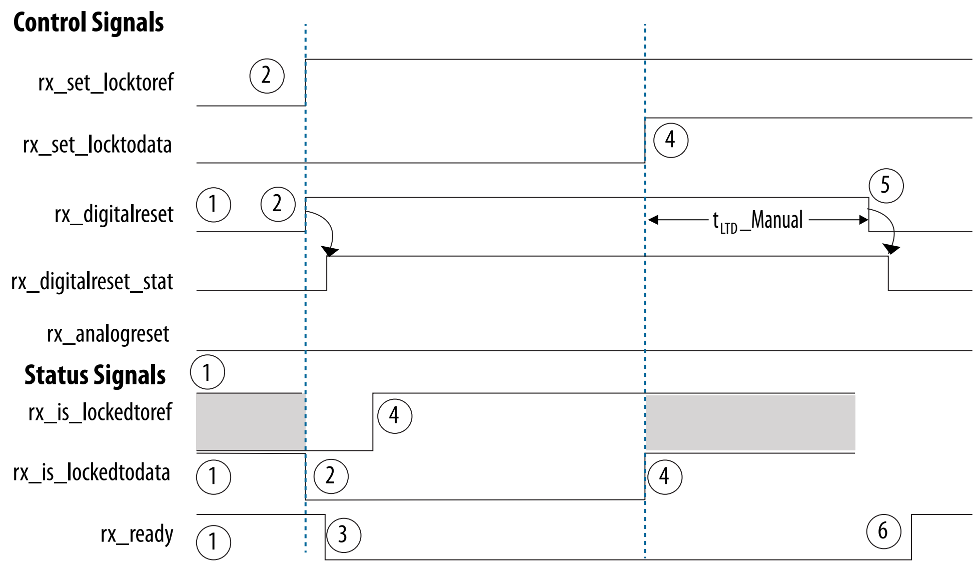
The numbers in this list correspond to the numbers in the following figure, which guides you through the steps to put the CDR in manual lock mode.

1. Make sure that the calibration is complete (`rx_cal_busy` is low) and the transceiver goes through the initial reset sequence. The `rx_digitalreset` and `rx_analogreset` signals should be low. The `rx_is_lockedtoref` is a don't care and can be either high or low. The `rx_is_lockedtodata` and `rx_ready`

signals should be high, indicating that the transceiver is out of reset. Alternatively, you can start directly with the CDR in manual lock mode after the calibration is complete.

2. Assert the `rx_set_locktoref` signal high to switch the CDR to the lock-to-reference mode. The `rx_is_lockedtodata` status signal is deasserted. Assert the `rx_digitalreset` signal high at the same time or after `rx_set_lockedtoref` is asserted if you use the user-coded reset. When the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP is used in auto reset mode, the `rx_digitalreset` is automatically asserted. When the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP is used in manual reset mode, the `rx_digitalreset` must be manually asserted after the assertion of `rx_set_lockedtoref`.
    a. Wait for `rx_digitalreset_stat` to assert, to ensure that `rx_digitalreset` asserts successfully in the PCS.
3. After the `rx_digitalreset_stat` signal gets asserted, the `rx_ready` status signal is deasserted.
4. Assert the `rx_set_locktodata` signal high $t_{LTR\_LTD\_Manual}$ (minimum 15 µs) after the CDR is locked to reference i.e. `rx_is_lockedtoref` should be high and stable for a minimum $t_{LTR\_LTD\_Manual}$ (15 µs), before asserting `rx_set_lockedtodata`. This is required to filter spurious glitches on `rx_is_lockedtoref`. The `rx_is_lockedtodata` status signal gets asserted, which indicates that the CDR is now set to LTD mode. The `rx_is_lockedtoref` status signal can be a high or low and can be ignored after asserting `rx_set_locktodata` high after the CDR is locked to reference.
5. Deassert the `rx_digitalreset` signal after a minimum of t_{LTD_Manual}.
    a. Wait for `rx_digitalreset_stat` to deassert, to ensure that `rx_digitalreset` deasserts successfully in the PCS
6. If you are using the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP, the `rx_ready` status signal gets asserted after the `rx_digitalreset` signal is deasserted. This indicates that the receiver is now ready to receive data with the CDR in manual mode.

Figure 175. Reset Sequence Timing Diagram for Receiver when CDR is in Manual Lock Mode



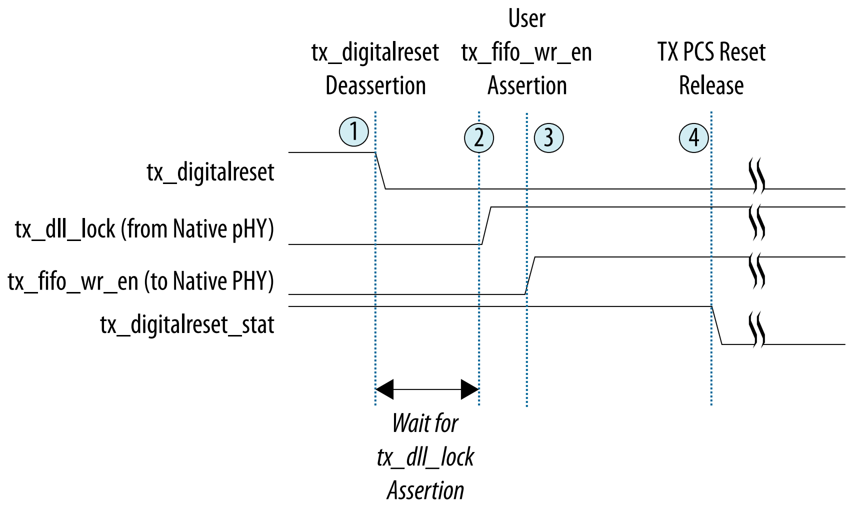
4.3.1.6. Special TX PCS Reset Release Sequence

The release of TX PCS reset is handled differently in special cases. To ensure that transmitter channels are ready to transmit, you must properly release the TX PCS reset for these special cases.

While using Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP or while implementing your own reset controller, you must account for the following special cases

- TX Core FIFO in Interlaken/Basic Mode
- Double rate transfer mode enabled
  - TX Core FIFO in Phase Compensation Mode
  - TX Core FIFO in Basic Mode

4.3.1.6.1. TX Core FIFO in Interlaken/Basic Mode


tx_digitalreset Deassertion
User tx_fifo_wr_en Assertion
TX PCS Reset Release
1
2
3
4
tx_digitalreset
tx_dll_lock (from Native pHY)
tx_fifo_wr_en (to Native PHY)
tx_digitalreset_stat
Wait for tx_dll_lock Assertion

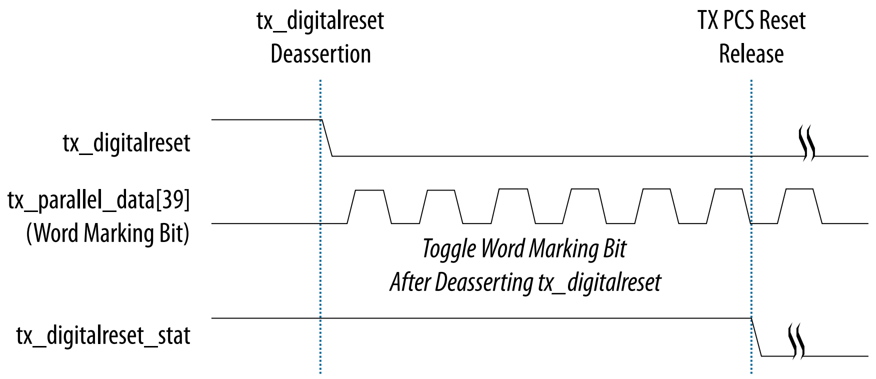
1. Deassert `tx_digitalreset` after PLL has acquired lock.
2. Wait for `tx_dll_lock` (from Transceiver Native PHY), to assert.
3. Assert `tx_fifo_wr_en` after `tx_dll_lock` asserts.
4. Wait for `tx_digitalreset_stat` signal from the PHY to deassert, to ensure that `tx_digitalreset` deasserts successfully.

4.3.1.6.2. Double Rate Transfer Mode enabled

While using Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP, if you enable Double Rate Transfer Mode, you must account for the following two cases:

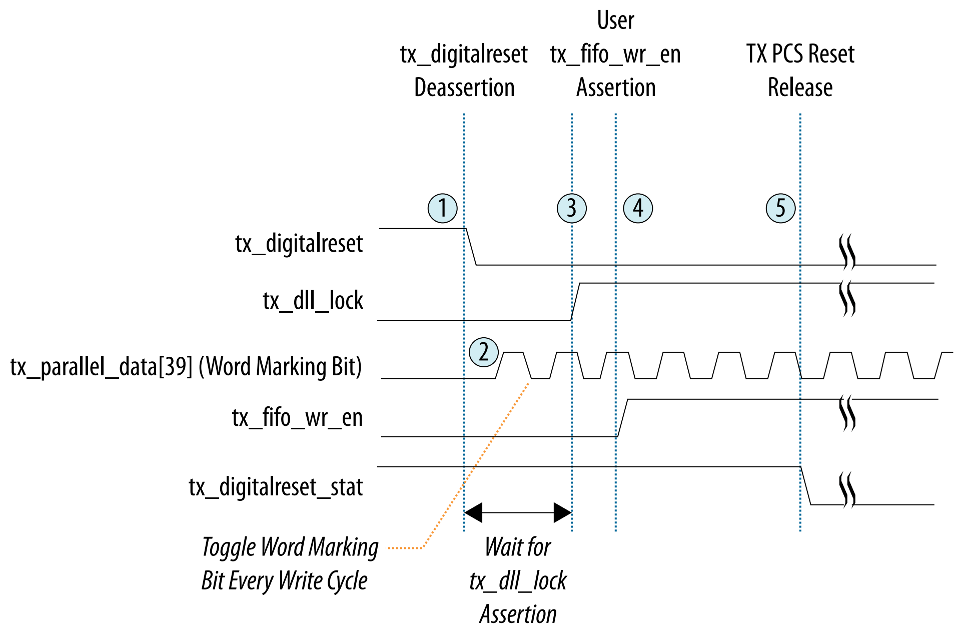
- TX Core FIFO in Phase Compensation Mode
- TX Core FIFO in Basic Mode

TX Core FIFO in Phase Compensation Mode


tx_digitalreset Deassertion
TX PCS Reset Release
tx_digitalreset
tx_parallel_data[39] (Word Marking Bit)
Toggle Word Marking Bit After Deasserting tx_digitalreset
tx_digitalreset_stat

1. Deassert `tx_digitalreset` after PLL has acquired lock.
2. Start to toggle the word marking bit `tx_parallel_data[39]` until `tx_digitalreset_stat` is deasserted.
3. Wait for `tx_digitalreset_stat` signal from the PHY to deassert, to ensure that `tx_digitalreset` deasserts successfully.

TX Core FIFO in Basic Mode


tx_digitalreset Deassertion
User tx_fifo_wr_en Assertion
TX PCS Reset Release
1
2
3
4
5
tx_digitalreset
tx_dll_lock
tx_parallel_data[39] (Word Marking Bit)
tx_fifo_wr_en
tx_digitalreset_stat
Toggle Word Marking Bit Every Write Cycle
Wait for tx_dll_lock Assertion

1. Deassert `tx_digitalreset` after pll has acquired lock.
2. Start to toggle the word marking bit `tx_parallel_data[39]`.
3. Wait for `tx_dll_lock` (from Transceiver Native PHY), to assert.
4. Assert `tx_fifo_wr_en` after `tx_dll_lock` is asserted.
5. Wait for `tx_digitalreset_stat` signal from the PHY to deassert, to ensure that `tx_digitalreset` deasserts successfully.

4.3.1.6.3. TX Gearbox Ratio *:67

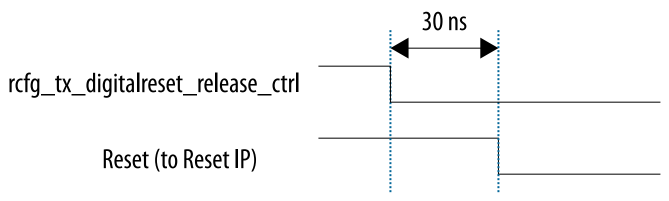
When Enhanced PCS gearbox is enabled with gearbox ratio of *:67 in TX channel, TX PCS reset release is handled differently.

Make sure you have enabled the `rcfg_tx_digitalreset_release_ctrl` port on **Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY**, under **Dynamic Reconfiguration** options, if you intend to dynamically reconfigure to/from gearbox ratio *:67 (i.e 32:67, 40:67 and 64:67). To ensure that transceiver channels are ready to transmit data, you must properly reset the transceiver PHY.

Configuring to *:67 Gearbox ratio

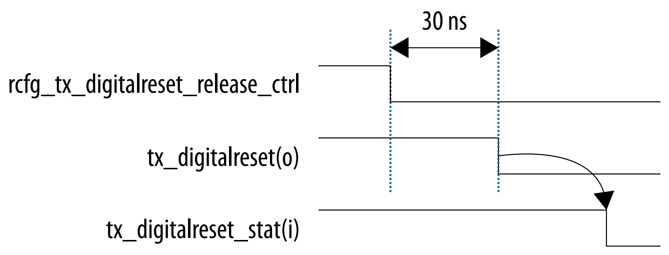
- **With Intel Stratix 10 Reset Controller IP**

  When configuring to gearbox ratio of *:67, deassert the `rcfg_tx_digitalreset_ctrl` port 30ns before you deassert the reset input of reset controller IP.



- **With User Code Reset Controller**

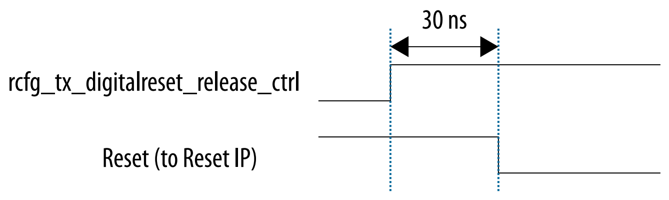
  When configuring to gearbox ratio of *:67, deassert the `rcfg_tx_digitalreset_release_ctrl` port 30ns before you deassert the `tx_digitalreset`.



Configuring from *:67 Gearbox Ratio

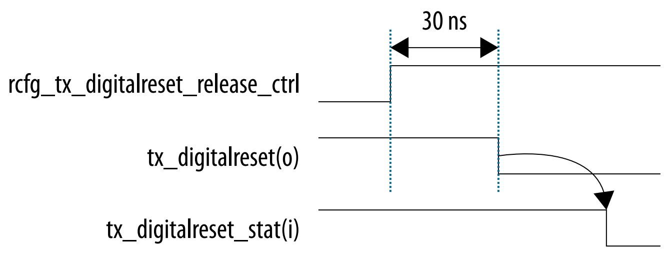
- **With Intel Stratix 10 Reset Controller IP**

  When configuring from gearbox ratio of *:67 to another mode, assert the `rcfg_tx_digitalreset_release_ctrl` port 30ns before you deassert the reset input of reset controller IP.



- **With User Code Reset Controller**

  When configuring from gearbox ratio of *:67 to another mode, assert the `rcfg_tx_digitalreset_release_ctrl` port 30ns before you deassert the `tx_digitalreset`.



4.3.2. Transceiver Blocks Affected by Reset and Power-down Signals

You must reset the PCS block each time you reset the PMA or PLL. However, you can reset the PCS block only without resetting the PMA or PLL.

Table 145. Transceiver Blocks Affected by Specified Reset and Power-down Signals

Transceiver Block	tx_analogreset	tx_digitalreset	rx_analogreset	rx_digitalreset
CDR			Yes	
Receiver Standard PCS				Yes
Receiver Enhanced PCS				Yes
Receiver PMA			Yes	
Receiver PCIe Gen3 PCS				Yes
Transmitter Standard PCS		Yes		
Transmitter Enhanced PCS		Yes		
Transmitter PMA	Yes			
Transmitter PCIe Gen3 PCS		Yes		

4.4. Using PCS Reset Status Port

Table 146. PCS Status Reset Port Values for Different Conditions

PCS reset status port	Correct Reset Applied	No Reset Applied	rx_coreclkin not connected	tx_coreclkin not connected
`tx_transfer_ready`	1	0	1	0
`osc_transfer_en`	1	1	1	1
`tx_digitalreset_timeout`	0	0	0	0
`tx_fifo_ready`	1	1	1	0
`rx_transfer_ready`	1	0	0	1
`rx_digitalreset_timeout`	0	0	0	0
`rx_fifo_ready`	1	0	0	1
`tx_digitalreset_stat`	0	1	0	1
`rx_digitalreset_stat`	0	1	1	0

4.5. Using Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP

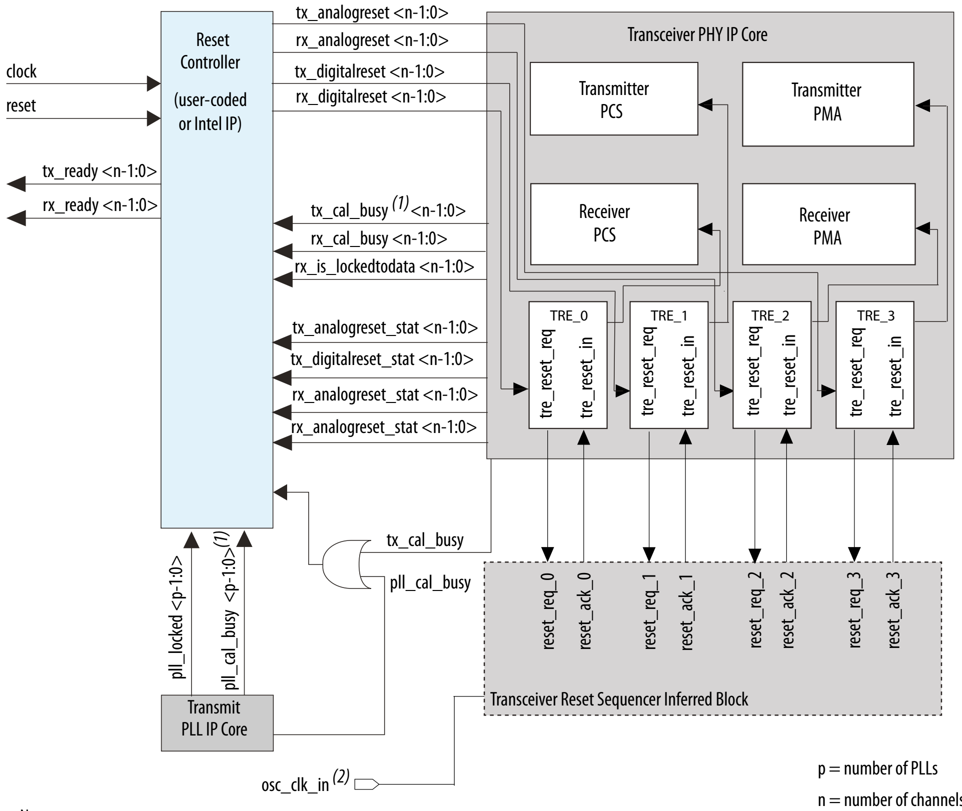
Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP is a configurable IP core that resets transceivers. You can use this IP core rather than creating your own user-coded reset controller. You can define a custom reset sequence for the IP core. You can also modify the IP cores's generated clear text Verilog HDL file to implement custom reset logic.

The Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP handles the reset sequence and supports the following options:

- Separate or shared reset controls per channel
- Separate controls for the TX and RX channels
- Hysteresis for PLL locked status inputs
- Configurable reset timing
- Automatic or manual reset recovery mode in response to loss of PLL lock
- Sequencing TX PCS Reset before RX PCS reset (for PIPE Application)

You should create your own reset controller if the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP does not meet your requirements, especially when you require independent transceiver channel reset. The following figure illustrates the typical use of the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP in a design that includes a transceiver PHY instance and the transmit PLL.

Figure 176. Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP System Diagram



Notes:
(1) You can logical OR the pll_cal_busy and tx_cal_busy signals. tx_cal_busy connects to the reset controller's tx_cal_busy input port.
(2) Internal Oscillator feeds this input clock port. No user clock input required.

The Transceiver PHY Reset Controller IP connects to the Transceiver PHY and the Transmit PLL. The Transceiver PHY Reset Controller IP receives status from the Transceiver PHY and the Transmit PLL. Based on the status signals or the reset input, it generates TX and RX reset signals to the Transceiver PHY.

The `tx_ready` signal indicates whether the TX PMA has exited the reset state, and if the TX PCS is ready to transmit data. The `rx_ready` signal indicates whether the RX PMA has exited the reset state, and if the RX PCS is ready to receive data. You must monitor these signals to determine when the transmitter and receiver are out of the reset sequence.

4.5.1. Parameterizing Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP

This section lists steps to configure the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP in the IP Catalog. You can customize the following Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP parameters for different modes of operation.

To parameterize and instantiate the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP:

1. Make sure the correct **Device Family** is selected under **Assignments ➤ Device**.
2. Click **Tools ➤ IP Catalog ➤ , then Installed IP ➤ Library ➤ Interface Protocols ➤ Transceiver PHY ➤ Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP**.
3. Select the options required for your design. For a description of these options, refer to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP Parameters on page 333.
4. Click **Finish**. The wizard generates files representing your parameterized IP variation for synthesis and simulation.

4.5.2. Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP Parameters

The Intel Quartus Prime Pro Edition software provides a GUI to define and instantiate a Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP to reset transceiver PHY.

Table 147. General Options

Name	Range	Description
Tile Type of Native PHY IP	`L-Tile ES, L-Tile Production / H-Tile`	Specifies the tile type to which the Reset Controller is connected.
Number of transceiver channels	`1-1000`	Specifies the number of channels that connect to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP. The upper limit of the range is determined by your FPGA architecture.
Number of TX PLLs	`1-1000`	Specifies the number of TX PLLs that connect to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP.
Input clock frequency	`1-500 MHz`	Input clock to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP. The frequency of the input clock in MHz. The upper limit on the input clock frequency is the frequency achieved in timing closure.

continued...

Name	Range	Description
Use fast reset for simulation	**On** /**Off**	When **On**, the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP uses reduced reset counters for simulation. Therefore, the reset behavior in simulation and hardware are different when you enable this option.
Sequence RX digital reset after TX digital reset	**On** /**Off**	When **On**, the IP staggers the deassertion of TX digital reset before RX digital reset (i.e TX digital reset deassertion gates RX digital reset deassertion) . Typically this is used for PIPE application where TX PCS must be out of reset before RX PCS.
Separate interface per channel/PLL	**On** /**Off**	When **On**, the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP provides a separate reset interface for each channel and PLL.
	TX Channel	
Enable TX channel reset control	**On** /**Off**	When **On**, the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP enables the control logic and associated status signals for TX reset. When **Off**, disables TX reset control and status signals.
Use separate TX reset per channel	**On** /**Off**	When **On**, each TX channel has a separate reset. When **Off**, the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP uses a shared TX reset controller for all channels.
TX digital reset mode	**Auto**, **Manual**	Specifies the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP behavior when the `pll_locked` signal is deasserted. The following modes are available: • **Auto**—The associated `tx_digitalreset` controller automatically resets whenever the `pll_locked` signal is deasserted. Intel recommends this mode. • **Manual**—The associated `tx_digitalreset` controller is not reset when the `pll_locked` signal is deasserted, allowing you to choose corrective action.
tx_analogreset duration	`1-999999999`	Specifies the time in ns ($t_{tx\_analogreset}$) to continue to assert `tx_analogreset` after the reset input and all other gating conditions are removed. The value is rounded up to the nearest clock cycle. The Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP shows a default value.
tx_digitalreset duration	`1-999999999`	Specifies the time in ns ($t_{tx\_digitalreset}$) to continue to assert the `tx_digitalreset` after the reset input and all other gating conditions are removed. The value is rounded up to the nearest clock cycle. The Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP shows a default value.
pll_locked input hysteresis	`0-999999999`	Specifies the amount of hysteresis in ns to add to the `pll_locked` status input to filter spurious unreliable assertions of the `pll_locked` signal. A value of 0 adds no hysteresis. A higher value filters glitches on the `pll_locked` signal. Intel recommends that the amount of hysteresis be longer than `tpll_lock_max_time`.

continued...

Name	Range	Description
Enable pll_cal_busy input port	**On/ Off**	When **On**, the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP enables/ exposes the **pll_cal_busy** input port. When **Off**, disables **pll_cal_busy** input port.
	RX Channel	
Enable RX channel reset control	**On /Off**	When **On**, each RX channel has a separate reset input. When **Off**, each RX channel uses a shared RX reset input for all channels. This implies that if one of the RX channels is not locked, all other RX channels are held in reset until all RX channels are locked. Digital reset stays asserted until all RX channels have acquired lock.
Use separate RX reset per channel	**On /Off**	When **On**, each RX channel has a separate reset input. When **Off**, uses a shared RX reset controller for all channels.
RX digital reset mode	**Auto**, **Manual**	Specifies the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP behavior when the PLL lock signal is deasserted. The following modes are available: • **Auto**—The associated `rx_digitalreset` controller automatically resets whenever the `rx_is_lockedtodata` signal is deasserted. • **Manual**—The associated `rx_digitalreset` controller is not reset when the `rx_is_lockedtodata` signal is deasserted, allowing you to choose corrective action.
rx_analogreset duration	1-999999999	Specifies the time in ns to continue to assert the `rx_analogreset` after the reset input and all other gating conditions are removed. The value is rounded up to the nearest clock cycle. The default value is 40 ns.
rx_digitalreset duration	1-999999999	Specifies the time in ns to continue to assert the `rx_digitalreset` after the reset input and all other gating conditions are removed. The value is rounded up to the nearest clock cycle. The default value is 5000 ns.

4.5.3. Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP Interfaces

This section describes the top-level signals for the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP.

The following figure illustrates the top-level signals of the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP. Many of the signals in the figure become buses if you choose separate reset controls. The variables in the figure represent the following parameters:

- <*n*>—The number of lanes
- <*p*>—The number of PLLs

Figure 177. Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP Top-Level Signals

Generating the IP core creates signals and ports based on your parameter settings.

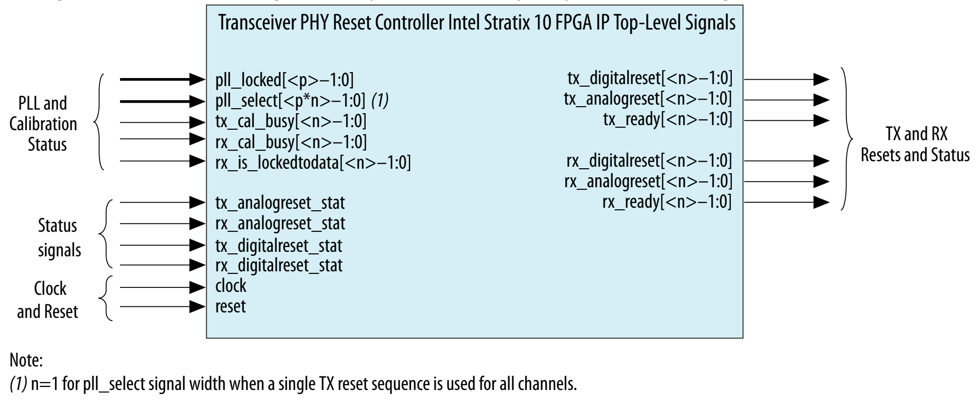


Table 148. Top-Level Signals

This table describes the signals in the above figure in the order that they are shown in the figure.

Signal Name	Direction	Clock Domain	Description
`pll_locked[<p>-1:0]`	Input	Asynchronous	Provides the PLL locked status input from each PLL. When asserted, indicates that the TX PLL is locked. When deasserted, the PLL is not locked. There is one signal per PLL.
`pll_select[<p*n>-1:0]`	Input	Synchronous to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP input clock. Set to zero when not using multiple PLLs.	When you select **Use separate TX reset per channel**, this bus provides enough inputs to specify an index for each `pll_locked` signal to listen to for each channel. When **Use separate TX reset per channel** is disabled, the `pll_select` signal is used for all channels. n=1 when a single TX reset sequence is used for all channels.
`tx_cal_busy[<n> -1:0]`	Input	Asynchronous	This is the calibration status signal that results from the logical OR of `pll_cal_busy` and `tx_cal_busy` signals. The signal goes high when either the TX PLL or Transceiver PHY initial calibration is active. It is not asserted if you manually re-trigger the calibration IP. The signal goes low when calibration is completed. This signal gates the TX reset sequence. The width of this signals depends on the number of TX channels.
`rx_cal_busy[<n> -1:0]`	Input	Asynchronous	This is calibration status signal from the Transceiver PHY IP core. When asserted, the initial calibration is active. When deasserted, calibration has completed. It is not asserted if you manually re-trigger the calibration IP. This signal gates the RX reset sequence. The width of this signals depends on the number of RX channels.
`rx_is_lockedtodata[<n>-1:0]`	Input	Synchronous to CDR	Provides the `rx_is_lockedtodata` status from each RX CDR. When asserted, indicates that a particular RX CDR is ready to receive input data. If you do not choose separate controls for the RX channels, these inputs are ANDed together internally to provide a single status signal.
`tx_analogreset_stat`	Input	Asynchronous	**This is reset status signal from the Transceiver Native PHY IP Core. There is one `tx_analogreset_stat` per channel**. When asserted, reset sequence for TX PMA has begun.

continued...

Signal Name	Direction	Clock Domain	Description
			When deasserted, reset sequence for TX PMA has finished.
`rx_analogreset_stat`	Input	Asynchronous	**This is reset status signal from the Transceiver Native PHY IP Core. There is one `rx_analogreset_stat` per channel**. When asserted, reset sequence for RX PMA has begun. When deasserted, reset sequence for RX PMA has finished.
`tx_digitalreset_stat`	Input	Asynchronous	**This is reset status signal from the Transceiver Native PHY IP Core. There is one `tx_digitalreset_stat` per channel**. When asserted, reset sequence for TX PCS has begun. When deasserted, reset sequence for TX PCS has finished.
`rx_digitalreset_stat`	Input	Asynchronous	**This is reset status signal from the Transceiver Native PHY IP Core. There is one `rx_digitalreset_stat` per channel**. When asserted, reset sequence for RX PCS has begun. When deasserted, reset sequence for RX PCS has finished.
`clock`	Input	N/A	A free running system clock input to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP from which all internal logic is driven. If a free running clock is not available, hold reset until the system clock is stable.
`reset`	Input	Asynchronous	Asynchronous reset input to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP. When asserted, all configured reset outputs are asserted. Holding the reset input signal asserted holds all other reset outputs asserted. An option is available to synchronize with the system clock. In synchronous mode, the reset signal needs to stay asserted for at least two clock cycles by default.
`tx_digitalreset[<n>-1:0]`	Output	Synchronous to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP input clock.	Digital reset for TX channels. The width of this signal depends on the number of TX channels. This signal is asserted when any of the following conditions is true: • `reset` is asserted • `pll_cal_busy` is asserted • `tx_cal_busy` is asserted • PLL has not reached the initial lock (`pll_locked` deasserted) • `pll_locked` is deasserted and `tx_manual` is deasserted • `tx_analogreset_stat` is asserted When all of these conditions are false, the reset counter begins its countdown for deassertion of `tx_digitalreset`.
`tx_analogreset[<n>-1:0]`	Output	Synchronous to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP input clock.	Analog reset for TX channels. The width of this signal depends on the number of TX channels. This signal is asserted when `reset` and `tx_cal_busy` are asserted.

continued...

Signal Name	Direction	Clock Domain	Description
`tx_ready[<n>-1:0]`	Output	Synchronous to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP input clock.	Status signal to indicate when the TX reset sequence is complete. This signal is deasserted while the TX reset is active. It is asserted a few clock cycles after the deassertion of `tx_digitalreset`. Some protocol implementations may require you to monitor this signal prior to sending data. The width of this signal depends on the number of TX channels.
`rx_digitalreset[<n>-1:0]`	Output	Synchronous to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP input clock.	Digital reset for RX. The width of this signal depends on the number of channels. This signal is asserted when any of the following conditions is true: • `reset` is asserted • `rx_analogreset` is asserted • `rx_cal_busy` is asserted • `rx_is_lockedtodata` is deasserted and `rx_manual` is deasserted • `tx_digitalreset_stat` is asserted (if TX reset and sequencing TX and RX digital resets are enabled) • `rx_analogreset_stat` is asserted When all of these conditions are false, the reset counter begins its countdown for deassertion of `rx_digitalreset`.
`rx_analogreset[<n>-1:0]`	Output	Synchronous to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP input clock.	Analog reset for RX. When asserted, resets the RX CDR and the RX PMA blocks of the transceiver PHY. This signal is asserted when any of the following conditions is true: • `reset` is asserted • `rx_cal_busy` is asserted The width of this signal depends on the number of channels.
`rx_ready[<n>-1:0]`	Output	Synchronous to the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP input clock.	Status signal to indicate when the RX reset sequence is complete. This signal is deasserted while the RX reset is active. It is asserted a few clock cycles after the deassertion of `rx_digitalreset`. Some protocol implementations may require you to monitor this signal prior to sending data. The width of this signal depends on the number of RX channels.

Usage Examples for `pll_select`

- If a single channel can switch between three TX PLLs, the `pll_select` signal indicates which one of the selected three TX PLL's `pll_locked` signal is used to communicate the PLL lock status to the TX reset sequence. In this case, to select the 3-bits wide `pll_locked` port, the `pll_select` port is 2-bits wide.
- If three channels are instantiated with three TX PLLs and with a separate TX reset sequence per channel, the `pll_select` field is 6-bits wide (2-bits per channel). In this case, `pll_select [1:0]` represents channel 0, `pll_select[3:2]` represents channel 1, and `pll_select[5:4]` represents channel 2. For each channel, a separate `pll_locked` signal indicates the PLL lock status.
- If three channels are instantiated with three TX PLLs and with a single TX reset sequence for all three channels, then `pll_select` field is 2-bits wide. In this case, the same `pll_locked` signal indicates the PLL lock status for all three channels.
- If one channel is instantiated with one TX PLL, `pll_select` field is 1-bit wide. Connect `pll_select` to logic 0.
- If three channels are instantiated with only one TX PLL and with a separate TX reset sequence per channel, the `pll_select` field is 3-bits wide. In this case, `pll_select` should be set to 0 since there is only one TX PLL available.

4.5.4. Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP Resource Utilization

This section describes the estimated device resource utilization for two configurations of the Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP. The exact resource count varies by the version of the Intel Quartus Prime Pro Edition software, as well as by optimization options.

Table 149. Reset Controller Resource Utilization

Configuration	Combination ALUTs	Logic Registers
Single transceiver channel	approximately 35	approximately 45
Four transceiver channels, shared TX reset, separate RX resets	approximately 100	approximately 150

4.6. Using a User-Coded Reset Controller

You can design your own user-coded reset controller instead of using Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP. Your user-coded reset controller must provide the following functionality for the recommended reset sequence:

- A clock signal input for your reset logic
- Holds the transceiver channels in reset by asserting the appropriate reset control signals
- Checks the PLL status (for example, checks the status of `pll_locked` and `pll_cal_busy`)

4.6.1. User-Coded Reset Controller Signals

Refer to the signals in the following figure and table for implementation of a user-coded reset controller.

Figure 178. User-Coded Reset Controller, Transceiver PHY, and TX PLL Interaction

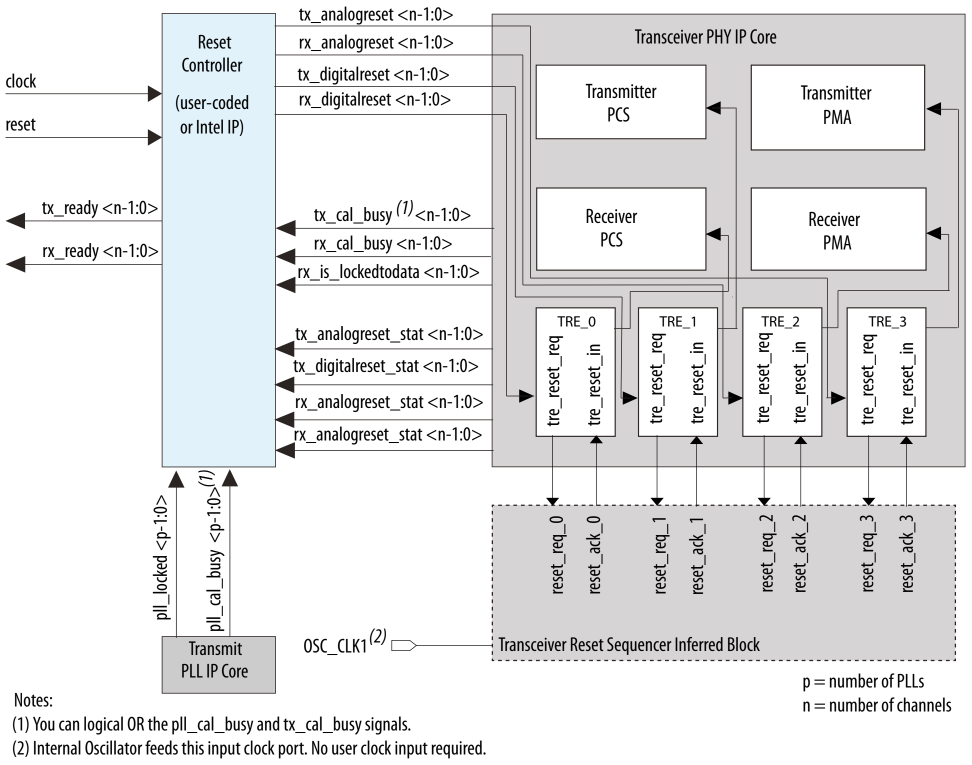


Table 150. User-coded Reset Controller, Transceiver PHY, and TX PLL Signals

Signal Name	Direction	Description
`tx_analogreset`	Output	Resets the TX PMA when asserted high.
`tx_digitalreset`	Output	Resets the TX PCS when asserted high.
`rx_analogreset`	Output	Resets the RX PMA when asserted high.
`rx_digitalreset`	Output	Resets the RX PCS when asserted high.
`clock`	Input	Clock signal for the user-coded reset controller. You can use the system clock without synchronizing it to the PHY parallel clock. The upper limit on the input clock frequency is the frequency achieved in timing closure.
`pll_cal_busy`	Input	A high on this signal indicates the PLL is being calibrated.
`pll_locked`	Input	A high on this signal indicates that the TX PLL is locked to the ref clock.
`tx_cal_busy`	Input	A high on this signal indicates that TX calibration is active. If you have multiple PLLs, you can OR their `pll_cal_busy` signals together.
`rx_is_lockedtodata`	Input	A high on this signal indicates that the RX CDR is in the lock-to-data (LTD) mode.

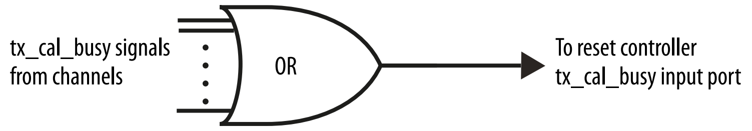
continued...

Signal Name	Direction	Description
`rx_cal_busy`	Input	A high on this signal indicates that RX calibration is active.
`tx_analogreset_stat`	Input	A high on this signal indicates that reset sequence for TX PMA has begun. A low on this signal indicates that reset sequence for TX PMA has finished.
`rx_analogreset_stat`	Input	A high on this signal indicates that reset sequence for RX PMA has begun. A low on this signal indicates that reset sequence for RX PMA has finished.
`tx_digitalreset_stat`	Input	A high on this signal indicates that reset sequence for TX PCS has begun. A low on this signal indicates that reset sequence for TX PCS has finished.
`rx_digitalreset_stat`	Input	A high on this signal indicates that reset sequence for RX PCS has begun. A low on this signal indicates that reset sequence for RX PCS has finished.

4.7. Combining Status or PLL Lock Signals with User Coded Reset Controller

You can combine multiple PHY status signals before feeding into the reset controller as shown below.

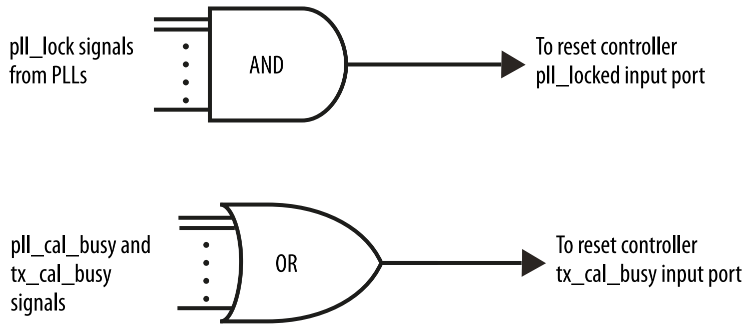
Figure 179. Combining Multiple PHY Status Signals



Note: This configuration also applies to the `rx_cal_busy` signals.

When using multiple PLLs, you can logical AND the `pll_locked` signals feeding the reset controller. Similarly, you can logical OR the `pll_cal_busy` signals to the reset controller `tx_cal_busy` port as shown below.

Figure 180. Multiple PLL Configuration



Resetting different channels separately requires multiple reset controllers. For example, a group of channels configured for Interlaken requires a separate reset controller from another group of channels that are configured for optical communication.

4.8. Resetting Transceiver Channels Revision History

Document Version	Changes
2018.07.06	Made the following changes: • Changed "user reset" to "reset."
2018.05.16	Made the following changes: • For the Reset Controller System Diagrams, moved the tx_ready and rx_ready signals and added "or Intel IP" to the reset controller.
2018.03.16	Made the following changes: • Added Tile Type of Native PHY IP options to "Transceiver PHY Reset Controller Intel Stratix 10 FPGA IP Parameters."
2017.06.06	Made the following change: • Added a new section "Using PCS Reset Status Ports".
2016.12.21	Initial release

5. Intel Stratix 10 L-Tile/H-Tile Transceiver PHY Architecture

5.1. PMA Architecture

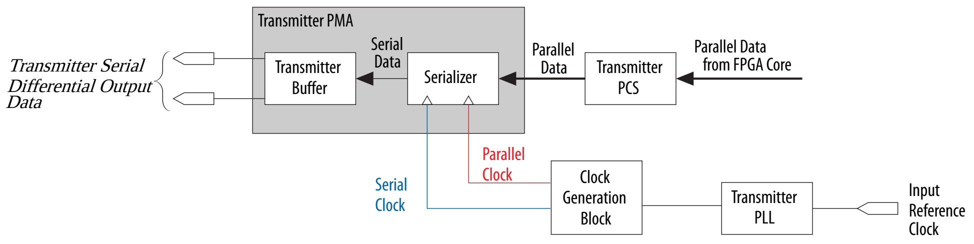
The Physical Medium Attachment (PMA) is the analog front end for the Intel Stratix 10 transceivers.

The PMA receives and transmits high-speed serial data depending on the transceiver channel configuration. All serial data transmitted and received passes through the PMA.

5.1.1. Transmitter PMA

The transmitter serializes the parallel data to create a high-speed serial data stream. Transmitter PMA is composed of the transmitter serializer and the transmitter buffer. The serializer clock is provided from the transmitter PLL.

Figure 181. Transmitter PMA Block Diagram



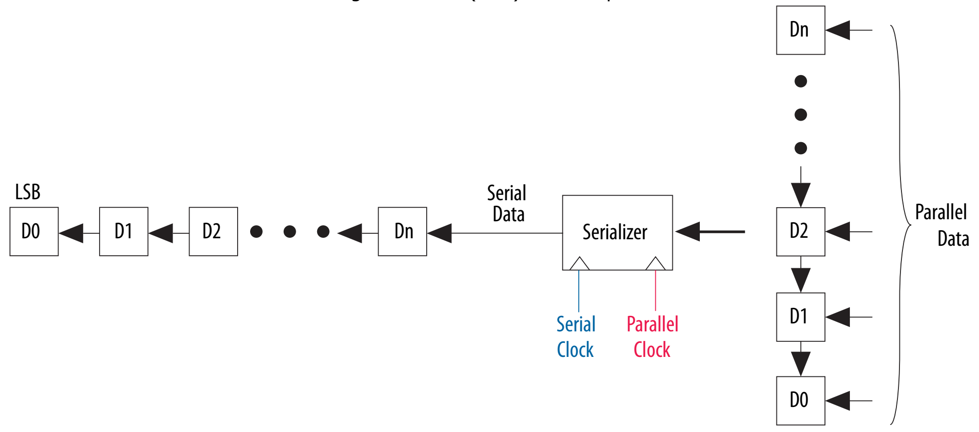
5.1.1.1. Serializer

The serializer converts the incoming low-speed parallel data from the transceiver PCS or FPGA fabric to high-speed serial data and sends the data to the transmitter buffer.

The channel serializer supports the following serialization factors: 8, 10, 16, 20, 32, 40, and 64.

Figure 182. Serializer Block

The serializer block sends out the least significant bit (LSB) of the input data first.

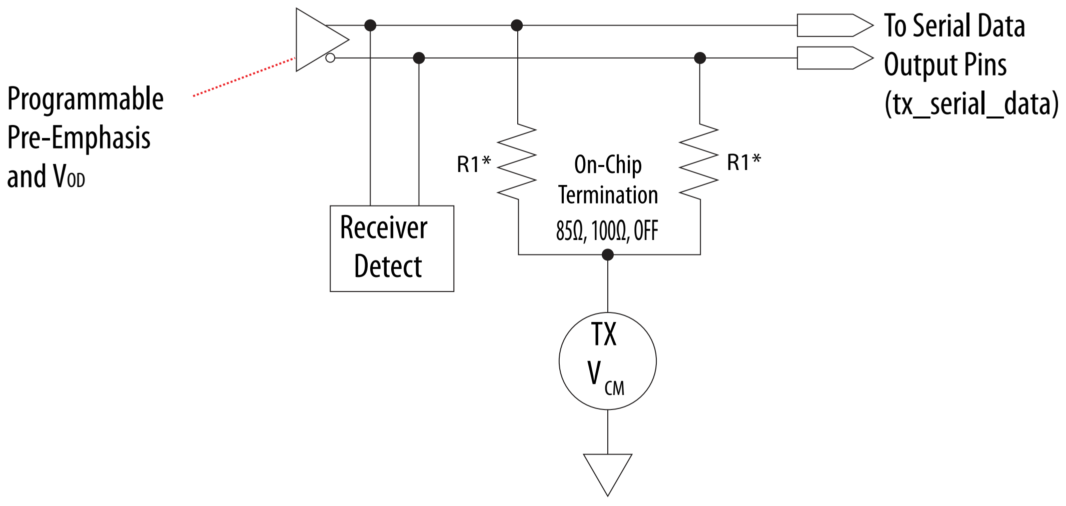


5.1.1.2. Transmitter Buffer

The transmitter buffer includes the following circuitry:

- High Speed Differential I/O
- Programmable differential output voltage (V_{OD})
- Programmable two tap pre-emphasis circuitry with pre-tap and post-tap polarity
  - One pre-cursor tap
  - One post-cursor tap
- Slew rate control
- Internal termination circuitry
- Electrical idle to support PCI Express configuration

Figure 183. Transmitter Buffer


Programmable Pre-Emphasis and V_OD
To Serial Data Output Pins (tx_serial_data)
R1*
On-Chip Termination 85Ω, 100Ω, OFF
R1*
Receiver Detect
TX V_CM

R1* - Half of the actual on-chip termination selected.

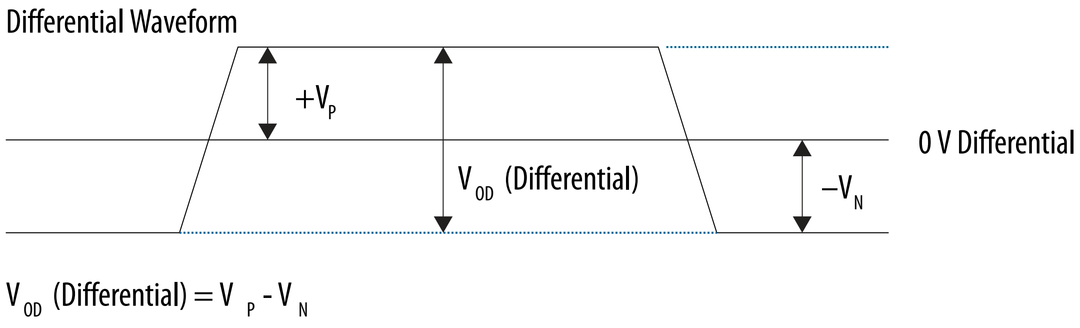
5.1.1.2.1. High-Speed Differential I/O

To improve performance, the Intel Stratix 10 transmitter uses a new architecture in the output buffer—High-Speed Differential I/O.

5.1.1.2.2. Programmable Output Differential Voltage

You can program the differential output voltage (output swing) to handle different channel losses and receiver requirements. There are 32 differential V_{OD} settings up to VCCT power supply level. The step size is 1/31 of the VCCT power supply level.

Figure 184. V_{OD} (Differential) Signal Level



5.1.1.2.3. Programmable Pre-Emphasis

Pre-emphasis can maximize the eye at the far-end receiver. The programmable pre-emphasis module in each transmit buffer amplifies high frequencies in the transmit data signal, to compensate for attenuation in the transmission media.

The pre-tap emphasizes the bit before the transition and de-emphasizes the remaining bits. A different polarity on pre-tap does the opposite.

Related Information

Intel Stratix 10 L-Tile/H-Tile Pre-emphasis and Output Swing Estimator

5.1.2. Receiver PMA

The receiver recovers the clock information from the received data, deserializes the high-speed serial data and creates a parallel data stream for either the receiver PCS or the FPGA fabric.

The receiver portion of the PMA is comprised of the receiver buffer, the clock data recovery (CDR) unit, and the deserializer.

Figure 185. Receiver PMA Block Diagram

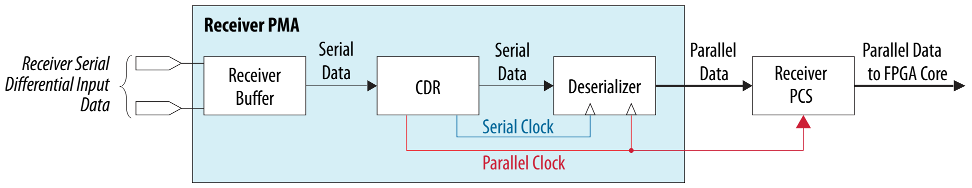
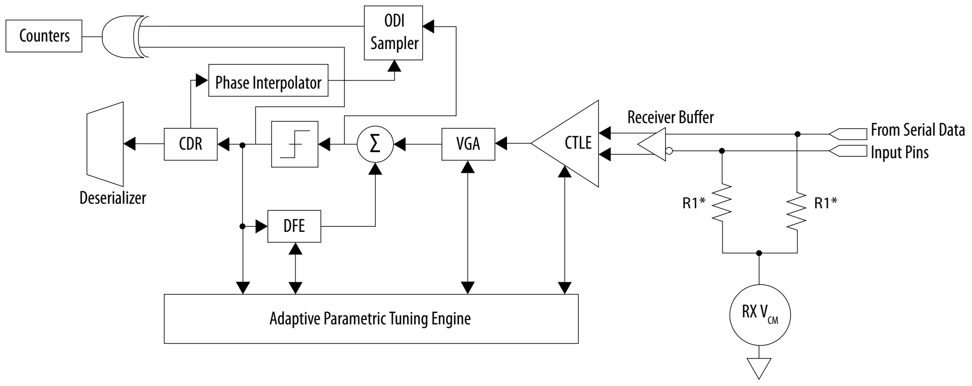


Figure 186. Receiver Buffer



R1* - Half of the actual on-chip termination selected.

5.1.2.1. Receiver Buffer

The receiver input buffer receives serial data from input pin and feeds the serial data to the clock data recovery (CDR) unit and deserializer.

The receiver buffer supports the following features:

- Programmable differential On-Chip Termination (OCT)
- Signal Detector
- Continuous Time Linear Equalization (CTLE)
- Variable Gain Amplifiers (VGA)
- Adaptive Parametric Tuning Engine
- Decision Feedback Equalization (DFE)
- On-Die-Instrumentation (ODI)

Note: ODI is fully supported for H-Tile devices. However, for L-Tile devices, ODI is a functional diagnostic utility for remote system debug and link tuning, in other words, up to 25.8 Gbps. The ODI feature does not support relative channel-to-channel comparisons on L-Tile devices.

5.1.2.1.1. Programmable Differential On-Chip Termination (OCT)

Receiver buffers include programmable on-chip differential termination of 85Ω, 100Ω, or OFF.

You can disable OCT and use external termination. If you select external termination, the receiver common mode is tri-stated. Common mode is based on the external termination connection. You must also implement off-chip biasing circuitry to establish the V_{CM} at the receiver buffer.

Receiver termination is enabled by default even before the device is configured. This is to help mitigate hot swap.

5.1.2.1.2. Signal Detector

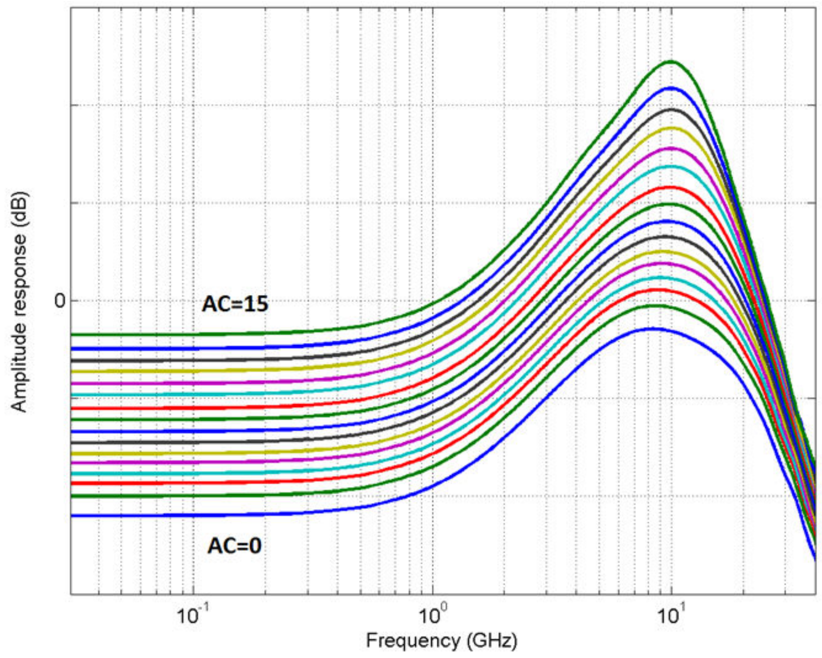
You can enable the optional signal threshold detection circuitry. If enabled, this option senses whether the signal level present at the receiver input buffer is above the signal detect threshold voltage.

5.1.2.1.3. Continuous Time Linear Equalization (CTLE)

The CTLE boosts the signal that is attenuated due to channel characteristics. Each receiver buffer has independently programmable equalization circuits. These equalization circuits amplify the high-frequency component of the incoming signal by compensating for the low-pass characteristics of the physical medium. The CTLE supports AC gain and another setting called, EQ gain, that modifies the DC gain for equalization tuning.

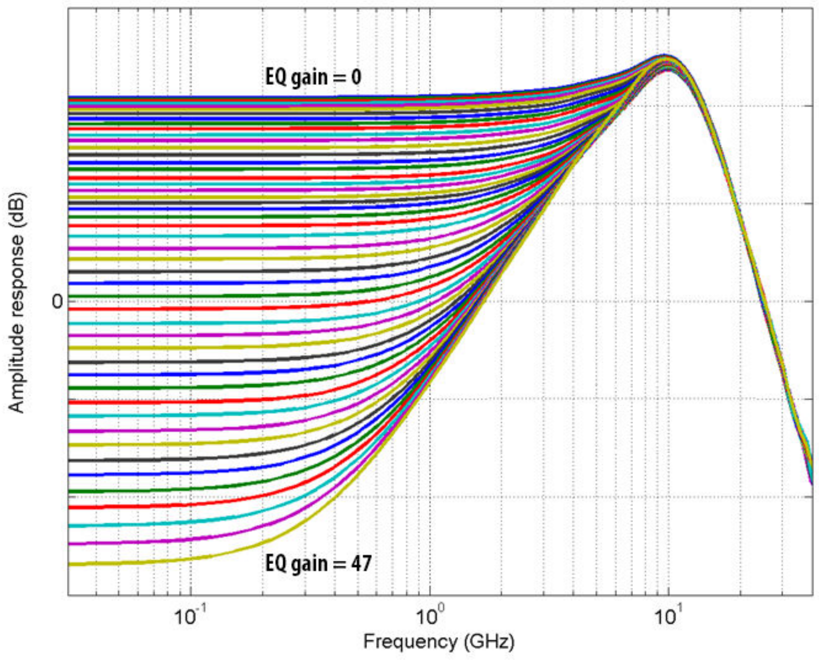
Based on the AC gain selected, the EQ gain applies de-emphasis to the low-frequency spectrum for equalization. See the figures below for the relative effect of these settings on the frequency response of your signal.

Figure 187. Effect of AC Gain on the Frequency Response



Note: This plot was derived with EQ gain = 32. This is only an approximation and is expected to vary across PVT.

Figure 188. Effect of EQ Gain on the Frequency Response



Note: This plot was derived with AC gain = 15. This is only an approximation and is expected to vary across PVT.

5.1.2.1.4. Variable Gain Amplifier (VGA)

Intel Stratix 10 channels have a variable gain amplifier to optimize the signal amplitude prior to the CDR sampling.

5.1.2.1.5. Adaptive Parametric Tuning (ADAPT) Engine

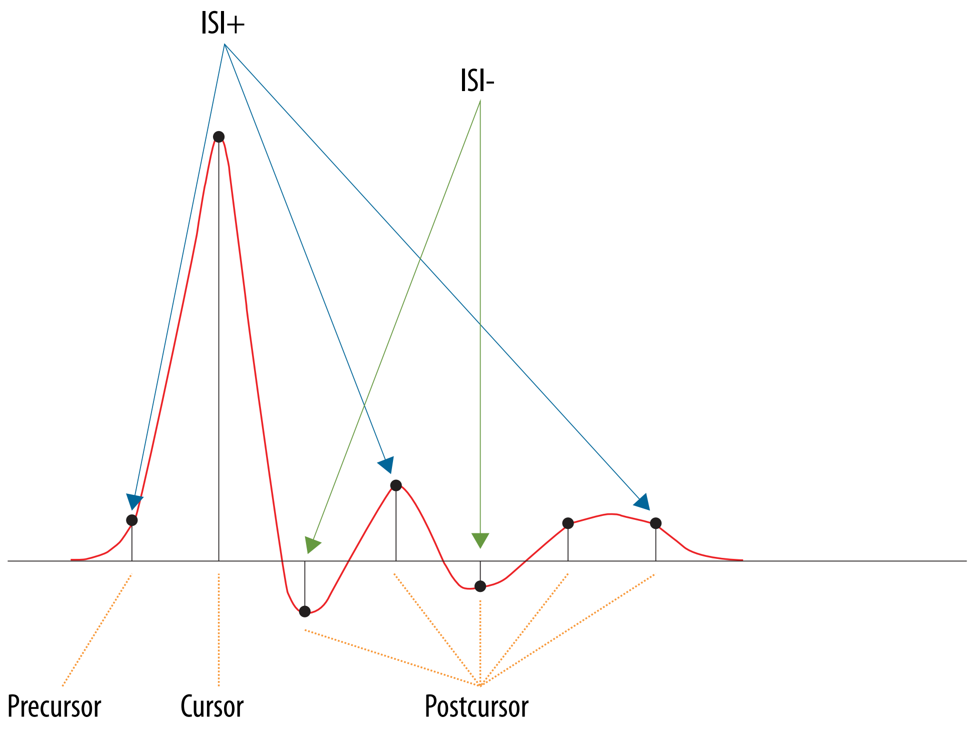
The Adaptive Parametric Tuning Engine includes loops for CTLE, VGA, and DFE adaptation.

5.1.2.1.6. Decision Feedback Equalization (DFE)

DFE amplifies the high frequency components of a signal without amplifying the noise content. It compensates for inter-symbol interference (ISI). DFE minimizes post-cursor ISI by adding or subtracting weighted versions of the previously received bits from the current bit. DFE works in synchronization with the TX pre-emphasis and downstream RX CTLE. This enables the RX CDR to receive the correct data that was transmitted through a lossy and noisy backplane.

The DFE advantage over CTLE is improved Signal to Noise Ratio (SNR).

Figure 189. Signal ISI



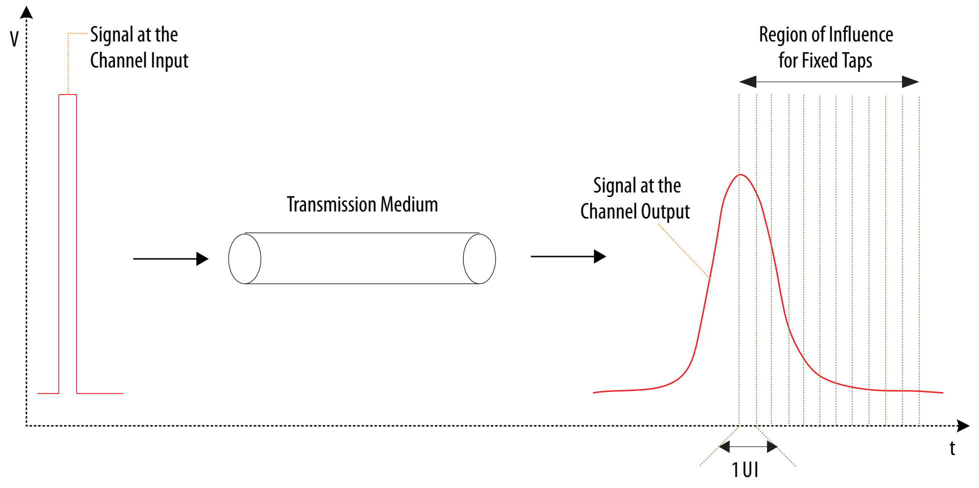
Notes:
- An ideal pulse response is a single data point at the cursor.
- Real world pulse response is non-zero before the cursor (precursor) and after the cursor (postcursor).
- ISI occurs when the data sampled at precursor or postcursor is not zero.

The DFE circuit stores delayed versions of the data. The stored bit is multiplied by a coefficient and then added to the incoming signal. The polarity of each coefficient is programmable.

The DFE architecture supports 15 fixed taps.

The 15 fixed taps translate to the DFE capable of removing the ISI from the previous 15 bits, beginning from the current bit.

Figure 190. Channel Pulse Response


V
Signal at the Channel Input
Transmission Medium
Signal at the Channel Output
Region of Influence for Fixed Taps
1 UI
t

Note: The pulse at the output of the channel shows a long decaying tail. Frequency-dependent losses and quality degradation induces ISI.

5.1.2.1.7. On-Die Instrumentation

The On-Die Instrumentation block allows users to monitor the eye width and height at the summing node of the DFE. This allows you to view the effect of the CTLE, VGA and DFE taps on the received signal.

Note: ODI is fully supported for H-Tile devices. However, for L-Tile devices, ODI is a functional diagnostic utility for remote system debug and link tuning, in other words, up to 25.8 Gbps. The ODI feature does not support relative channel-to-channel comparisons on L-Tile devices.

Refer to *Debug Functions* for details on how to use ODI.

Related Information

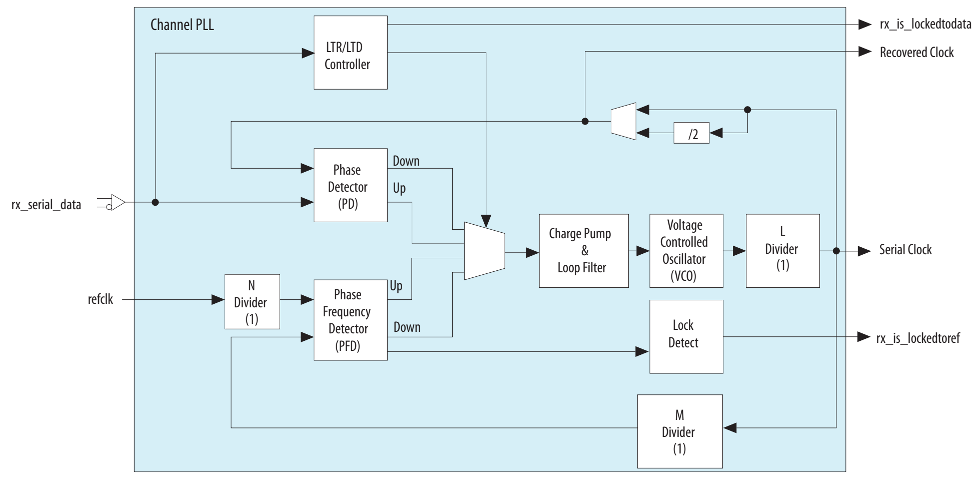
Debug Functions on page 154

5.1.2.2. Clock Data Recovery (CDR) Unit

The PMA of each channel includes a channel PLL that you can configure as a receiver clock data recovery (CDR) for the receiver. You can also configure the channel PLL of channels 1 and 4 as a clock multiplier unit (CMU) PLL for the transmitter in the same bank. The CDR block locks onto the received serial data stream and extracts the embedded clock information in the serial data. There are two operating modes:

- CDR mode—The CDR initially locks onto the reference clock, causing it to operate near the received data rate. After locking to the reference clock, the CDR transitions to lock-to-data mode where it adjusts the clock phase and frequency based on incoming data.
- CMU mode—The CDR locks onto the reference clock and acts as a TX PLL, generating the clock source for the TX. The CDR cannot capture any recovered data in this mode and can only drive x1 clock lines.

Figure 191. Channel PLL Configured as CDR



Note:
1. The Quartus® Prime Pro Edition software automatically chooses the optimal values.

5.1.2.2.1. Lock-to-Reference Mode

In LTR mode, the phase frequency detector (PFD) in the CDR tracks the receiver input reference clock. The PFD controls the charge pump that tunes the VCO in the CDR. The `rx_is_lockedtoref` status signal is asserted active high to indicate that the CDR has locked to the phase and frequency of the receiver input reference clock.

Note: The phase detector (PD) is inactive in LTR mode.

5.1.2.2.2. Lock-to-Data Mode

During normal operation, the CDR must be in LTD mode to recover the clock from the incoming serial data. In LTD mode, the PD in the CDR tracks the incoming serial data at the receiver input. Depending on the phase difference between the incoming data and the CDR output clock, the PD controls the CDR charge pump that tunes the VCO.

Note: The PFD is inactive in LTD mode. The `rx_is_lockedtoref` status signal goes high and low randomly, and is not significant in LTD mode.

5.1.2.2.3. CDR Lock Modes

You can configure the CDR in either automatic lock mode or manual lock mode. By default, the software configures the CDR in automatic lock mode.

Automatic Lock Mode

In automatic lock mode, the CDR initially locks to the input reference clock (LTR mode). After the CDR locks to the input reference clock, the CDR locks to the incoming serial data (LTD mode) when the following conditions are met:

- The signal threshold detection circuitry indicates the presence of valid signal levels at the receiver input buffer when `rx_std_signaldetect` is enabled.
- The CDR output clock is within the configured ppm frequency threshold setting with respect to the input reference clock (frequency locked).
- The CDR output clock and the input reference clock are phase matched within approximately 0.08 unit interval (UI) (phase locked).

If the CDR does not stay locked to data because of frequency drift or severe amplitude attenuation, the CDR switches back to LTR mode.

Manual Lock Mode

The ppm detector and phase relationship detector reaction times can be too long for some applications that require faster CDR lock time. You can manually control the CDR to reduce its lock time using two optional input ports (`rx_set_locktoref` and `rx_set_locktodata`).

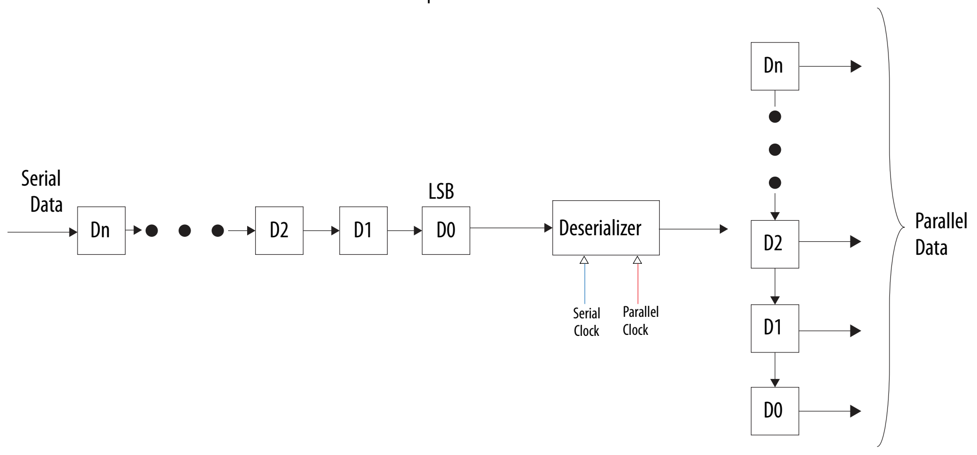
5.1.2.3. Deserializer

The deserializer block clocks in serial input data from the receiver buffer using the high-speed serial recovered clock and deserializes the data using the low-speed parallel recovered clock. The deserializer forwards the deserialized data to the receiver PCS or FPGA fabric.

The deserializer supports the following deserialization factors: 8, 10, 16, 20, 32, 40, and 64. The deserializer can be powered down when the CDR acts as a CMU.

Figure 192. Deserializer Block Diagram

The deserializer block sends out the LSB of the input data first.



5.2. Enhanced PCS Architecture

Intel Stratix 10 transceivers support the following PCS types:

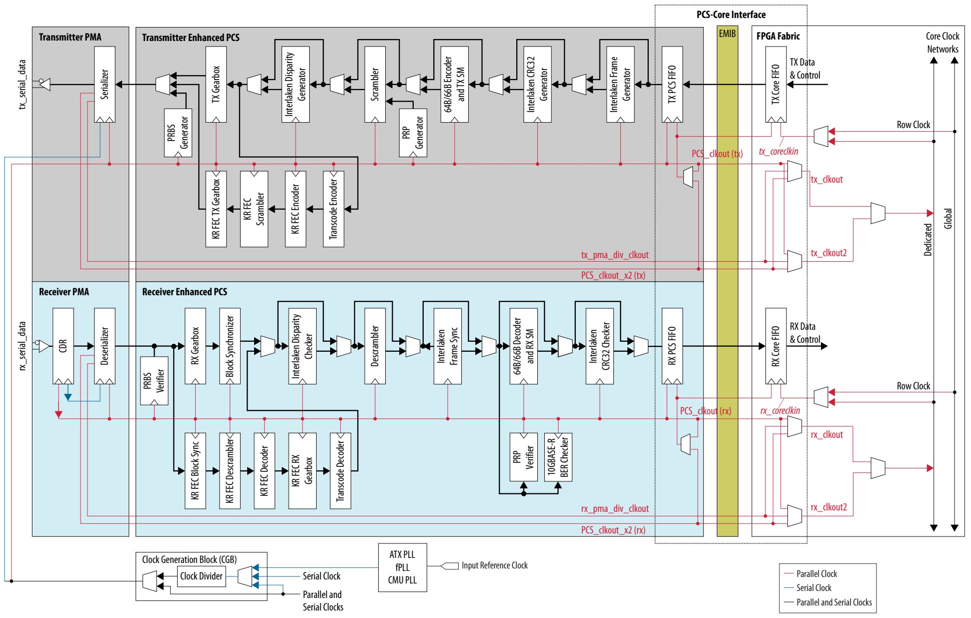
- Enhanced PCS
- Standard PCS
- PCI Express Gen3 PCS

You can use the Enhanced PCS to implement multiple protocols that operate up to 17.4 Gbps line rates on GX channels.

The Enhanced PCS provides the following functions:

- Performs functions common to most serial data industry standards, such as word alignment, block synchronization, encoding/decoding, and framing, before data is sent or received off-chip through the PMA
- Handles data transfer to and from the FPGA fabric
- Internally handles data transfer to and from the PMA
- Provides frequency compensation
- Performs channel bonding for multi-channel low skew applications

Figure 193. Enhanced PCS Datapath Diagram



5.2.1. Transmitter Datapath

5.2.1.1. TX Core FIFO

The TX Core FIFO provides an interface between the FPGA Fabric and across the EMIB to the TX PCS FIFO. It ensures reliable transfer of the data and status signals.

The TX Core FIFO operates in the following modes:

1. Phase Compensation Mode
2. Register Mode
3. Interlaken Mode
4. Basic Mode

Phase Compensation Mode

In Phase Compensation mode, the TX Core FIFO decouples phase variations between `tx_coreclkin` and `PCS_clkout_x2(tx)`. In this mode, the read and write controls of the TX Core FIFO, can be driven by clocks from asynchronous clock sources but must be the same frequency with 0 ppm difference. You can use the FPGA fabric clock or `tx_clkout` (TX parallel clock) to clock the write side of the TX Core FIFO.

Note: In Phase Compensation mode, TX parallel data is valid for every low-speed clock cycle, and `tx_enh_data_valid` signal must be tied with logic level 1.

Register Mode

The Register Mode bypasses the FIFO functionality to eliminate the FIFO latency uncertainty for applications with stringent latency requirements. This is accomplished by tying the read clock of the FIFO with its write clock. In Register Mode, `tx_parallel_data` (data), tx_control (indicates whether `tx_parallel_data` is a data or control word), and `tx_enh_data_valid` (data valid) are registered at the FIFO output. The FIFO in Register Mode has one register stage or one parallel clock latency.

Interlaken Mode

In Interlaken mode, the TX Core FIFO operates as an elastic buffer. In this mode, you have additional signals to control the data flow into the FIFO. Therefore, the FIFO write clock frequency does not have to be the same as the read clock frequency. You can control the writing to the TX Core FIFO with `tx_fifo_wr_en` by monitoring the FIFO flags. The goal is to prevent the FIFO from becoming full or empty. On the read side, read enable is controlled by the Interlaken frame generator.

Basic Mode

In Basic mode, the TX FIFO operates as an elastic buffer, where buffer depths can vary. This mode allows driving write and read side of TX Core FIFO with different clock frequencies. Monitor the FIFO flag to control write and read operations. For TX Core FIFO, assert `tx_fifo_wr_en` signal with `tx_fifo_pempty` signal going low.

5.2.1.2. TX PCS FIFO

The TX PCS FIFO operates in Phase Compensation Mode.

The TX PCS FIFO interfaces between the transmitter PCS and across EMIB to the TX Core FIFO, and ensures reliable transfer of data and status signals. It compensates for the phase difference between the `PCS_clkout_2x`(tx) and `PCS_clkout`(tx).

Related Information

Reconfiguration Interface and Dynamic Reconfiguration on page 394

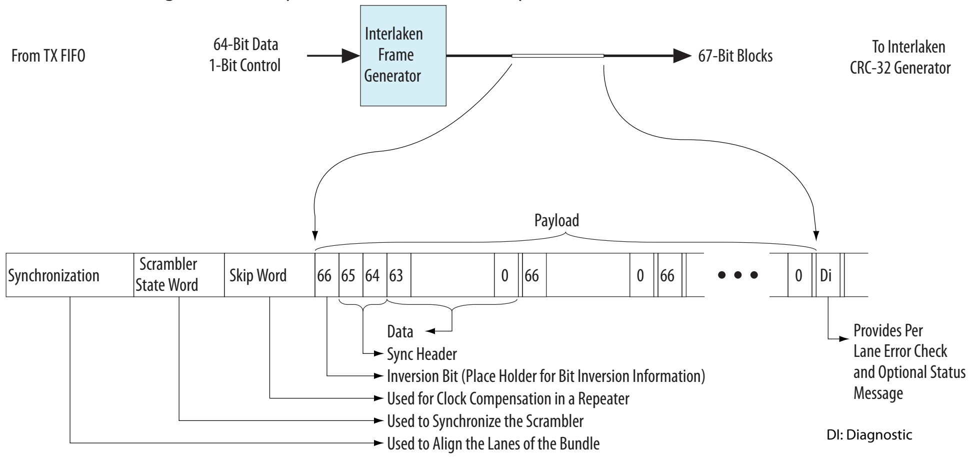
5.2.1.3. Interlaken Frame Generator

The Interlaken frame generator block takes the data from the TX PCS FIFO and encapsulates the payload and burst/idle control words from the FPGA fabric with the framing layer's control words (synchronization word, scrambler state word, skip word, and diagnostic word) to form a metaframe. The Native PHY IP core allows you to set the metaframe length from five 8-byte words to a maximum value of 8192 (64Kbyte words).

Use the same value on frame generator metaframe length for the transmitter and receiver.

Figure 194. Interlaken Frame Generator

The Interlaken frame generator implements the Interlaken protocol.


From TX FIFO
64-Bit Data
1-Bit Control
Interlaken Frame Generator
67-Bit Blocks
To Interlaken CRC-32 Generator
Payload
Synchronization
Scrambler State Word
Skip Word
66
65
64
63
0
66
0
66
0
Di
Data
Sync Header
Inversion Bit (Place Holder for Bit Inversion Information)
Used for Clock Compensation in a Repeater
Used to Synchronize the Scrambler
Used to Align the Lanes of the Bundle
Provides Per Lane Error Check and Optional Status Message
DI: Diagnostic

5.2.1.4. Interlaken CRC-32 Generator

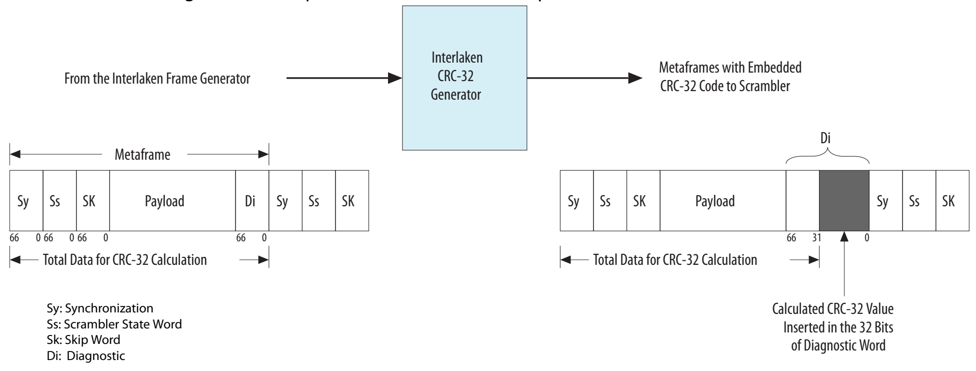
The Interlaken CRC-32 generator block receives data from the Interlaken frame generator and calculates the cyclic redundancy check (CRC) code for each block of data. This CRC code value is stored in the CRC32 field of the diagnostic word. CRC-32 provides a diagnostic tool for each lane. This helps to trace the errors on the interface back to an individual lane.

The CRC-32 calculation covers most of the metaframe, including the diagnostic word, except the following:

- Bits [66:64] of each word
- 58-bit scrambler state within the scrambler state word
- 32-bit CRC-32 field within the diagnostic word

Figure 195. Interlaken CRC-32 Generator

The Interlaken CRC-32 generator implements the Interlaken protocol.


From the Interlaken Frame Generator
Interlaken CRC-32 Generator
Metaframes with Embedded CRC-32 Code to Scrambler
Metaframe
Sy
Ss
SK
Payload
Di
Total Data for CRC-32 Calculation
Calculated CRC-32 Value Inserted in the 32 Bits of Diagnostic Word
Sy: Synchronization
Ss: Scrambler State Word
Sk: Skip Word
Di: Diagnostic

5.2.1.5. 64B/66B Encoder and Transmitter State Machine (TX SM)

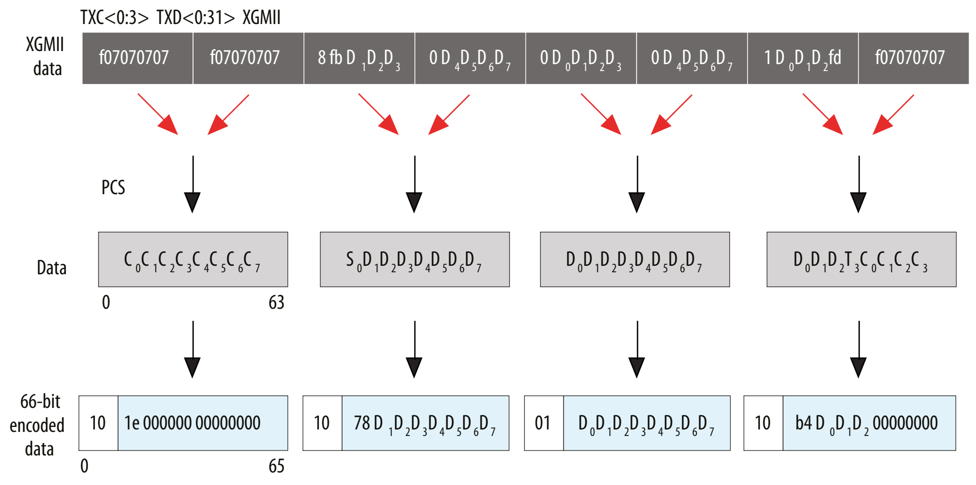
The 64B/66B encoder is used to achieve DC-balance and sufficient data transitions for clock recovery. It encodes 64-bit XGMII data and 8-bit XGMII control into 10GBASE-R 66-bit control or data blocks in accordance with Clause 49 of the IEEE802.3-2008 specification.

The 66-bit encoded data contains two overhead sync header bits that the receiver PCS uses for block synchronization and bit-error rate (BER) monitoring. The sync header is 01 for data blocks and 10 for control blocks. Sync headers are not scrambled and are used for block synchronization. (The sync headers 00 and 11 are not used, and generate an error if seen.) The remainder of the block contains the payload. The payload is scrambled and the sync header bypasses the scrambler.

The encoder block also has a state machine (TX SM) designed in accordance with the IEEE802.3-2008 specification. The TX SM ensures valid packet construction on data sent from the MAC layer. It also performs functions such as transmitting local faults under reset, as well as transmitting error codes when the 10GBASE-R PCS rules are violated.

Note: The 64B/66B encoder is available to implement the 10GBASE-R protocol.

Figure 196. Example Data Pattern for 64B/66B Encoding



64B/66B Encoder Reset Condition

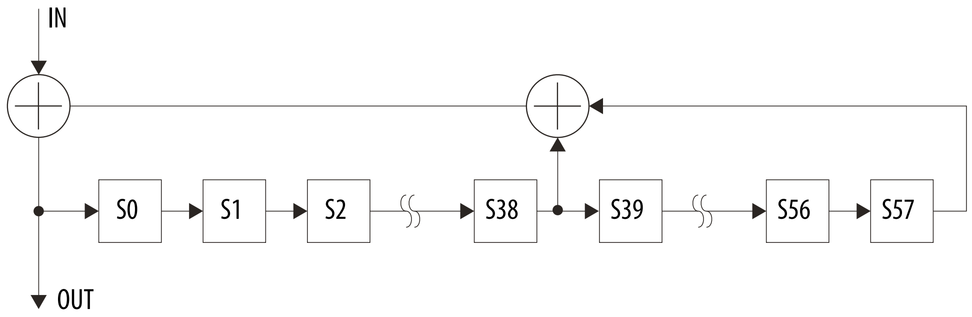
The `tx_digitalreset` signal resets the 64B/66B encoder. During the reset condition, the 64B/66B encoder does not output any signal in contrast with the 8B/10B encoder.

5.2.1.6. Scrambler

The scrambler randomizes data to create transitions to DC-balance the signal and help CDR circuits. The scrambler uses a $x^{58} + x^{39} + 1$ polynomial and supports both synchronous scrambling used for Interlaken and asynchronous (also called self-synchronized) scrambling used for the 10GBASE-R protocol.

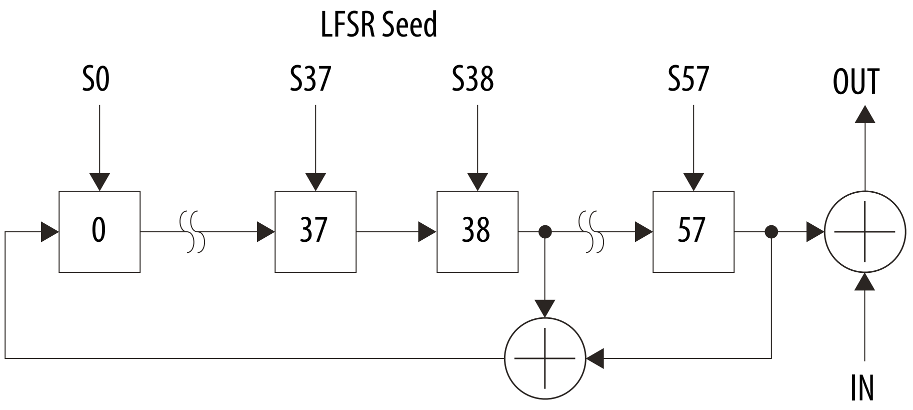
The asynchronous (self-synchronizing) mode does not require an initialization seed. Except for the two sync header bits in each 66-bit data block, the entire 64-bit payload is scrambled by feeding it into a linear feedback shift register (LFSR) continuously to generate scrambled data while the sync-header bits bypass the scrambler. The initial seed is set to all 1s. You can change the seed for the 10GBASE-R protocol using the Native PHY IP core.

Figure 197. Asynchronous Scrambler in Serial Implementation



In synchronous mode, the scrambler is initially reset to different programmable seeds on each lane. The scrambler then runs by itself. Its current state is XOR'd with the data to generate scrambled data. A data checker in the scrambler monitors the data to determine if it should be scrambled or not. If a synchronization word is found, it is transmitted without scrambling. If a scrambler state word is detected, the current scramble state is written into the 58-bit scramble state field in the scrambler state word and sent over the link. The receiver uses this scramble state to synchronize the descrambler. The seed is automatically set for Interlaken protocol.

Figure 198. Synchronous Scrambler Showing Different Programmable Seeds



5.2.1.7. Interlaken Disparity Generator

The Interlaken disparity generator block is in accordance with the Interlaken protocol specification and provides a DC-balanced data output.

The Interlaken protocol solves the unbounded baseline wander, or DC imbalance, of the 64B/66B coding scheme used in 10Gb Ethernet by inverting the transmitted data. The disparity generator monitors the transmitted data and makes sure that the running disparity always stays within a ±96-bit bound. It adds the 67th bit (bit 66) to signal the receiver whether the data is inverted or not.

Table 151. Inversion Bit Definition

Bit 66	Interpretation
0	Bits [63:0] are not inverted; the receiver processes this word without modification
1	Bits [63:0] are inverted; the receiver inverts the bits before processing this word

Note: The Interlaken disparity generator is available to implement the Interlaken protocol.

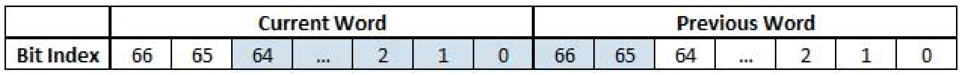
5.2.1.8. TX Gearbox, TX Bitslip and Polarity Inversion

The TX gearbox adapts the PCS data width to the smaller bus width of the PCS-PMA interface (Gearbox Reduction). It supports different ratios (FPGA fabric-PCS Interface Width: PCS-PMA Interface Width) such as 66:32, 66:40, 64:32, 40:40, 32:32, 64:64, 67:64, and 66:64. The gearbox mux selects a group of consecutive bits from the input data bus depending on the gearbox ratio and the data valid control signals.

The TX gearbox also has a bit slipping feature to adjust the data skew between channels. The TX parallel data is slipped on the rising edge of `tx_enh_bitslip` before it is passed to the PMA. The maximum number of the supported bitslips is PCS data width-1 and the slip direction is from MSB to LSB and from current to previous word.

Figure 199. TX Bitslip

`tx_enh_bitslip` = 2 and PCS width of gearbox is 67



	Current Word							Previous Word						
Bit Index	66	65	64	...	2	1	0	66	65	64	...	2	1	0

You can use transmitter data polarity inversion to invert the polarity of every bit of the input data word to the serializer in the transmitter path. The inversion has the same effect as swapping the positive and negative signals of the differential TX buffer. This is useful if these signals are reversed on the board or backplane layout. Enable polarity inversion through the Native PHY IP core.

5.2.1.9. KR FEC Blocks

The KR FEC blocks in the Enhanced PCS are designed in accordance with the 10G-KRFEC and 40G-KRFEC of the IEEE 802.3 specification. The KR FEC implements the Forward Error Correction (FEC) sublayer, a sublayer between the PCS and PMA sublayers.

Most data transmission systems, such as Ethernet, have minimum requirements for the bit error rate (BER). However, due to channel distortion or noise in the channel, the required BER may not be achievable. In these cases, adding a forward error control correction can improve the BER performance of the system.

The FEC sublayer is optional and can be bypassed. When used, it can provide additional margin to allow for variations in manufacturing and environmental conditions. FEC can achieve the following objectives:

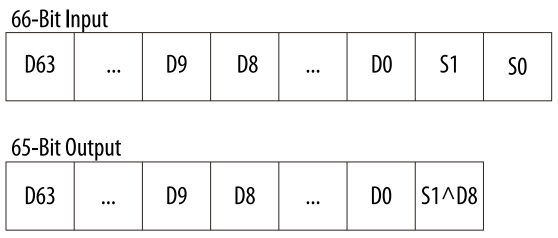
- Support a forward error correction mechanism for the 10GBASE-R/KR and 40GBASE-R/KR protocols
- Support the full duplex mode of operation of the Ethernet MAC
- Support the PCS, PMA, and Physical Medium Dependent (PMD) sublayers defined for the 10GBASE-R/KR and 40GBASE-R/KR protocols

With KR FEC, the BER performance of the system can be improved.

Transcode Encoder

The KR forward error correction (KR FEC) transcode encoder block performs the 64B/66B to 65-bit transcoder function by generating the transcode bit. The transcode bit is generated from a combination of 66 bits after the 64B/66B encoder which consists of a 2-bit synchronization header (S0 and S1) and a 64-bit payload (D0, D1, …, D63). To ensure a DC-balanced pattern, the transcode word is generated by performing an XOR function on the second synchronization bit S1 and payload bit D8. The transcode bit becomes the LSB of the 65-bit pattern output of the transcode encoder.

Figure 200. Transcode Encoder



KR FEC Encoder

FEC (2112,2080) is an FEC code specified in Clause 74 of the IEEE 802.3 specification. The code is a shortened cyclic code (2112, 2080). For each block of 2080 message bits, another 32 parity checks are generated by the encoder to form a total of 2112 bits. The generator polynomial is:

$$g(x) = x^{32} + x^{23} + x^{21} + x^{11} + x^{2} + 1$$

KR FEC Scrambler

The KR FEC scrambler block performs scrambling based on the generation polynomial $x^{58} + x^{39} + 1$, which is necessary for establishing FEC block synchronization in the receiver and to ensure DC balance.

KR FEC TX Gearbox

The KR FEC TX gearbox converts 65-bit input words to 64-bit output words to interface the KR FEC encoder with the PMA. This gearbox is different from the TX gearbox used in the Enhanced PCS. The KR FEC TX gearbox aligns with the FEC block. Because the encoder output (also the scrambler output) has its unique word size pattern, the gearbox is specially designed to handle that pattern.

5.2.2. Receiver Datapath

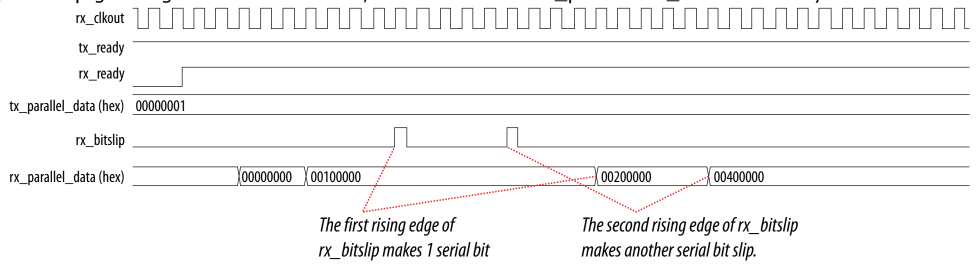
5.2.2.1. RX Gearbox, RX Bitslip, and Polarity Inversion

The RX gearbox adapts the PMA data width to the larger bus width of the PCS channel (Gearbox Expansion). It supports different ratios (PCS-PMA interface width : FPGA fabric to PCS-Core width) such as 32:66, 40:66, 32:67, 32:64, 40:40, 32:32, 64:64, 67:64, and 66:64 and a bit slipping feature.

RX bitslip is engaged when the RX block synchronizer or `rx_bitslip` is enabled to shift the word boundary. On the rising edge of the bitslip signal of the RX block synchronizer or `rx_bitslip` from the FPGA fabric, the word boundary is shifted by one serial bit or 1UI. Each bit slip removes the earliest received bit from the received data.

Figure 201. RX Bitslip

`rx_bitslip` goes high and low two times, which shifts the `rx_parallel_data` boundary two bits.


rx_clkout
tx_ready
rx_ready
tx_parallel_data (hex)
00000001
rx_bitslip
rx_parallel_data (hex)
00000000
00100000
00200000
00400000
The first rising edge of rx_bitslip makes 1 serial bit
The second rising edge of rx_bitslip makes another serial bit slip.

The receiver gearbox can invert the polarity of the incoming data. This is useful if the receiver signals are reversed on the board or backplane layout. Enable polarity inversion through the Native PHY IP core.

Data valid generation logic is essential for gearbox operation. Each block of data is accompanied by `rx_enh_data_valid` data valid signal which "qualifies" the block as valid or not. The data valid toggling pattern is dependent on the data width conversion ratio. For example, if the ratio is 66:40, the data valid signal is high in 20 out of 33 cycles or approximately 2 out of 3 cycles and the pattern repeats every 33 `rx_clkout` RX low-speed parallel clock cycles.

5.2.2.2. Block Synchronizer

The block synchronizer determines the block boundary of a 66-bit word in the case of the 10GBASE-R protocol or a 67-bit word in the case of the Interlaken protocol. The incoming data stream is slipped one bit at a time until a valid synchronization header (bits 65 and 66) is detected in the received data stream. After the predefined number of synchronization headers (as required by the protocol specification) is detected, the block synchronizer asserts `rx_enh_blk_lock` (block lock status signal) to other receiver PCS blocks down the receiver datapath and to the FPGA fabric.

5.2.2.3. Interlaken Disparity Checker

The Interlaken disparity checker examines the received inversion bit inserted by the far end disparity generator, to determine whether to reverse the inversion process of the Interlaken disparity generation.

Note: The Interlaken disparity checker is available to implement the Interlaken protocol.

5.2.2.4. Descrambler

The descrambler block descrambles received data to regenerate unscrambled data using the $x^{58} + x^{39} +1$ polynomial. Like the scrambler, it operates in asynchronous mode or synchronous mode.

Related Information

Scrambler on page 357

5.2.2.5. Interlaken Frame Synchronizer

The Interlaken frame synchronizer delineates the metaframe boundaries and searches for each of the framing layer control words: Synchronization, Scrambler State, Skip, and Diagnostic. When four consecutive synchronization words have been identified, the frame synchronizer achieves the frame locked state. Subsequent metaframes are then checked for valid synchronization and scrambler state words. If four consecutive invalid synchronization words or three consecutive mismatched scrambler state words are received, the frame synchronizer loses frame lock. In addition, the frame synchronizer provides `rx_enh_frame_lock` (receiver metaframe lock status) to the FPGA fabric.

Note: The Interlaken frame synchronizer is available to implement the Interlaken protocol.

5.2.2.6. 64B/66B Decoder and Receiver State Machine (RX SM)

The 64B/66B decoder reverses the 64B/66B encoding process. The decoder block also contains a state machine (RX SM) designed in accordance with the IEEE802.3-2008 specification. The RX SM checks for a valid packet structure in the data sent from the remote side. It also performs functions such as sending local faults to the Media Access Control (MAC)/Reconciliation Sublayer (RS) under reset and substituting error codes when the 10GBASE-R and 10GBASE-KR PCS rules are violated.

Note: The 64B/66B decoder is available to implement the 10GBASE-R protocol.

5.2.2.7. 10GBASE-R Bit-Error Rate (BER) Checker

The 10GBASE-R BER checker block is designed in accordance with the 10GBASE-R protocol specification as described in IEEE 802.3-2008 clause-49. After block lock synchronization is achieved, the BER checker starts to count the number of invalid synchronization headers within a 125-µs period. If more than 16 invalid synchronization headers are observed in a 125-µs period, the BER checker provides the status signal `rx_enh_highber` to the FPGA fabric, indicating a high bit error rate condition.

When the optional control input `rx_enh_highber_clr_cnt` is asserted, the internal counter for the number of times the BER state machine has entered the "BER_BAD_SH" state is cleared.

When the optional control input `rx_enh_clr_errblk_count` is asserted, the internal counter for the number of times the RX state machine has entered the "RX_E" state for the 10GBASE-R protocol is cleared. In modes where the FEC block in enabled, the assertion of this signal resets the status counters within the RX FEC block.

Note: The 10GBASE-R BER checker is available to implement the 10GBASE-R protocol.

5.2.2.8. Interlaken CRC-32 Checker

The Interlaken CRC-32 checker verifies that the data transmitted has not been corrupted between the transmit PCS and the receive PCS. The CRC-32 checker calculates the 32-bit CRC for the received data and compares it against the CRC value that is transmitted within the diagnostic word. `rx_enh_crc32_err` (CRC error signal) is sent to the FPGA fabric.

5.2.2.9. RX PCS FIFO

The RX PCS FIFO interfaces between the receiver PCS and across EMIB to the RX Core FIFO and ensures reliable transfer of data and status signals. It compensates for the phase difference between the PCS_`clkout_x2(rx)` and PCS_`clkout(rx)`.

The RX PCS FIFO operates in the following modes:

1. Phase Compensation Mode
2. Register Mode

5.2.2.9.1. Phase Compensation Mode

In Phase Compensation mode, the RX PCS FIFO decouples phase variations between PCS_`clkout(rx)` and PCS_`clkout_x2(rx)`. In this mode, read and write of the RX PCS FIFO can be driven by clocks from asynchronous clock sources but must be same frequency.

5.2.2.9.2. Register Mode

The Register Mode bypasses the FIFO functionality to eliminate the FIFO latency uncertainty for applications with stringent latency requirements. This is accomplished by tying the read clock of the FIFO with its write clock.

5.2.2.10. RX Core FIFO

The RX Core FIFO provides an interface between the FPGA Fabric and across EMIB to RX PCS FIFO. It ensures reliable transfer of data and status signals.

The RX Core FIFO Operates in the following modes:

- Phase Compensation Mode
- Register Mode
- Basic Mode
- Interlaken
- 10GBase-R

5.2.2.10.1. Phase Compensation Mode

The RX Core FIFO compensates for the phase difference between the read clock and write clocks. `PCS_clkout_x2(rx)`(RX parallel clock) clocks the write side of the RX Core FIFO and `rx_coreclkin` (FPGA fabric clock or `rx_clkout`) clocks the read side of the RX Core FIFO. Depth of RX Core FIFO is constant in this mode, therefore RX FIFO flag status can be ignored. You can tie `rx_enh_data_valid` with one.

5.2.2.10.2. Register Mode

The register mode bypasses the FIFO functionality to eliminate the FIFO latency uncertainty for applications with stringent latency requirements. This is accomplished by tying the read clock of the FIFO with its write clock. In Register mode, `rx_parallel_data` (data), `rx_control` indicates whether `rx_parallel_data` is a data or control word, and `rx_enh_data_valid` (data valid) are registered at the FIFO output.

5.2.2.10.3. Basic Mode

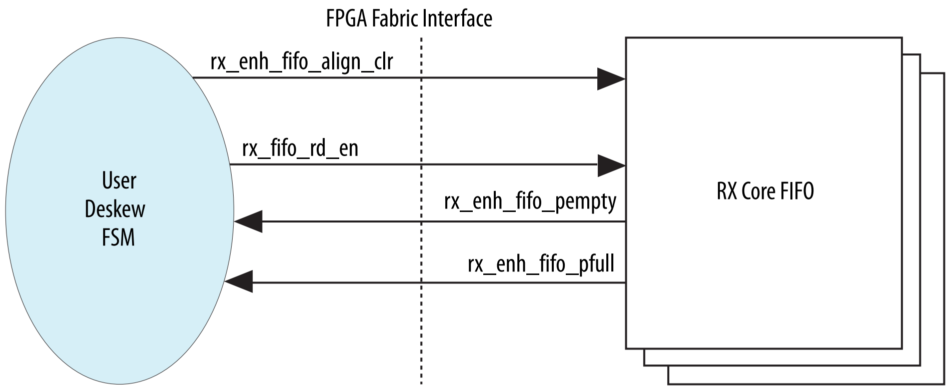
In Basic mode, the RX Core FIFO operates as an elastic buffer, where buffer depths can vary. This mode allows driving the write and read side of the RX Core FIFO with different clock frequencies. Monitor the FIFO flag to control write and read operations. For RX Core FIFO, assert `rx_data_valid` signal with `rx_fifo_pfull` signal going low.

5.2.2.10.4. Interlaken Mode

In Interlaken mode, the RX Core FIFO operates as an Interlaken deskew FIFO. To implement the deskew process, implement an FSM that controls the FIFO operation based on available FPGA input and output flags.

For example, after frame lock is achieved, data is written after the first alignment word (SYNC word) is found on that channel. As a result, `rx_fifo_pempty` (FIFO partially empty flag) of that channel goes low. You must monitor the `rx_fifo_pempty` and `rx_fifo_pfull` flags of all channels. If `rx_fifo_pempty` flags from all channels deassert before any `rx_fifo_pfull` flag asserts, which implies alignment word has been found on all lanes of the link, you start reading from all the FIFOs by asserting `rx_fifo_rd_en`. Otherwise, if a `rx_fifo_pfull` flag from any channel goes high before a `rx_fifo_pempty` flag deassertion on all channels, you must reset the FIFO by toggling the `rx_fifo_align_clr` signal and repeating the process.

Figure 202. RX Core FIFO as Interlaken Deskew FIFO



5.2.2.10.5. 10GBASE-R Mode

In 10GBASE-R mode, the RX Core FIFO operates as a clock compensation FIFO. When the block synchronizer achieves block lock, data is sent through the FIFO. Idle ordered sets (OS) are deleted and Idles are inserted to compensate for the clock difference between the RX low speed parallel clock and the FPGA fabric clock (±100 ppm for a maximum packet length of 64,000 bytes).

Idle OS Deletion

Deletion of Idles occurs in groups of four OS (when there are two consecutive OS) until the `rx_fifo_rd_pfull` flag deasserts. Every word—consisting of a lower word (LW) and an upper word (UW)—is checked for whether it can be deleted by looking at both the current and previous words.

Table 152. Conditions Under Which a Word Can be Deleted

In this table X=don't care, T=Terminate, I=Idle, and OS=Ordered Set.

Deletable	Case	Word	Previous	Current	Output	
Lower Word	1	UW	!T	X	!T	X
		LW	X	I	X	X
	2	UW	OS	X	OS	X
		LW	X	OS	X	X
Upper Word	1	UW	X	I	X	X
		LW	X	!T	X	!T
	2	UW	X	OS	X	X
		LW	X	OS	X	OS

If only one word is deleted, data shifting is necessary because the datapath is two words wide. After two words have been deleted, the FIFO stops writing for one cycle and a synchronous flag (`rx_control[8]`) appears on the next block of 8-byte data. There is also an asynchronous status signal `rx_enh_fifo_del`, which does not go through the FIFO.

Figure 203. Idle Word Deletion

This figure shows the deletion of Idle words from the receiver data stream.

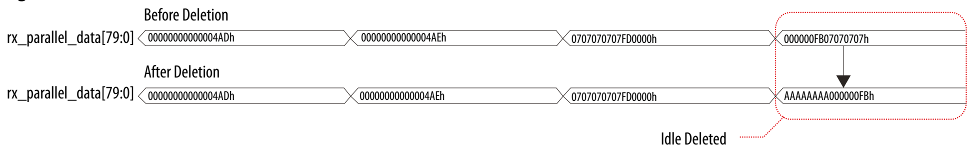
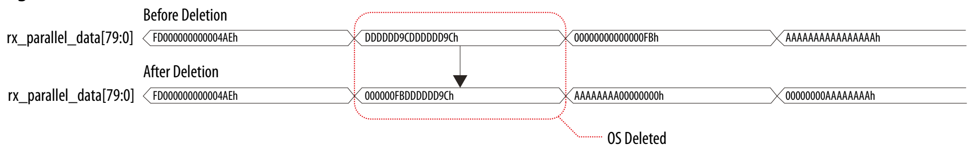


Figure 204. OS Word Deletion

This figure shows the deletion of Ordered set words in the receiver data stream.



Idle Insertion

Idle insertion occurs in groups of 8 Idles when the `rx_enh_fifo_pempty` flag is deasserted. Idles can be inserted following Idles or OS. Idles are inserted in groups of 8 bytes. Data shifting is not necessary. There is a synchronous status `rx_enh_fifo_insert` signal that is attached to the 8-byte Idles being inserted.

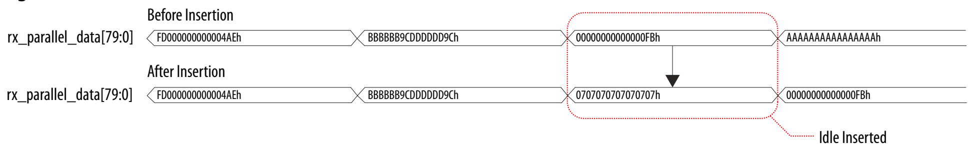
Table 153. Cases Where Two Idle Words are Inserted

In this table X=don't care, S=Start, OS=Ordered Set, I-DS=Idle in data stream, and I-In=Idle inserted. In cases 3 and 4, the Idles are inserted between the LW and UW.

Case	Word	Input	Output	
1	UW	I-DS	I-DS	I-In
	LW	X	X	I-In
2	UW	OS	OS	I-In
	LW	X	X	I-In
3	UW	S	I-In	S
	LW	I-DS	I-DS	I-In
4	UW	S	I-In	S
	LW	OS	OS	I-In

Figure 205. Idle Word Insertion

This figure shows the insertion of Idle words in the receiver data stream.



5.2.3. RX KR FEC Blocks

KR FEC Block Synchronization

You can obtain FEC block delineation for the RX KR FEC by locking onto correctly received FEC blocks with the KR FEC block synchronization.

Note: The KR FEC block synchronization is available to implement the 10GBASE-KR protocol.

KR FEC Descrambler

The KR FEC descrambler block descrambles received data to regenerate unscrambled data using the $x^{58} + x^{39} + 1$ polynomial. Before the block boundary in the KR FEC sync block is detected, the data at the input of the descrambler is sent directly to the KR FEC decoder. When the boundary is detected, the aligned word from the KR FEC sync block is descrambled with the Pseudo Noise (PN) sequence and then sent to the KR FEC decoder.

KR FEC Decoder

The KR FEC decoder block performs the FEC (2112, 2080) decoding function by analyzing the received 32 65-bit blocks for errors. It can correct burst errors of 11 bits or less per FEC block.

KR FEC RX Gearbox

The KR FEC RX gearbox block adapts the PMA data width to the larger bus width of the PCS channel. It supports a 64:65 ratio.

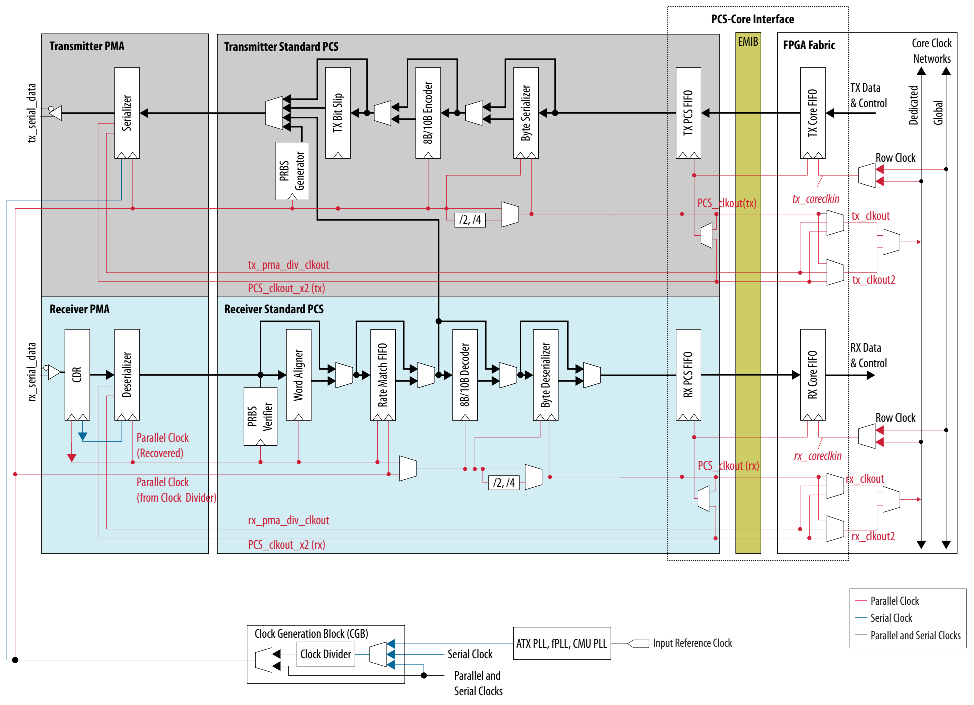
Transcode Decoder

The transcode decoder block performs the 65-bit to 64B/66B reconstruction function by regenerating the 64B/66B synchronization header.

5.3. Intel Stratix 10 Standard PCS Architecture

The standard PCS can operate at a data rate of up to 12 Gbps. Protocols such as PCI-Express, CPRI 4.2+, GigE, IEEE 1588 are supported in Hard PCS while the other protocols can be implemented using Basic/Custom (Standard PCS) transceiver configuration rules. The standard PCS supports data rates up to 10 Gbps when using devices with transceiver speed grades 2 and 3, and 12 Gbps when using devices with transceiver speed grade 1. Refer to the *Intel Stratix 10 Device Datasheet* for details about the maximum datarate supported by each transceiver speed grade.

Figure 206. Standard PCS Datapath Diagram



Related Information

Intel Stratix 10 Device Datasheet

5.3.1. Transmitter Datapath

5.3.1.1. TX Core FIFO

The TX Core FIFO operates in the following modes:

- Phase Compensation Mode
- Register Mode
- Basic Mode

Related Information

TX Core FIFO on page 353

5.3.1.2. TX PCS FIFO

The TX PCS FIFO operates in Phase Compensation Mode.

For more information, refer to the *TX PCS FIFO* section.

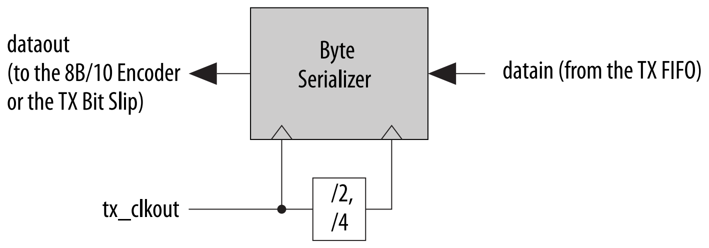
Related Information

TX PCS FIFO on page 354

5.3.1.3. Byte Serializer

In certain applications, the FPGA fabric cannot operate at the same clock rate as the transmitter channel (PCS) because the transmitter channel is capable of operating at higher clock rates compared to the FPGA fabric. The byte serializer allows the transmitter channel to operate at higher data rates while keeping the FPGA fabric interface clock rate below its maximum limit. This is accomplished by increasing the channel width two or four times (FPGA fabric-to-PCS interface width) and dividing the clock (tx_clkout) rate by 2 or 4. The byte serializer can be disabled, or operate in Serialize x2 or Serialize x4 modes.

Figure 207. Byte Serializer Block Diagram



Related Information

- Implementing the Transceiver PHY Layer in L-Tile/H-Tile on page 30
- Resetting Transceiver Channels on page 318

5.3.1.3.1. Bonded Byte Serializer

The bonded byte serializer is available in Intel Stratix 10 devices, and is used in applications such as PIPE, CPRI, and custom applications where multiple channels are grouped together. The bonded byte serializer is implemented by bonding all the control signals to prevent skew induction between channels during byte serialization. In this configuration, one of the channels acts as master and the remaining channels act as slaves.

5.3.1.3.2. Byte Serializer Disabled Mode

In disabled mode, the byte serializer is bypassed. The data from the TX FIFO is directly transmitted to the 8B/10B encoder, TX Bitslip, or Serializer, depending on whether or not the 8B/10B encoder and TX Bitslip are enabled. Disabled mode is used in low-speed applications such as GigE, where the FPGA fabric and the TX standard PCS can operate at the same clock rate.

5.3.1.3.3. Byte Serializer Serialize x2 Mode

The serialize x2 mode is used in high-speed applications such as the PCIe Gen1 or Gen2 protocol implementation, where the FPGA fabric cannot operate as fast as the TX PCS.

In serialize x2 mode, the byte serializer serializes 16-bit, 20-bit (when 8B/10B encoder is not enabled), 32-bit, and 40-bit (when 8B/10B encoder is not enabled) input data into 8-bit, 10-bit, 16-bit, and 20-bit data, respectively. As the parallel data width from the TX FIFO is halved, the clock rate is doubled.

After byte serialization, the byte serializer forwards the least significant word first followed by the most significant word. For example, if the FPGA fabric-to-PCS Interface width is 32, the byte serializer forwards `tx_parallel_data[15:0]` first, followed by `tx_parallel_data[31:16]`.

Related Information

PCI Express (PIPE) on page 164

5.3.1.3.4. Byte Serializer Serialize x4 Mode

The serialize x4 mode is used in high-speed applications such as the PCIe Gen3 protocol mode, where the FPGA fabric cannot operate as fast as the TX PCS.

In serialize x4 mode, the byte serializer serializes 32-bit data into 8-bit data. As the parallel data width from the TX FIFO is divided four times, the clock rate is quadrupled.

After byte serialization, the byte serializer forwards the least significant word first followed by the most significant word. For example, if the FPGA fabric-to-PCS Interface width is 32, the byte serializer forwards `tx_parallel_data[7:0]` first, followed by `tx_parallel_data[15:8]`, `tx_parallel_data[23:16]` and `tx_parallel_data[31:24]`.

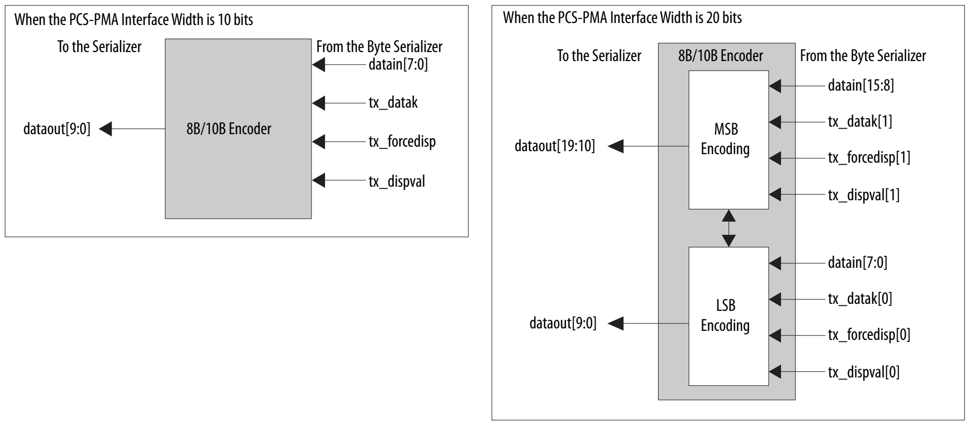
Related Information

PCI Express (PIPE) on page 164

5.3.1.4. 8B/10B Encoder

The 8B/10B encoder takes in 8-bit data and 1-bit control as input and converts them into a 10-bit output. The 8B/10B encoder automatically performs running disparity check for the 10-bit output. Additionally, the 8B/10B encoder can control the running disparity manually using the `tx_forcedisp` and `tx_dispval` ports.

Figure 208. 8B/10B Encoder Block Diagrams

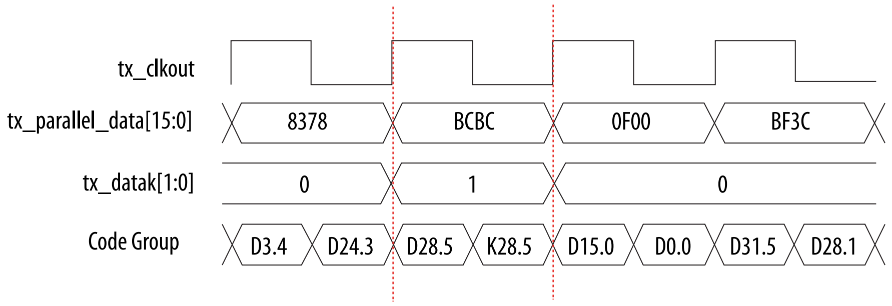


When the PCS-PMA interface width is 10 bits, one 8B/10B encoder is used to convert the 8-bit data into a 10-bit output. When the PCS-PMA interface width is 20 bits, two cascaded 8B/10B encoders are used to convert the 16-bit data into a 20-bit output. The first eight bits (LSByte) is encoded by the first 8B/10B encoder and the next eight bits (MSByte) is encoded by the second 8B/10B encoder. The running disparity of the LSByte is calculated first and passed on to the second encoder to calculate the running disparity of the MSByte.

Note: You cannot enable the 8B/10B encoder when the PCS-PMA interface width is 8 bits or 16 bits.

5.3.1.4.1. 8B/10B Encoder Control Code Encoding

Figure 209. Control Code Encoding Diagram



The `tx_datak` signal indicates whether the 8-bit data being sent at the `tx_parallel_data` port should be a control word or a data word. When `tx_datak` is high, the 8-bit data is encoded as a control word (Kx.y). When `tx_datak` is low, the 8-bit data is encoded as a data word (Dx.y). Depending upon the PCS-PMA interface width, the width of `tx_datak` is either 1 bit or 2 bits. When the PCS-PMA interface width is 10 bits, `tx_datak` is a 1-bit word. When the PCS-PMA interface width is 20 bits, `tx_datak` is a 2-bit word. The LSB of `tx_datak` corresponds to the LSByte of the input data sent to the 8B/10B encoder and the MSB corresponds to the MSByte of the input data sent to the 8B/10B encoder.

Related Information

Refer to *Specifications & Additional Information* for more information about 8B/10B encoder codes.

5.3.1.4.2. 8B/10B Encoder Reset Condition

The `tx_digitalreset` signal resets the 8B/10B encoder. During the reset condition, the 8B/10B encoder outputs K28.5 continuously until `tx_digitalreset` goes low.

5.3.1.4.3. 8B/10B Encoder Idle Character Replacement Feature

The idle character replacement feature is used in protocols such as Gigabit Ethernet, which requires the running disparity to be maintained during Idle sequences. During these Idle sequences, the running disparity has to be maintained such that the first byte of the next packet always starts when the running disparity of the current packet is negative.

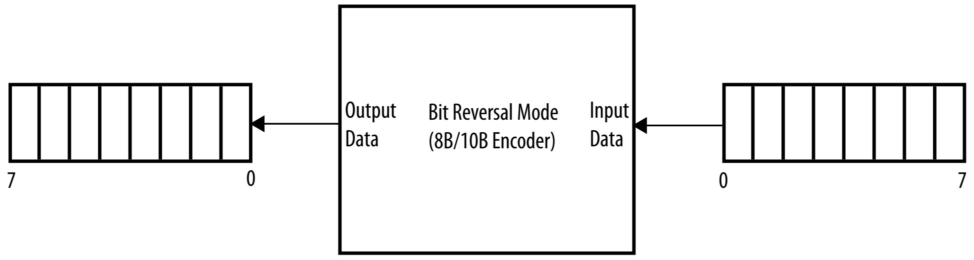
When an Ordered Set, which consists of two code-groups, is received by the 8B/10B encoder, the second code group is converted into /I1/ or /I2 so that the final running disparity of the data code-group is negative. The first code group is /K28.5/ and the second code group is a data code-group other than /D21.5/ or /D2.2/. The ordered set /I1/ (/K28.5/D5.6/) is used to flip the running disparity and /I2/ (/K28.5/D16.2/) is used to preserve the running disparity.

5.3.1.4.4. 8B/10B Encoder Current Running Disparity Control Feature

The 8B/10B encoder performs a running disparity check on the 10-bit output data. The running disparity can also be controlled using `tx_forcedisp` and `tx_dispval`. When the PCS-PMA interface width is 10 bits, `tx_forcedisp` and `tx_dispval` are one bit each. When the PCS-PMA interface width is 20 bits, `tx_forcedisp` and `tx_dispval` are two bits each. The LSB of `tx_forcedisp` and `tx_dispval` corresponds to the LSByte of the input data and the MSB corresponds to the MSByte of the input data.

5.3.1.4.5. 8B/10B Encoder Bit Reversal Feature

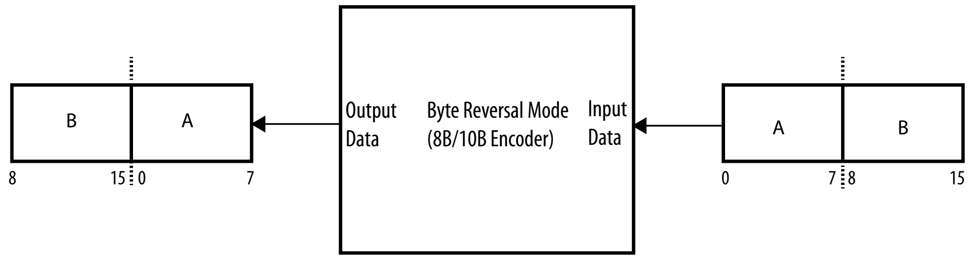
Figure 210. 8B/10B Encoder Bit Reversal Feature



The bit reversal feature reverses the order of the bits of the input data. Bit reversal is performed at the output of the 8B/10B Encoder and is available even when the 8B/10B Encoder is disabled. For example, if the input data is 20-bits wide, bit reversal switches bit [0] with bit [19], bit [1] with bit [18] and so on.

5.3.1.4.6. 8B/10B Encoder Byte Reversal Feature

Figure 211. 8B/10B Encoder Byte Reversal Feature



The byte reversal feature is available only when the PCS-PMA interface width is 16 bits or 20 bits. Byte reversal is performed at the output of the 8B/10B Encoder and is available even when the 8B/10B Encoder is disabled. This feature swaps the LSByte with the MSByte. For example, when the PCS-PMA interface width is 16-bits, [7:0] bits (LSByte) gets swapped with [15:8] bits (MSByte).

5.3.1.5. Polarity Inversion Feature

The polarity inversion feature is used in situations where the positive and the negative signals of a serial differential link are erroneously swapped during board layout. You can control this feature through `tx_polinv` port, by enabling both the **Enable tx_polinv port** and **Enable TX Polarity Inversion** options under the **Standard PCS** tab of the Native PHY IP Core. The polarity inversion feature inverts the value of each bit of the input data. For example, if the input data is 00101001, then the data gets changed to 11010110 after polarity inversion.

5.3.1.6. TX Bit Slip

The TX bit slip allows the word boundary to be controlled by `tx_std_bitslipboundarysel`. The TX bit slip feature is used in applications, such as CPRI, which has a data rate greater than 6 Gbps. The maximum number of the supported bit slips is PCS data width-1 and the slip direction is from MSB to LSB and from current to previous word.

5.3.2. Receiver Datapath

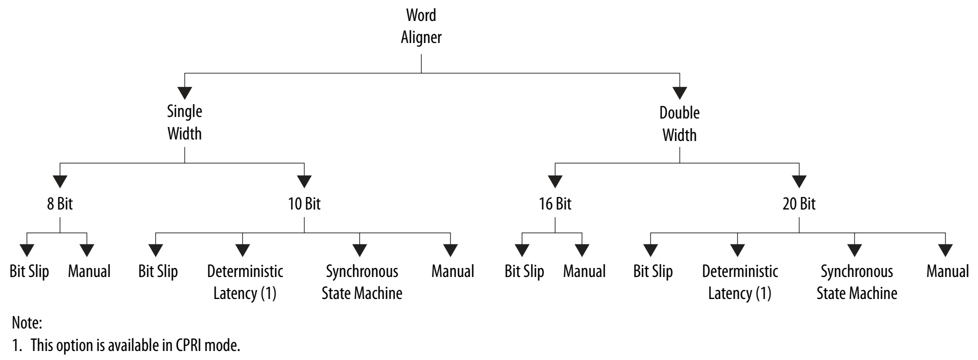
5.3.2.1. Word Aligner

The word aligner receives the serial data from the PMA and realigns the serial data to have the correct word boundary according to the word alignment pattern configured. This word alignment pattern can be 7, 8, 10, 16, 20, 32, or 40 bits in length.

Depending on your PCS-PMA interface width, the word aligner can be configured in one of the following modes:

- Bit slip
- Manual alignment
- Synchronous state machine
- Deterministic latency

Figure 212. Word Aligner Conditions and Modes


Word Aligner
Single Width
Double Width
8 Bit
10 Bit
16 Bit
20 Bit
Bit Slip
Manual
Bit Slip
Deterministic Latency (1)
Synchronous State Machine
Manual
Bit Slip
Manual
Bit Slip
Deterministic Latency (1)
Synchronous State Machine
Manual
Note:
1. This option is available in CPRI mode.

5.3.2.1.1. Word Aligner Bitslip Mode

In bitslip mode, the word aligner operation is controlled by `rx_bitslip`, which has to be held for two parallel clock cycles. At every rising edge of `rx_bitslip`, the bitslip circuitry slips one bit into the received data stream, effectively shifting the word boundary by one bit. Pattern detection is not used in bitslip mode; therefore, `rx_syncstatus` is not valid in this mode.

Related Information

Intel Stratix 10 (L/H-Tile) Word Aligner Bitslip Calculator
Use this tool to calculate the number of slips you require to achieve alignment based on the word alignment pattern and length.

5.3.2.1.2. Word Aligner Manual Mode

In manual alignment mode, the word aligner operation is controlled by `rx_std_wa_patternalign`. The word aligner operation is edge-sensitive or level-sensitive to `rx_std_wa_patternalign`, depending upon the PCS-PMA interface width selected.

Table 154. Word Aligner `rx_std_wa_patternalign` Behavior

PCS-PMA Interface Width	`rx_std_wa_patternalign` Behavior
8	Rising edge sensitive
10	Level sensitive
16	Rising edge sensitive
20	Rising edge sensitive

If `rx_std_wa_patternalign` is asserted, the word aligner looks for the programmed word alignment pattern in the received data stream. It updates the word boundary if it finds the word alignment pattern in a new word boundary. If `rx_std_wa_patternalign` is deasserted, the word aligner maintains the current word boundary even when it sees the word alignment pattern in a new word boundary.

The `rx_syncstatus` and `rx_patterndetect` signals, with the same latency as the datapath, are forwarded to the FPGA fabric to indicate the word aligner status.

After receiving the first word alignment pattern after `rx_std_wa_patternalign` is asserted, both `rx_syncstatus` and `rx_patterndetect` are driven high for one parallel clock cycle. Any word alignment pattern received thereafter in the same word boundary causes only `rx_patterndetect` to assert for one clock cycle. Any word alignment pattern received thereafter in a different word boundary causes the word aligner to re-align to the new word boundary only if `rx_std_wa_patternalign` is asserted. The word aligner asserts `rx_syncstatus` for one parallel clock cycle whenever it re-aligns to the new word boundary.

5.3.2.1.3. Word Aligner Synchronous State Machine Mode

In synchronous state machine mode, when the programmed number of valid synchronization code groups or ordered sets is received, `rx_syncstatus` is driven high to indicate that synchronization is acquired. The `rx_syncstatus` signal is constantly driven high until the programmed number of erroneous code groups is received without receiving intermediate good groups, after which `rx_syncstatus` is driven low.

The word aligner indicates loss of synchronization (`rx_syncstatus` remains low) until the programmed number of valid synchronization code groups are received again.

5.3.2.1.4. Word Aligner Deterministic Latency Mode

In deterministic latency mode, the state machine removes the bit level latency uncertainty. The deserializer of the PMA creates the bit level latency uncertainty as it comes out of reset.

The PCS performs pattern detection on the incoming data from the PMA. The PCS aligns the data, after it indicates to the PMA the number of serial bits to clock slip the boundary.

If the incoming data has to be realigned, `rx_std_wa_patternalign` must be reasserted to initiate another pattern alignment. Asserting `rx_std_wa_patternalign` can cause the word align to lose synchronization if already achieved. This may cause `rx_syncstatus` to deassert.

5.3.2.1.5. Word Aligner Pattern Length for Various Word Aligner Modes

Table 155. PCS-PMA Interface Widths and Protocol Implementations

PCS-PMA Interface Width	Protocol Implementations
8	Basic
10	• Basic • Basic rate match • CPRI • PCIe Gen1 and Gen2 • GigE
16	Basic
20	• CPRI • Basic • Basic rate match

Table 156. Word Aligner Pattern Length for Various Word Aligner Modes

PCS-PMA Interface Width	Supported Word Aligner Modes	Supported Word Aligner Pattern Lengths	`rx_std_wa_patternalign` behavior	`rx_syncstatus` behavior	`rx_patterndetect` behavior
8	Bit slip	8	`rx_std_wa_patternalign` has no effect on word alignment. The single width word aligner updates the word boundary, only when you assert and deassert the`rx_bitslip` signal.	N/A	N/A
	Manual	8, 16	Word alignment is controlled by `rx_std_wa_patternalign` and is edge-sensitive to this signal.	Asserted high for one parallel clock cycle when the word aligner aligns to a new boundary.	Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.

continued...

Send Feedback

PCS-PMA Interface Width	Supported Word Aligner Modes	Supported Word Aligner Pattern Lengths	`rx_std_wa_patternalign` behavior	`rx_syncstatus` behavior	`rx_patterndetect` behavior
10	Bit slip	7	`rx_std_wa_patternalign` has no effect on word alignment. The single width word aligner updates the word boundary, only when you assert and deassert the `rx_bitslip` signal.	N/A	N/A
	Manual	7, 10	Word alignment is controlled by `rx_std_wa_patternalign` and is level-sensitive to this signal.	Asserted high for one parallel clock cycle when the word aligner aligns to a new boundary.	Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.
	Deterministic latency (CPRI mode only)	10	Word alignment is controlled by `rx_std_wa_patternalign` (edge-sensitive to this signal) and the state machine works in conjunction with PMA to achieve deterministic latency on the RX path for CPRI and OBSAI applications.	—	—
	Synchronous State Machine	7, 10	`rx_std_wa_patternalign` has no effect on word alignment.	Stays high as long as the synchronization conditions are satisfied.	Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.
16	Bit slip	16	`rx_std_wa_patternalign` has no effect on word alignment. The double width word aligner updates the word boundary, only when you assert and deassert the `rx_bitslip` signal.	N/A	N/A
	Manual	8, 16, 32	Word alignment is controlled by rising-edge of `rx_std_wa_patternalign`.	Stays high after the word aligner aligns to the word alignment pattern. Goes low on receiving a rising edge on `rx_std_wa_patternalign` until a	Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.

continued...

PCS-PMA Interface Width	Supported Word Aligner Modes	Supported Word Aligner Pattern Lengths	`rx_std_wa_patternalign` behavior	`rx_syncstatus` behavior	`rx_patterndetect` behavior
				new word alignment pattern is received.	
20	Bit slip	7	`rx_std_wa_patternalign` has no effect on word alignment. The double width word aligner updates the word boundary, only when you assert and deassert the `rx_bitslip` signal.	N/A	N/A
	Manual	7, 10, 20, 40	Word alignment is controlled by rising edge of `rx_std_wa_patternalign`.	Stays high after the word aligner aligns to the word alignment pattern. Goes low on receiving a rising edge on `rx_std_wa_patternalign` until a new word alignment pattern is received.	Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.
	Deterministic latency (CPRI mode only)	10	Word alignment is controlled by `rx_std_wa_patternalign` (edge-sensitive to this signal) and the deterministic latency state machine which controls the PMA to achieve deterministic latency on the RX path for CPRI and OBSAI applications.	—	—
	Synchronous State Machine	7, 10, 20	FPGA fabric-driven `rx_std_wa_patternalign` signal has no effect on word alignment.	Stays high as long as the synchronization conditions are satisfied.	Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.

5.3.2.1.6. Word Aligner RX Bit Reversal Feature

The RX bit reversal feature reverses the order of the data received from the PMA. It is performed at the output of the word aligner and is available even when the word aligner is disabled. If the data received from the PMA is a 10-bit data width, the bit reversal feature switches bit [0] with bit [9], bit [1] with bit [8], and so on. For example, if the 10-bit data is 1000010011, the bit reversal feature, when enabled, changes the data to 1100100001.

5.3.2.1.7. Word Aligner RX Byte Reversal Feature

The RX byte reversal feature is available only when the PCS-PMA interface width is 16 bits or 20 bits. This feature reverses the order of the data received from the PMA. RX byte reversal reverses the LSByte of the received data with its MSByte and vice versa. If the data received is 20-bits, bits[0..9] are swapped with bits[10..20] so that the resulting 20-bit data is [[10..20],[0..9]]. For example, if the 20-bit data is 11001100001000011111, the byte reversal feature changes the data to 10000111111100110000.

5.3.2.2. RX Polarity Inversion Feature

The RX polarity inversion feature inverts each bit of the data received from the PMA. If the data received is a 10-bit data. Bit[0] content is inverted to its complement, ~bit[0], bit[1] is inverted to its complement, ~bit[1], bit[2] is inverted to its complement, ~bit[2], and so on. For example, if the 10-bit data is 1111100000, the polarity inversion feature inverts it to 0000011111.

5.3.2.3. Rate Match FIFO

The rate match FIFO compensates for the frequency differences between the local clock and the recovered clock up to ± 300 ppm by inserting and deleting skip/idle characters in the data stream. The rate match FIFO has several different protocol specific modes of operation. All of the protocol specific modes depend upon the following parameters:

- Rate match deletion—occurs when the distance between the write and read pointers exceeds a certain value due to write clock having a higher frequency than the read clock.
- Rate match insertion—occurs when the distance between the write and the read pointers becomes less than a certain value due to the read clock having a higher frequency than the write clock.
- Rate match full—occurs when the write pointer wraps around and catches up to the slower-advancing read pointer.
- Rate match empty—occurs when the read pointer catches up to the slower-advancing write pointer.

Rate match FIFO operates in the following modes:

- Basic 10-bit PMA
- Basic 20-bit PMA
- GbE
- PIPE
- PIPE 0 ppm

Related Information

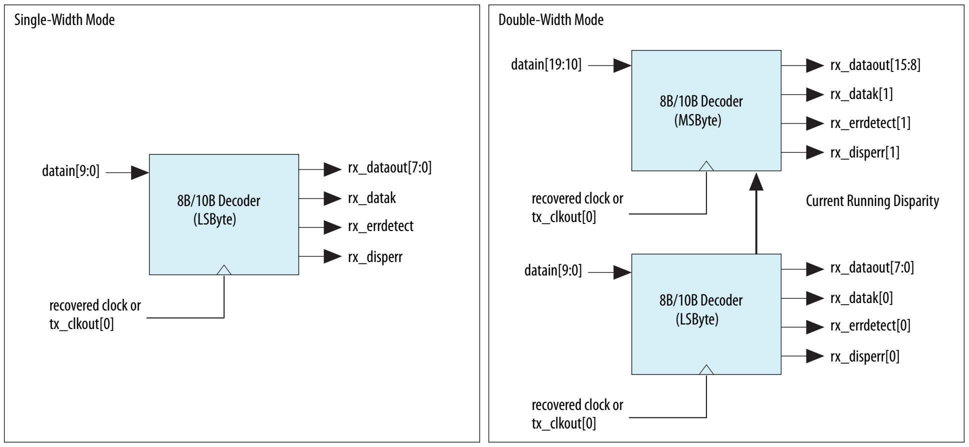
- Rate Match FIFO in Basic (Single Width) Mode on page 119
- Rate Match FIFO Basic (Double Width) Mode on page 121
- How to Implement PCI Express (PIPE) in Intel Stratix 10 Transceivers on page 183
- PCI Express (PIPE) on page 164

5.3.2.4. 8B/10B Decoder

The general functionality for the 8B/10B decoder is to take a 10-bit encoded value as input and produce an 8-bit data value and a 1-bit control value as output. In configurations with the rate match FIFO enabled, the 8B/10B decoder receives data from the rate match FIFO. In configurations with the rate match FIFO disabled, the 8B/10B decoder receives data from the word aligner. The 8B/10B decoder operates in two conditions:

- When the PCS-PMA interface width is 10 bits and PCS-Core interface to FPGA fabric width is 8 bits
- When the PCS-PMA interface width is 20 bits and PCS-Core interface to FPGA fabric width is 16 bits

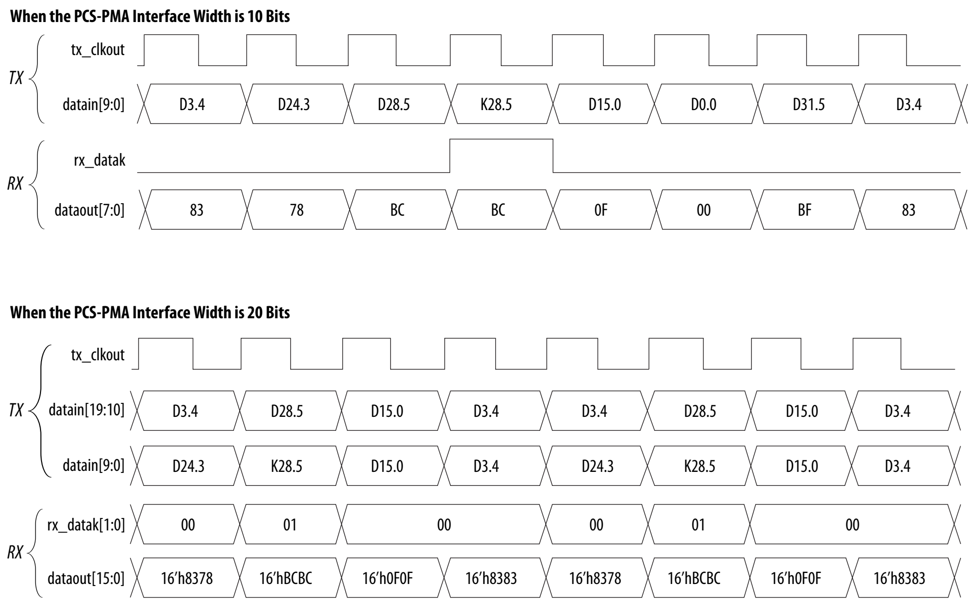
Figure 213. 8B/10B Decoder in Single-Width and Double-Width Mode



When the PCS-PMA interface width is 10 bits, only one 8B/10B decoder is used to perform the conversion. When the PCS-PMA interface width is 20 bits, two cascaded 8B/10B decoders are used. The 10-bit LSByte of the received 20-bit encoded data is decoded first and the ending running disparity is forwarded to the 8B/10B decoder responsible for decoding the 10-bit MSByte. The cascaded 8B/10B decoder decodes the 20- bit encoded data into 16-bit data + 2-bit control identifier. The MSB and LSB of the 2-bit control identifier correspond to the MSByte and LSByte of the 16-bit decoded data code group. The decoded data is fed to the byte deserializer or the RX PCS FIFO.

5.3.2.4.1. 8B/10B Decoder Control Code Encoding

Figure 214. 8B/10B Decoder in Control Code Group Detection



The 8B/10B decoder indicates whether the decoded 8-bit code group is a data or control code group on `rx_datak` . If the received 10-bit code group is one of the 12 control code groups (/Kx.y/) specified in the IEEE 802.3 specification, `rx_datak` is driven high. If the received 10-bit code group is a data code group (/Dx.y/), `rx_datak` is driven low.

5.3.2.4.2. 8B/10B Decoder Running Disparity Checker Feature

Running disparity checker resides in 8B/10B decoder module. This checker checks the current running disparity value and error based on the rate match output. `rx_runningdisp` and `rx_disperr` indicate positive or negative disparity and disparity errors, respectively.

5.3.2.5. PRBS Verifier

Refer to the *PRBS Verifier* section.

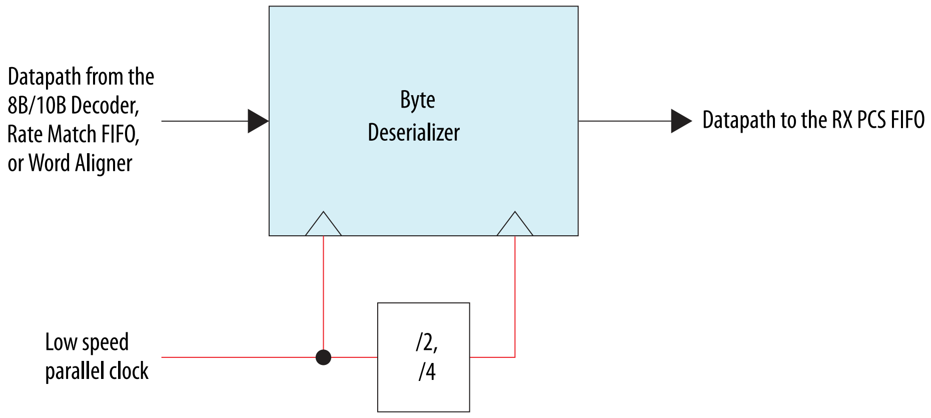
Related Information

PRBS Pattern Verifier on page 390

5.3.2.6. Byte Deserializer

The byte deserializer allows the transceiver to operate at data rates higher than those supported by the FPGA fabric. It deserializes the recovered data by multiplying the data width two or four times, depending upon the deserialization mode selected. The byte deserializer is optional in designs that do not exceed the FPGA fabric interface frequency upper limit. You can bypass the byte deserializer by disabling it in the Native PHY IP Core. The byte deserializer operates in disabled, deserialize x2, or deserialize x4 modes.

Figure 215. Byte Deserializer Block Diagram


Datapath from the 8B/10B Decoder, Rate Match FIFO, or Word Aligner
Byte Deserializer
Datapath to the RX PCS FIFO
Low speed parallel clock
/2, /4

5.3.2.6.1. Byte Deserializer Disabled Mode

In disabled mode, the byte deserializer is bypassed. The data from the 8B/10B decoder, rate match FIFO, or word aligner is directly transmitted to the RX PCS FIFO, depending on whether or not the 8B/10B decoder and rate match FIFO are enabled. Disabled mode is used in low-speed applications such as GigE, where the FPGA fabric and the PCS can operate at the same clock rate.

5.3.2.6.2. Byte Deserializer Deserialize x2 Mode

The deserialize x2 mode is used in high-speed applications such as the PCIe Gen1 or Gen2 protocol implementation, where the FPGA fabric cannot operate as fast as the RX PCS.

In deserialize x2 mode, the byte deserializer deserializes 8-bit, 10-bit (when the 8B/10B encoder is not enabled), 16-bit, and 20-bit (when the 8B/10B encoder is not enabled) input data into 16-bit, 20-bit, 32-bit, and 40-bit data, respectively. As the parallel data width from the word aligner is doubled, the clock rate is halved.

Note: Depending on when the receiver PCS logic comes out of reset, the byte ordering at the output of the byte deserializer may or may not match the original byte ordering of the transmitted data. The byte misalignment resulting from byte deserialization is unpredictable because it depends on which byte is being received by the byte deserializer when it comes out of reset. Implement byte ordering logic in the FPGA fabric to retain the order of transmitted data.

5.3.2.6.3. Byte Deserializer Deserialize x4 Mode

The deserialize x4 mode is used in high-speed applications where the FPGA fabric cannot operate as fast as the RX PCS.

In deserialize x4 mode, the byte deserializer deserializes 8-bit data into 32-bit data. As the parallel data width from the word aligner is quadrupled, the clock rate is divided four times.

Note: Depending on when the receiver PCS logic comes out of reset, the byte ordering at the output of the byte deserializer may or may not match the original byte ordering of the transmitted data. The byte misalignment resulting from byte deserialization is unpredictable because it depends on which byte is being received by the byte deserializer when it comes out of reset. Implement byte ordering logic in the FPGA fabric to retain the order of transmitted data.

5.3.2.6.4. Bonded Byte Deserializer

The bonded byte deserializer is also available for channel-bundled applications such as PIPE. In this configuration, the control signals of the byte deserializers of all the channels are bonded together. A master channel controls all the other channels to prevent skew between the channels.

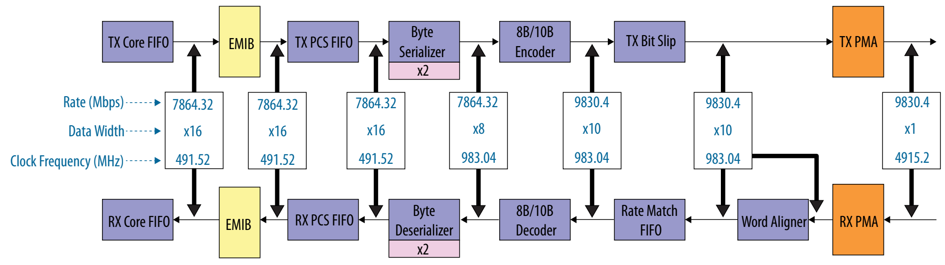
5.3.2.6.5. Byte Ordering Register-Transfer Level (RTL)

Intel Stratix 10 L-Tile and H-tile devices take advantage of the Standard PCS block to implement sub-10G CPRI and Ethernet protocols.

Figure 216. Byte Ordering in a Duplex Implementation with TX and RX Channels

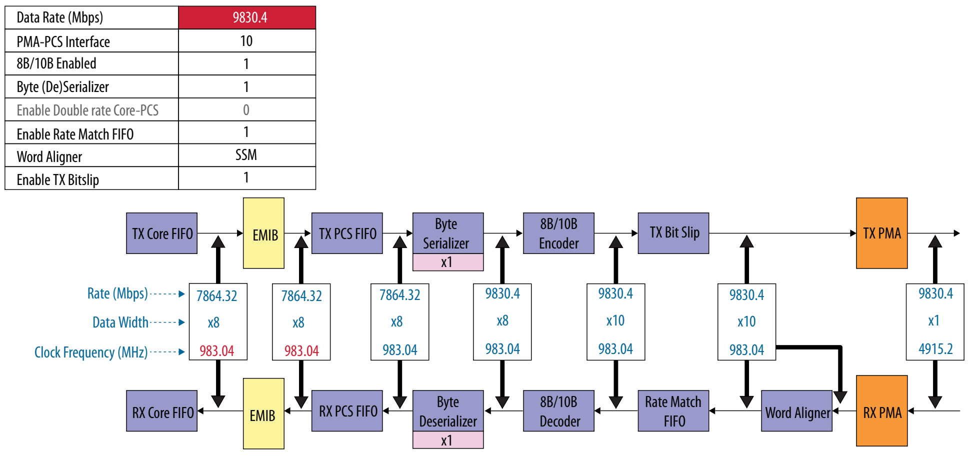
Both channels are configured with the same settings.

Data Rate (Mbps)	9830.4
PMA-PCS Interface	10
8B/10B Enabled	1
Byte (De)Serializer	2
Enable Double rate Core-PCS	0
Enable Rate Match FIFO	1
Word Aligner	SSM
Enable TX Bitslip	1



The maximum clock speed of the FPGA core limits the FIFO interface. In situations where the FIFO clock speed exceeds the FPGA core clock speed specification, the byte serializer block scales the data width (x2 or x4). The following figure demonstrates the behavior when there is a core clock speed violation.

Figure 217. Byte Ordering with a Core Clock Speed Violation

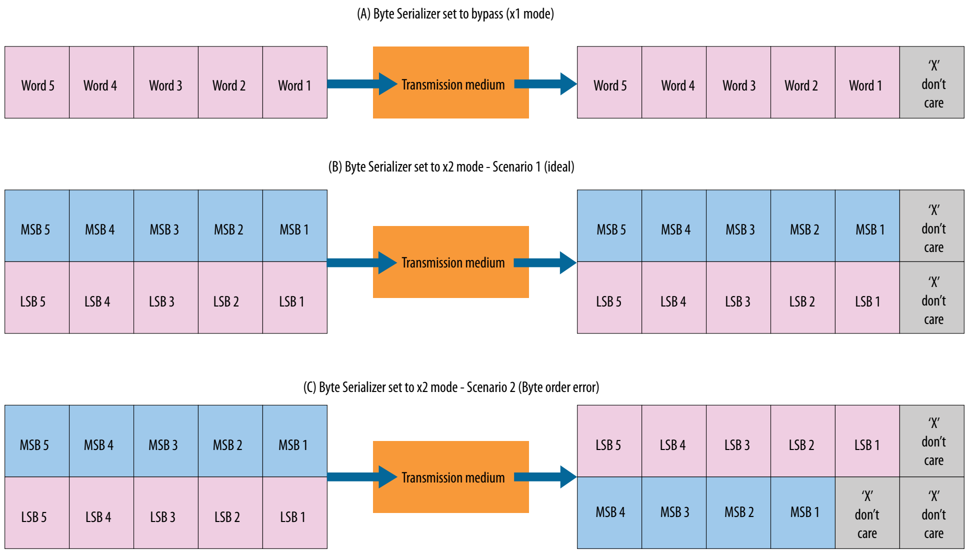


Data Rate (Mbps)	9830.4
PMA-PCS Interface	10
8B/10B Enabled	1
Byte (De)Serializer	1
Enable Double rate Core-PCS	0
Enable Rate Match FIFO	1
Word Aligner	SSM
Enable TX Bitslip	1

5.3.2.6.6. Byte Serializer Effects on Data Propagation at the RX Side

In either duplex mode (RX and TX adjacent) or simplex mode, where RX is at the far end compared to TX, the two channels initialize asynchronously. The FPGA core logic can solve this for duplex, but adds more RTL and latency. Additionally, signal integrity/ power integrity (SI/PI), power delivery network (PDN), and thermal issues contribute to misalignment.

Figure 218. TX-to-RX Word Scenarios



In scenario (A), the byte serializer is set to the x1 (bypass) mode. The word stream from the TX progresses to the RX, and the order of the words is not impacted in this scenario. Initial don't cares in the RX are disregarded.

In scenario (B), the byte serializer is set to the x2 (double width) mode. The word steam from the TX progresses to the RX and aligns ideally after the initial don't cares are disregarded. The position of LSB1 and MSB1 are correctly aligned to set the pattern for subsequent words.

In scenario (C), the byte serializer is also set to the x2 (double width) mode. However, LSB1 occupies an incorrect position, which forces the remaining words to misalign correspondingly. The align_byteorder.v RTL file resolves the byte ordering misalignment in the FPGA core.

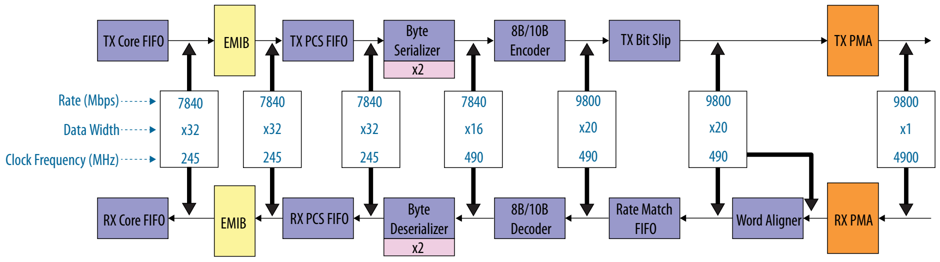
The RTL block is gated by `reset_n` and `rx_syncstatus`, and must be clocked by `rx_clockout`. The `rx_parallel_data` signal is the primary input for the block monitors and must be accompanied by `rx_datak` which captures the location of the control character. In Ethernet applications, K28.5 is a frequently used control character. The objective of the RTL is to bind the control character to the LSB.

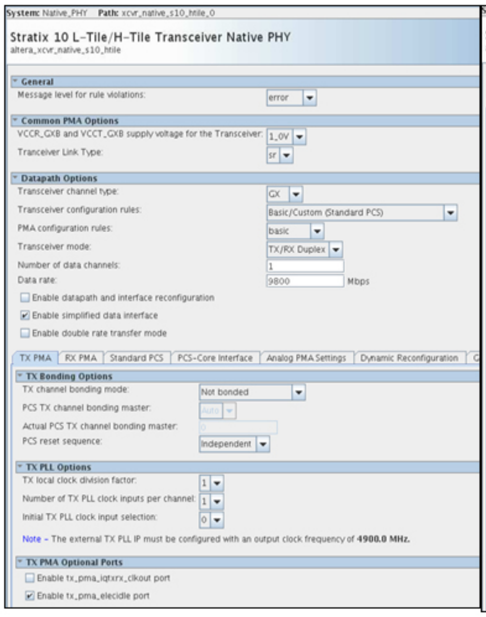
From the protocol perspective, this is the logic that eliminates the misalignment. When `rx_parallel_data` enters the byte ordering RTL with the correct alignment, it is automatically bypassed. Because the RTL logic is constantly searching for the control character, it continuously binds the same control character to the LSB. The continuous and real time correction is required for (SI/PI), PDN, and thermal conditions that produce the byte ordering misalignment.

Figure 219. Simulation Setup

This PCS configuration is an example of the RTL solution.

Data Rate (Mbps)	9800
PMA-PCS Interface	20
8B/10B Enabled	1
Byte (De)Serializer	2
Enable Double rate Core-PCS	0
Enable Rate Match FIFO	1
Word Aligner	SSM
Enable TX Bitslip	1





System: Native_PHY Path: xcvr_native_s10_htile_0

Stratix 10 L-Tile/H-Tile Transceiver Native PHY
altera_xcvr_native_s10_htile

General
Message level for rule violations: error

Common PMA Options
VCCR_GXB and VCCT_GXB supply voltage for the Transceiver: 1_0V
Tranceiver Link Type: sr

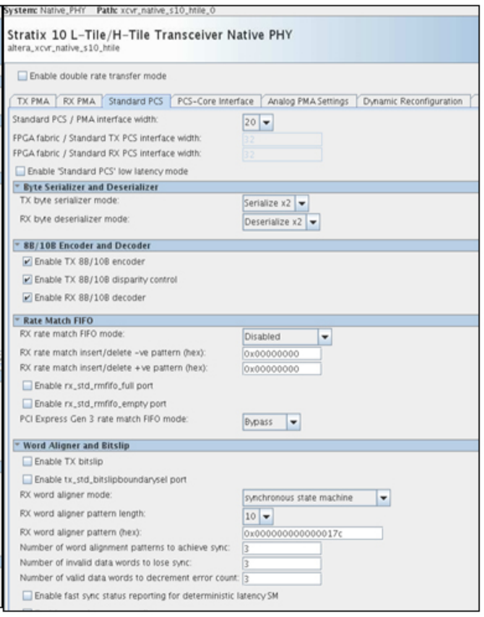
Datapath Options
Transceiver channel type: GX
Transceiver configuration rules: Basic/Custom (Standard PCS)
PMA configuration rules: basic
Transceiver mode: TX/RX Duplex
Number of data channels: 1
Data rate: 9800 Mbps
☐ Enable datapath and interface reconfiguration
☑ Enable simplified data interface
☐ Enable double rate transfer mode

TX PMA | RX PMA | Standard PCS | PCS-Core Interface | Analog PMA Settings | Dynamic Reconfiguration

TX Bonding Options
TX channel bonding mode: Not bonded
PCS TX channel bonding master: Auto
Actual PCS TX channel bonding master: 0
PCS reset sequence: Independent

TX PLL Options
TX local clock division factor: 1
Number of TX PLL clock inputs per channel: 1
Initial TX PLL clock input selection: 0
Note - The external TX PLL IP must be configured with an output clock frequency of **4900.0 MHz.**

TX PMA Optional Ports
☐ Enable tx_pma_iqtxrx_clkout port
☑ Enable tx_pma_elecidle port



System: Native_PHY Path: xcvr_native_s10_htile_0

Stratix 10 L-Tile/H-Tile Transceiver Native PHY
altera_xcvr_native_s10_htile

☐ Enable double rate transfer mode

TX PMA | RX PMA | Standard PCS | PCS-Core Interface | Analog PMA Settings | Dynamic Reconfiguration

Standard PCS / PMA interface width: 20
FPGA fabric / Standard TX PCS interface width: 32
FPGA fabric / Standard RX PCS interface width: 32
☐ Enable 'Standard PCS' low latency mode

Byte Serializer and Deserializer
TX byte serializer mode: Serialize x2
RX byte deserializer mode: Deserialize x2

8B/10B Encoder and Decoder
☑ Enable TX 8B/10B encoder
☑ Enable TX 8B/10B disparity control
☑ Enable RX 8B/10B decoder

Rate Match FIFO
RX rate match FIFO mode: Disabled
RX rate match insert/delete -ve pattern (hex): 0x00000000
RX rate match insert/delete +ve pattern (hex): 0x00000000
☐ Enable rx_std_rmfifo_full port
☐ Enable rx_std_rmfifo_empty port
PCI Express Gen 3 rate match FIFO mode: Bypass

Word Aligner and Bitslip
☐ Enable TX bitslip
☐ Enable tx_std_bitslipboundarysel port
RX word aligner mode: synchronous state machine
RX word aligner pattern length: 10
RX word aligner pattern (hex): 0x000000000000017c
Number of word alignment patterns to achieve sync: 3
Number of invalid data words to lose sync: 3
Number of valid data words to decrement error count: 3
☐ Enable fast sync status reporting for deterministic latency SM

This implements the required PLLs, reset, and supporting IPs to complete the transceiver design.

5.3.2.6.7. ModelSim Byte Ordering Analysis

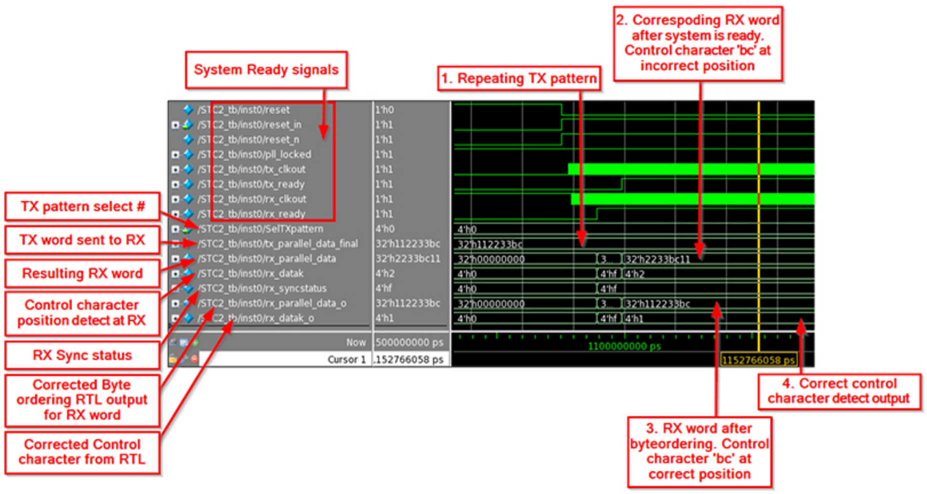
This analysis is performed on ModelSim.

The system is brought out of reset and a TX pattern is driven to the RX channel. Initially, RX is streaming data before the system is ready. Once the system is ready, the resulting RX word has the control character at the incorrect position. The output of the byte ordering RTL confirms that the control character is realigned to the correct LSB location.

Send Feedback

For the purposes of experimentation, the TX pattern is changed without reset. This is an attempt to emulate any inconsistencies experienced by the RX channel due to SI/PI, thermal, or PDN issues. In this example, the RX data binds the control character to the expected LSB even after the output of the byte ordering block.

Figure 220. ModelSim Byte Ordering Analysis



```
/STC2_tb/inst0/reset                    1'h0
/STC2_tb/inst0/reset_in                 1'h1
/STC2_tb/inst0/reset_n                  1'h1
/STC2_tb/inst0/pll_locked               1'h1
/STC2_tb/inst0/tx_clkout                1'h1
/STC2_tb/inst0/tx_ready                 1'h1
/STC2_tb/inst0/rx_clkout                1'h1
/STC2_tb/inst0/rx_ready                 1'h1
/STC2_tb/inst0/SelTXpattern             4'h0
/STC2_tb/inst0/tx_parallel_data_final   32'h112233bc
/STC2_tb/inst0/rx_parallel_data         32'h2233bc11
/STC2_tb/inst0/rx_datak                 4'h2
/STC2_tb/inst0/rx_syncstatus            4'hf
/STC2_tb/inst0/rx_parallel_data_o       32'h112233bc
/STC2_tb/inst0/rx_datak_o               4'h1
                                   Now  500000000 ps
                              Cursor 1  152766058 ps
```

System Ready signals
1. Repeating TX pattern
2. Correspoding RX word after system is ready. Control character 'bc' at incorrect position
TX pattern select #
TX word sent to RX
Resulting RX word
Control character position detect at RX
RX Sync status
Corrected Byte ordering RTL output for RX word
Corrected Control character from RTL
4'h0
32'h112233bc
32'h00000000
32'h2233bc11
4'hf
4'h2
32'h112233bc
4'h1
1100000000 ps
1152766058 ps
4. Correct control character detect output
3. RX word after byteordering. Control character 'bc' at correct position

Related Information

Byte ordering correction code

5.3.2.7. RX PCS FIFO

The RX PCS FIFO operates in the following modes:

- Phase Compensation Mode
- Register Mode

Related Information

RX PCS FIFO on page 362

5.3.2.8. RX Core FIFO

The RX Core FIFO operates in the following modes:

- Phase Compensation Mode
- Register Mode
- Basic Mode

Related Information

RX Core FIFO on page 363

5.4. Intel Stratix 10 PCI Express Gen3 PCS Architecture

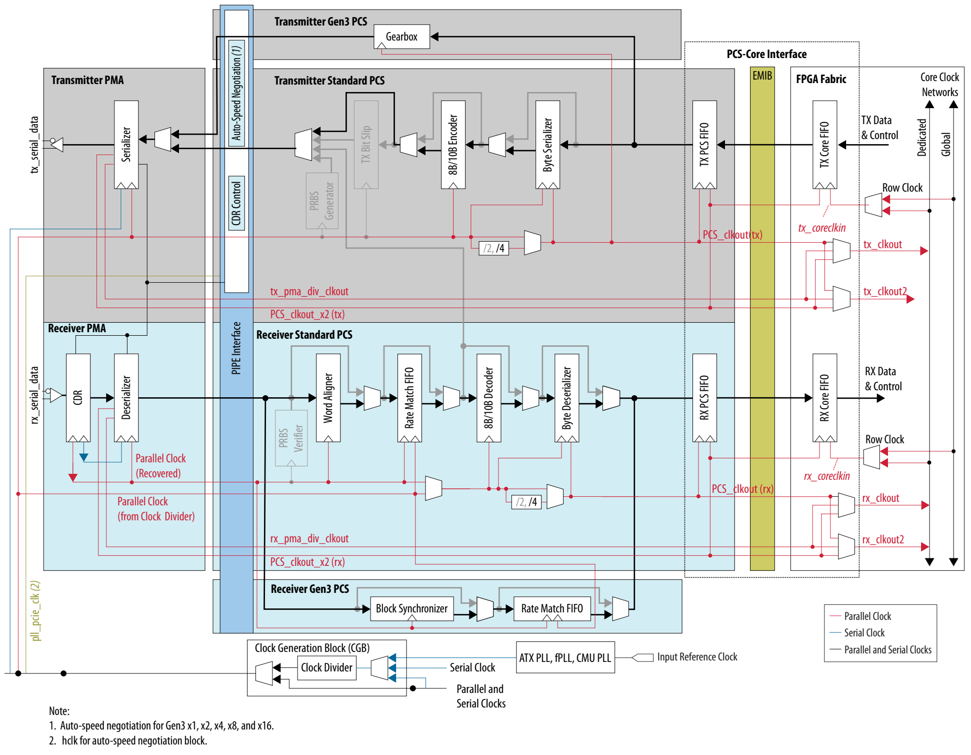
Intel Stratix 10 architecture supports the PCIe Gen3 specification. Intel provides two options to implement the PCI Express solution:

- You can use the Intel Hard IP solution. This complete package provides both the MAC layer and the physical (PHY) layer functionality.
- You can implement the MAC in the FPGA core and connect this MAC to the transceiver PHY through the PIPE interface.

This section focuses on the basic blocks of PIPE 3.0-based Gen3 PCS architecture. The PIPE 3.0-based Gen3 PCS uses a 128b/130b block encoding/decoding scheme, which is different from the 8B/10B scheme used in Gen1 and Gen2 present in the Standard PCS. The 130-bit block contains a 2-bit sync header and a 128-bit data payload. For this reason, Intel Stratix 10 devices include a separate Gen3 PCS that supports functionality at Gen3 speeds. This PIPE interface supports the seamless switching of Data and Clock between the Gen1, Gen2, and Gen3 data rates, and provides support for PIPE 3.0 features. The PCIe Gen3 PCS supports the PIPE interface with the Hard IP enabled, as well as with the Hard IP bypassed.

For more information about the blocks used for Gen1 and Gen2 data rates, see the Transmitter Datapath and Receiver Datapath sections of the *Standard PCS Architecture* chapter.

Figure 221. Gen3 PCS Block Diagram



Note:
1. Auto-speed negotiation for Gen3 x1, x2, x4, x8, and x16.
2. hclk for auto-speed negotiation block.

Related Information

- PCI Express (PIPE) on page 164

- Intel Stratix 10 Standard PCS Architecture on page 366
- Transmitter Datapath on page 367
- Receiver Datapath on page 372

5.4.1. Transmitter Datapath

This section describes the TX FIFO and the gearbox of the Gen3 PCS transmitter.

5.4.1.1. TX Core FIFO

The TX Core FIFO operates in the following modes:

- Phase Compensation Mode
- Register Mode
- Basic Mode

For more information, refer to the *TX Core FIFO* section.

Related Information

TX Core FIFO on page 353

5.4.1.2. TX PCS FIFO

The TX PCS FIFO operates in Phase Compensation Mode.

The TX PCS FIFO interfaces between the transmitter PCS and across EMIB to the TX Core FIFO, and ensures reliable transfer of data and status signals. It compensates for the phase difference between the PCS_`clkout_2x`(tx) and PCS_`clkout`(tx).

Related Information

TX PCS FIFO on page 354

5.4.1.3. Gearbox

The PCIe 3.0 base specification specifies a block size of 130 bits, with the exception of the SKP Ordered Sets, which can be of variable length. An implementation of a 130-bit data path takes significant resources, so the PCIe Gen3 PCS data path is implemented as 32-bits wide. Because the TX PMA data width is fixed to 32 bits, and the block size is 130 bits with variations, a gearbox is needed to convert 130 bits to 32 bits.

The gearbox block in the TX PCS converts the 130-bit data (`tx_parallel_data[127:0]` + `pipe_tx_sync_hdr[1:0]`) to 32-bit data required by the TX PMA as the datapath implementation is 32 bits to reduce usage of resources. The 130-bit data is received as follows in the 32-bit datapath: 34 (32 + 2-bit sync header), 32, 32, 32. During the first cycle the gearbox converts the 34-bit input data to 32-bit data. During the next 3 clock cycles the gearbox merges bits from adjacent cycles to form the 32-bit data. In order for the gearbox to work correctly, a gap must be provided in the data for every 16 shifts as each shift is 2 bits for converting the initial 34-bit to 32-bit in the gearbox. After 16 shifts the gearbox has an extra 32-bit data that was transmitted out, and thus a gap is required in the input data stream. This gap is achieved by driving `pipe_tx_data_valid` low for one cycle after every 16 blocks of input data`(tx_parallel_data)`.

5.4.2. Receiver Datapath

This section describes the blocks used in the receiver datapath for the Gen3 data rate from the Gen3 PCS through the PCS-Core Interface.

5.4.2.1. Block Synchronizer

PMA parallelization occurs at arbitrary word boundaries. Consequently, the parallel data from the RX PMA CDR must be realigned to meaningful character boundaries. The PCI-Express 3.0 base specification outlines that the data is formed using 130-bit blocks, with the exception of SKP blocks.

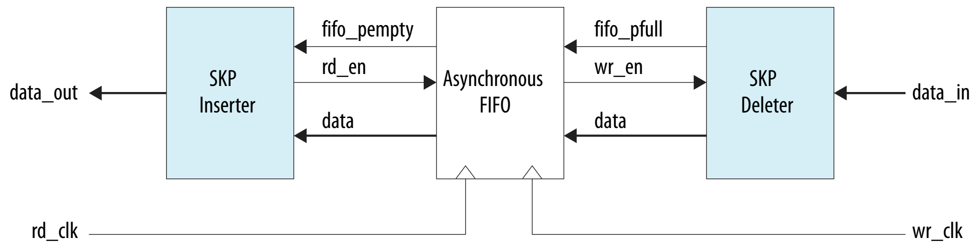
The SKP Ordered Set can be 66, 98, 130, 162, or 194 bits long. The block synchronizer searches for the Electrical Idle Exit Sequence Ordered Set (or the last number of fast training sequences (NFTS) Ordered Set) or skip (SKP) Ordered Set to identify the correct boundary for the incoming stream and to achieve the block alignment. The block is realigned to the new block boundary following the receipt of a SKP Ordered Set, as it can be of variable length.

5.4.2.2. Rate Match FIFO

In asynchronous systems, the upstream transmitter and local receiver can be clocked with independent reference clocks. Frequency differences in the order of a few hundred PPM can corrupt the data when latching from the recovered clock domain to the local receiver reference clock domain. The rate match FIFO compensates for small clock frequency differences between these two clock domains by inserting or removing SKP symbols in the data stream to keep the FIFO from going empty or full respectively.

The PCI-Express 3.0 base specification defines that the SKP Ordered Set (OS) can be 66, 98, 130, 162, or 194 bits long. The SKP OS has the following fixed bits: 2-bit Sync, 8-bit SKP END, and a 24-bit LFSR = 34 Bits. The Rate Match/Clock compensation block adds or deletes the 4 SKP characters (32-bit) to keep the FIFO from going empty or full, respectively. If the FIFO is nearly full, it deletes the 4 SKP characters (32-bit) by disabling write whenever a SKP is found. If the FIFO is nearly empty, the design waits for a SKP Ordered Set to start and then stops reading the data from the FIFO, and inserts a SKP in the outgoing data. The actual FIFO core (memory element) is in the Shared Memory block in the PCS channel.

Figure 222. Rate Match FIFO



5.4.2.3. RX PCS FIFO

The RX PCS FIFO operates in Phase Compensation Mode.

For more information, refer to the RX PCS FIFO section.

Related Information

RX PCS FIFO on page 362

5.4.2.4. RX Core FIFO

The RX Core FIFO operates in Phase Compensation Mode.

For more information, refer to the RX Core FIFO section.

Related Information

RX Core FIFO on page 363

5.4.3. PIPE Interface

This section describes the Auto-Speed Negotiation and the Clock Data Recovery Control of the PIPE interface.

5.4.3.1. Auto-Speed Negotiation

Auto-speed negotiation controls the operating speed of the transceiver when operating under PIPE 3.0 modes. By monitoring the `pipe_rate` signal from the PHY-MAC, this feature changes the transceiver operation modes to PIPE Gen1, PIPE Gen2, or PIPE Gen3.

Related Information

Rate Switch on page 173

5.4.3.2. Clock Data Recovery Control

The CDR control feature is used for the L0s fast exit when operating in PIPE Gen3 mode. Upon detecting an Electrical Idle Ordered Set (EIOS), this feature takes manual control of the CDR by forcing it into a lock-to-reference mode. When an exit from electrical idle is detected, this feature moves the CDR into lock-to-data mode to achieve fast data lock.

5.5. PCS Support for GXT Channels

GXT channels use the enhanced PCS TX/RX gearbox and FIFOs.

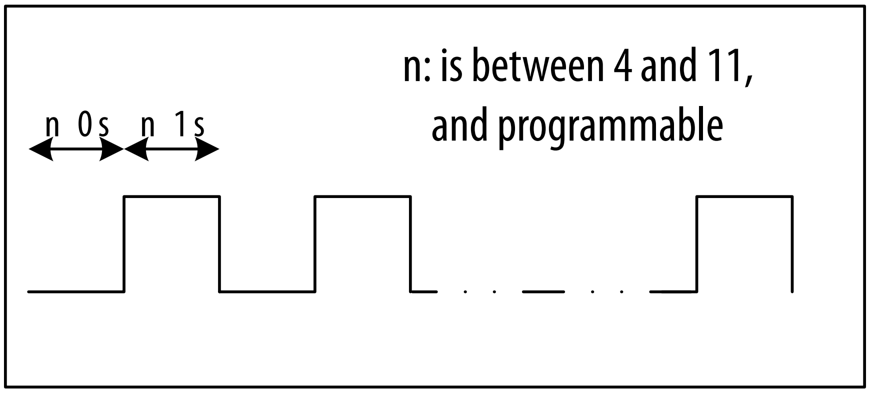
Related Information

Debug Functions on page 154

5.6. Square Wave Generator

The square wave generator has programmable n-number of consecutive serial bit 1s and 0s (where n = 1, 4, 6, or 8).

Figure 223. Square Wave Generator



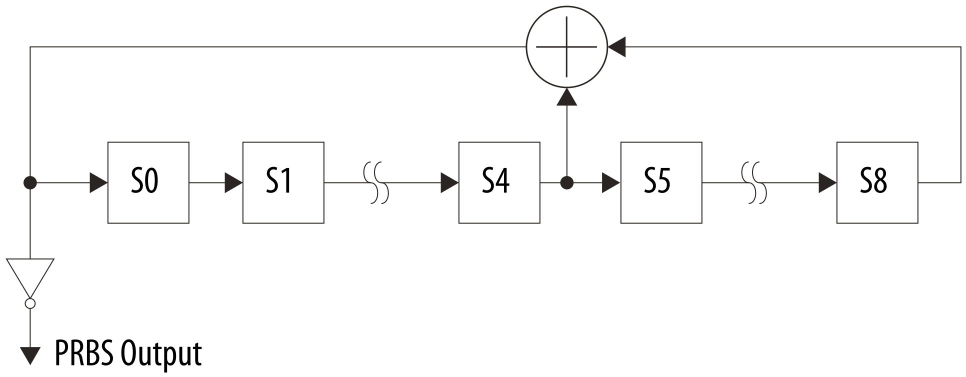
5.7. PRBS Pattern Generator

Intel Stratix 10 transceivers contain hardened Pseudo Random Binary Sequence (PRBS) generators and verifiers to provide a simple and easy way to verify and characterize high speed links.

The PRBS Pattern Generator supports 5 modes:

- PRBS7
- PRBS9
- PRBS15
- PRBS23
- PRBS31

Figure 224. PRBS Generator for Serial Implementation of PRBS9 Pattern



Note: All supported PRBS generators are similar to the PRBS9 generator.

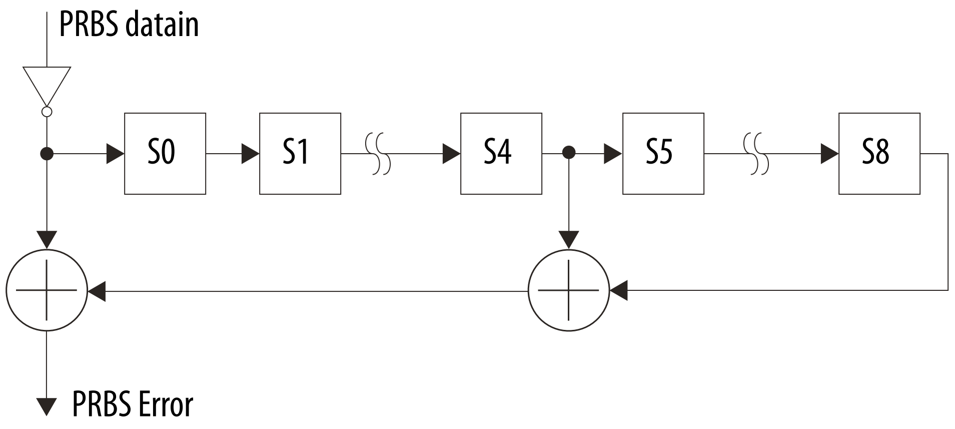
5.8. PRBS Pattern Verifier

You can use the PRBS pattern verifier to easily characterize high-speed links.

The PRBS Pattern Verifier supports 5 modes:

- PRBS7
- PRBS9
- PRBS15
- PRBS23
- PRBS31

Figure 225. PRBS9 Verify Serial Implementation



The PRBS verifier has the following control and status signals available to the FPGA fabric:

- `rx_prbs_done`—Indicates the PRBS sequence has completed one full cycle. It stays high until you reset it with `rx_prbs_err_clr`.
- `rx_prbs_err`—Goes high if an error occurs. This signal is pulse-extended to allow you to capture it in the RX FPGA CLK domain.
- `rx_prbs_err_clr`—Used to reset the `rx_prbs_err` signal.

Enable the PRBS verifier control and status ports through the Native PHY IP core.

Related Information

Debug Functions on page 154

5.9. Loopback Modes

The PMA supports three serial loopback modes:

- Serial loopback
- Pre-CDR reverse serial loopback
- Post-CDR reverse serial loopback

Note: Intel Stratix 10 transceiver channels configured in GXT mode do not support Pre-CDR reverse serial loopback and Post-CDR reverse serial loopback.

Figure 226. Serial Loopback Mode

The serial loopback path sets the CDR to recover data from the serializer instead of the receiver serial input pin. The transmitter buffer sends data normally, but serial loopback takes the data before the buffer. Note that this also skips the receiver CTLE, but passes through the VGA before going into the CDR. The VGA can be adjusted accordingly.

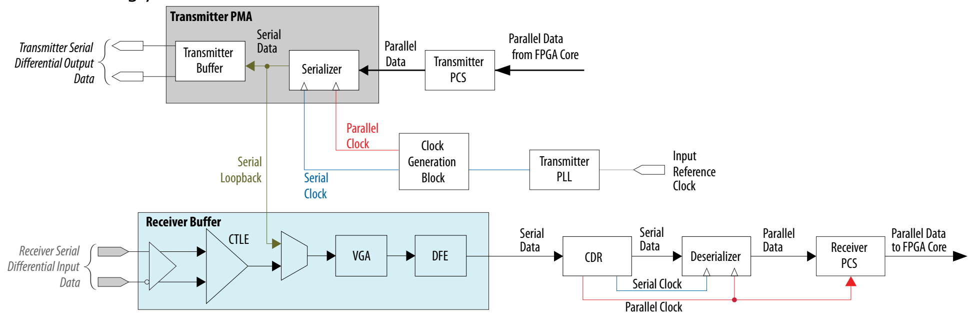


Figure 227. Pre-CDR Reverse Serial Loopback Mode

Note: TX pre-emphasis is not supported in pre-CDR loopback. Intel recommends setting TX pre-emphasis to 0 for all taps.

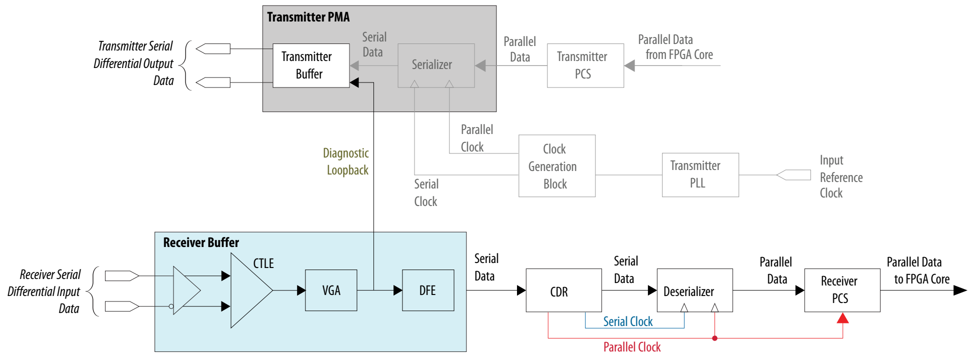
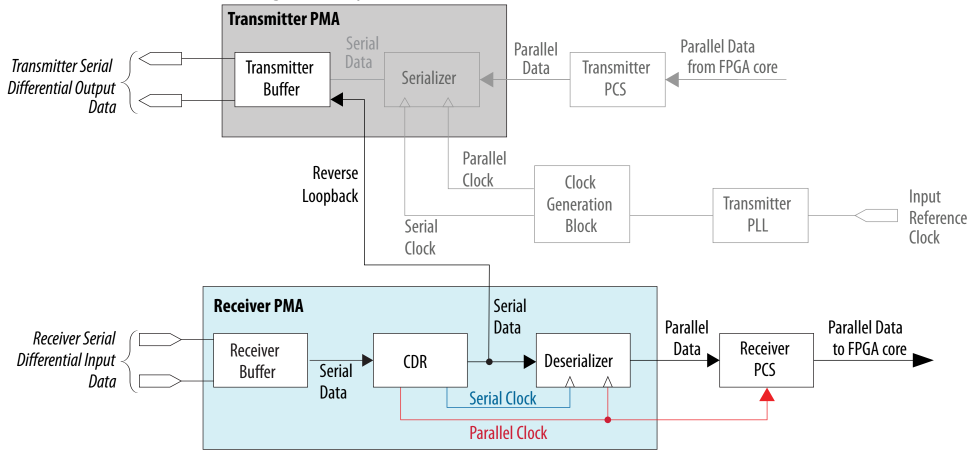


Figure 228. Post-CDR Reverse Serial Loopback Mode

The reverse loopback path sets the transmitter buffer to transmit data fed directly from the CDR recovered data. Data from the serializer is ignored by the transmitter buffer.



Related Information

Debug Functions on page 154

5.10. Intel Stratix 10 L-Tile/H-Tile Transceiver PHY Architecture Revision History

Document Version	Changes
2020.03.03	Made the following changes: • Updated the following figures to make it clear that `rx_clkout` is driven by CDR. — Enhanced PCS Datapath Diagram — Standard PCS Datapath Diagram — Gen3 PCS Block Diagram • Clarified ODI support for L-Tile.
2019.03.22	Made the following change: • Updated the "Serial Loopback Mode," "Pre-CDR Reverse Serial Loopback Mode," and "Post-CDR Reverse Serial Loopback Mode" figures.
2018.07.06	Made the following changes: • Added a note to the "Loopback Modes" section. • Clarified the data rate for GX channels in the "Enhanced PCS Architecture" section. • Added a note describing ODI support in L-Tile devices in the "On-die Instrumentation" section.
2018.03.16	Made the following changes: • Changed the number of programmable taps in the "Transmitter Buffer" section. • Changed the rate match FIFO modes in the "Rate Match FIFO" section. • Added the "Byte Ordering Register-Transfer Level (RTL)" section. • Added the "Byte Serializer Effects on Data Propagation at the RX Side" section. • Added the "ModelSim Byte Ordering Analysis" section.
2017.08.11	Made the following changes: • Added note that ODI is only supported for H-Tile in the "Receiver Buffer" and "On-Die Instrumentation" sections.
2017.06.06	Made the following changes: • Remove the QPI configuration from the "Transmitter Buffer" section. • Added the "PRBS Generator and PRBS Verifier" section. • Removed the "PRBS Generator" section from the "Standard PCS Architecture" section. • Removed the "PRBS Verifier" section from the "Standard PCS Architecture" section. • Added clarification about receiver termination in the "Programmable Differential On-Chip Termination (OCT)" section. • Added a link to the *Stratix 10 H-Tile Pre-emphasis and Output Swing Estimator* in the "Programmable Pre-Emphasis" section. • Updated the "Receiver Buffer" figure. • Added the "Effect of AC Gain on the Frequency Response" figure. • Added the "Effect of EQ Gain on the Frequency Response" figure. • Changed "66 Bit Blocks" to "67 Bit Blocks" in the "Interlaken Frame Generator" figure. • Added the "Square-wave Generator" section.
2017.02.17	Made the following changes: • Added the **Enable tx_polinv port** option to the "Polarity Inversion Feature" section.
2016.12.21	Initial release.

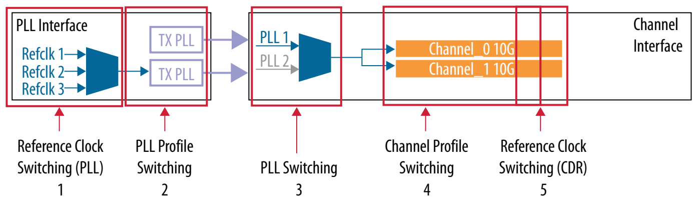
6. Reconfiguration Interface and Dynamic Reconfiguration

This chapter explains the purpose and the use of the Intel Stratix 10 reconfiguration interface that is part of the Transceiver Native PHY IP core and the Transceiver PLL IP cores.

Dynamic reconfiguration is the process of modifying transceiver channels and PLLs to meet changing requirements during device operation. You can customize channels and PLLs by triggering reconfiguration during device operation or following power-up. Dynamic reconfiguration is available for Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY, fPLL, ATX PLL, and CMU PLL IP cores.

Note: In Intel Stratix 10, the Embedded Multi-die Interconnect Bridge (EMIB) must also be reconfigured in addition to channels and PLLs using the reconfiguration interface.

Figure 229. Reconfigurable Interfaces



Use the reconfiguration interface to dynamically change the transceiver channel or PLL settings, EMIB settings for the following applications:

- Fine tuning signal integrity by adjusting TX and RX analog settings
- Enabling or disabling transceiver channel blocks, such as the PRBS generator and the verifier
- Changing data rates to perform auto negotiation in CPRI, SATA, or SAS applications
- Changing data rates in Ethernet (1G/10G) applications by switching between standard and enhanced PCS datapaths
- Changing TX PLL settings for multi-data rate support protocols such as CPRI
- Changing RX CDR settings from one data rate to another
- Switching between multiple TX PLLs for multi-data rate support

The Native PHY and Transmit PLL IP cores provide the following features that allow dynamic reconfiguration:

- Reconfiguration interface
- Configuration files
- Multiple reconfiguration profiles

- Embedded reconfiguration streamer
- Native PHY Debug Master Endpoint (NPDME)
- Optional reconfiguration logic

6.1. Reconfiguring Channel and PLL Blocks

The following table lists some of the available dynamic reconfiguration features in Intel Stratix 10 devices.

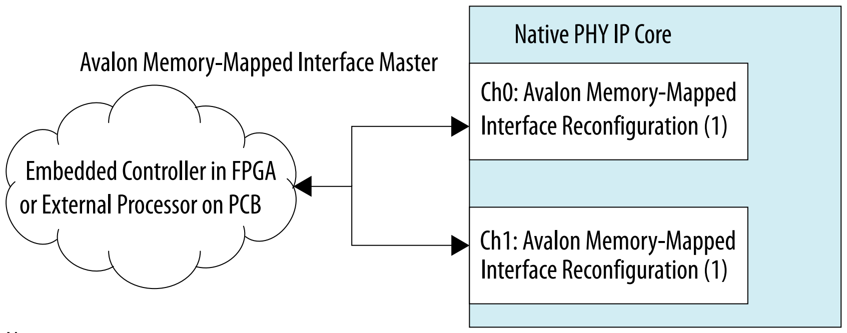
Table 157. Intel Stratix 10 Dynamic Reconfiguration Feature Support

Reconfiguration	Features
Channel Reconfiguration	PMA analog features • V_{OD} • Pre-emphasis • Continuous Time Linear Equalizer (CTLE) • Decision Feedback Equalization (DFE) • Variable Gain Amplifier (VGA)
	TX PLL • TX local clock dividers • TX PLL switching
	RX CDR • RX CDR settings • RX CDR reference clock switching
	Reconfiguration of PCS blocks within the datapath
	Datapath switching • Standard, Enhanced, PCS Direct
PLL Reconfiguration	PLL settings • Counters
	PLL reference clock switching

6.2. Interacting with the Reconfiguration Interface

Each transceiver channel and PLL contains Avalon memory-mapped interface reconfiguration.The reconfiguration interface on the channel is shared between the channel's PCS, PMA and Embedded Multi-die Interconnect Bridge (EMIB). The reconfiguration interface provides direct access to the programmable space of each channel and PLL. Communication with the channel and PLL reconfiguration interface requires an Avalon memory-mapped interface master. You can start dynamic reconfiguration sequence of each channel and PLL concurrently or sequentially, depending on how the Avalon memory-mapped interface master is connected to Avalon memory-mapped interface reconfiguration. However, you must check for the internal configuration bus arbitration before performing reconfiguration. Refer to *Arbitration* for more details about requesting access to and returning control of the internal configuration bus from PreSICE.

Figure 230. Reconfiguration Interface in Intel Stratix 10 Transceiver IP Cores


Native PHY IP Core
Avalon Memory-Mapped Interface Master
Ch0: Avalon Memory-Mapped Interface Reconfiguration (1)
Embedded Controller in FPGA or External Processor on PCB
Ch1: Avalon Memory-Mapped Interface Reconfiguration (1)

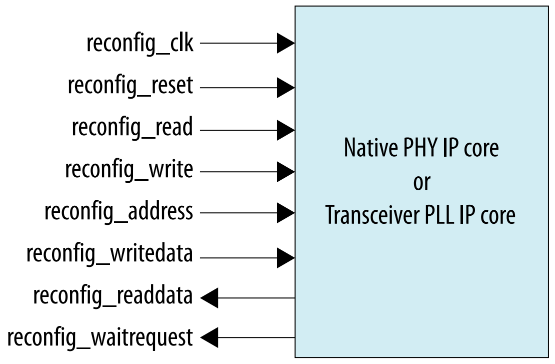
Note:

1. The Native PHY IP core, user reconfiguration logic (Avalon memory-mapped interface master), interfaces with the hard registers and EMIB using the Avalon memory-mapped interface reconfiguration.

A transmit PLL instance has a maximum of one reconfiguration interface. Unlike PLL instances, a Native PHY IP core instance can specify multiple channels. You can use a dedicated reconfiguration interface for each channel or share a single reconfiguration interface across multiple channels to perform dynamic reconfiguration.

Avalon memory-mapped interface masters interact with the reconfiguration interface by performing Avalon memory-mapped interface read and write operations to initiate dynamic reconfiguration of specific transceiver parameters. All read and write operations must comply with Avalon memory-mapped interface specifications.

Figure 231. Top-Level Signals of the Reconfiguration Interface



User-accessible Avalon memory-mapped interface reconfiguration and PreSICE Avalon memory-mapped interface share a single internal configuration bus. This bus is arbitrated to get access to the Avalon memory-mapped interface of the channel or PLL. Refer to the "Arbitration" section for more details about requesting access to and returning control of the internal configuration bus from PreSICE.

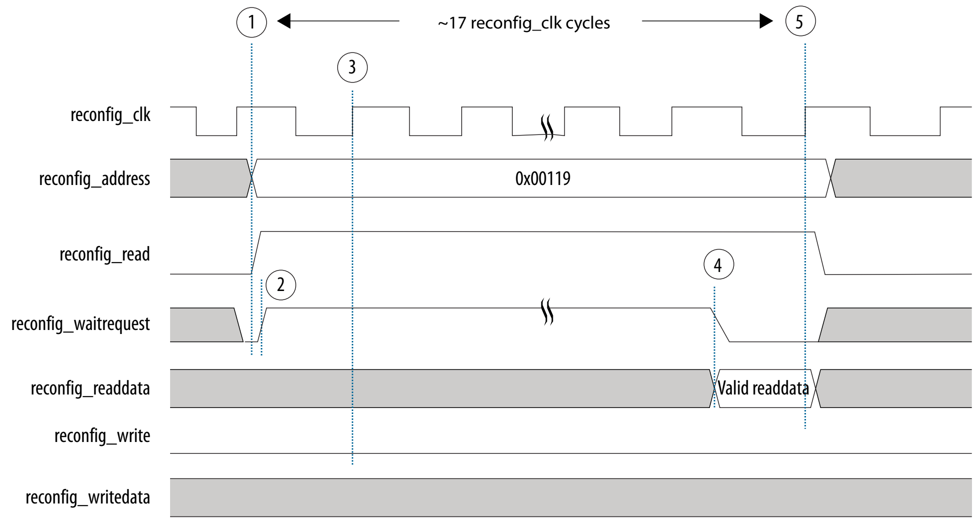
Related Information

- Arbitration on page 403
- *Avalon Interface Specifications*

6.2.1. Reading from the Reconfiguration Interface

Reading from the reconfiguration interface of the Transceiver Native PHY IP core or Transceiver PLL IP core retrieves the current value at a specific address.

Figure 232. Reading from the Reconfiguration Interface



1. The master asserts reconfig_address and reconfig_read after the rising edge of reconfig_clk.
2. The slave asserts reconfig_waitrequest, stalling the transfer.
3. The master samples reconfig_waitrequest. Because reconfig_waitrequest is asserted, the cycle becomes a wait state and reconfig_address, reconfig_read, and reconfig_write remain constant.
4. The slave presents valid reconfig_readdata and deasserts reconfig_waitrequest.
5. The master samples reconfig_waitrequest and reconfig_readdata, completing the transfer.

After the `reconfig_read` signal is asserted, the `reconfig_waitrequest` signal asserts for a few `reconfig_clock` cycles, then deasserts. This deassertion indicates the `reconfig_readdata` bus contains valid data.

Note: You must check for the internal configuration bus arbitration before performing reconfiguration. Refer to the "Arbitration" section for more details about requesting access to and returning control of the internal configuration bus from PreSICE.

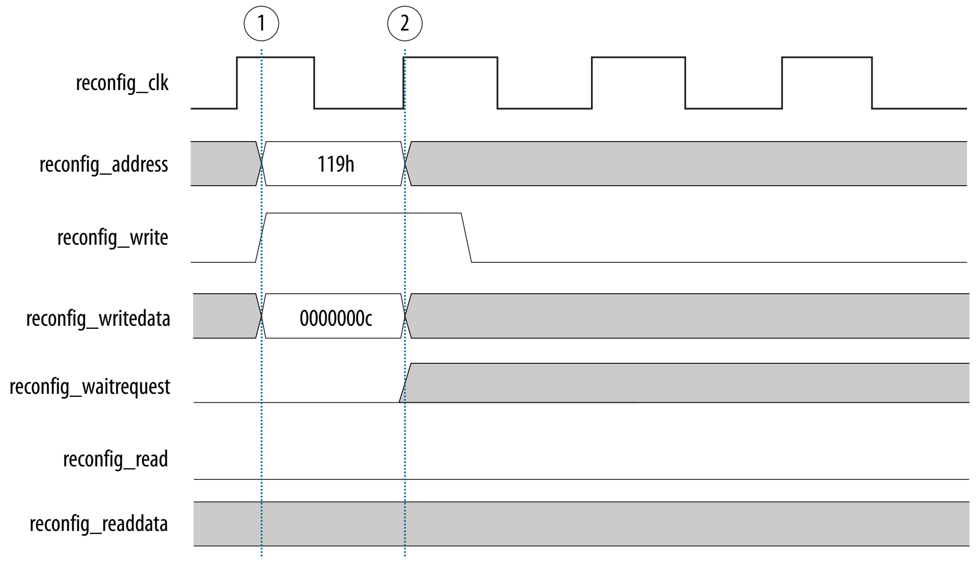
Related Information

Arbitration on page 403

6.2.2. Writing to the Reconfiguration Interface

Writing to the reconfiguration interface of the Transceiver Native PHY IP core or TX PLL IP core changes the data value at a specific address. All writes to the reconfiguration interface must be read-modify-writes, because two or more features may share the same reconfiguration address.

Figure 233. Writing to the Reconfiguration Interface



1. The master asserts the reconfig_address, reconfig_write, and reconfig_writedata signals.
2. The slave (channel or PLL) captures reconfig_writedata, ending the transfer.

Note: You must check for the internal configuration bus arbitration before performing reconfiguration. Refer to the "Arbitration" section for more details about requesting access to and returning control of the internal configuration bus from PreSICE.

Related Information

Arbitration on page 403

6.3. Multiple Reconfiguration Profiles

You should enable multiple configurations or profiles in the same Native PHY, Transmit PLL IP core, or both Parameter Editors for performing dynamic reconfiguration. This allows the IP Parameter Editor to create, store, and analyze the parameter settings for multiple configurations or profiles.

Note: fPLL in **Core** mode does not support the dynamic reconfiguration feature.

When you enable the multiple reconfiguration profiles feature, the Native PHY, Transmit PLL, or both IP cores can generate configuration files for all the profiles in the format desired (SystemVerilog package, MIF, or C header file). The configuration files are located in the `<IP instance name>/reconfig/` subfolder of the IP instance with the configuration profile index added to the filename. For example, the configuration file for Profile 0 is stored as `<filename_CFG0.sv>`. The Intel Quartus Prime Timing Analyzer includes the necessary timing paths for all the configurations based on initial and target profiles. You can also generate reduced configuration files that contain only the attributes that differ between the multiple configured profiles. You can create up to eight reconfiguration profiles (Profile 0 to Profile 7) at a time for each instance of the Native PHY/Transmit PLL IP core.

Note: The addresses and bit settings of EMIB for a chosen configuration are available in the configuration files generated by the Native PHY IP.

The configuration files generated by Native PHY IP also include PMA analog settings specified in the **Analog PMA settings** tab of the Native PHY IP Parameter Editor. The analog settings selected in the Native PHY IP Parameter Editor are used to include these settings and their dependent settings in the selected configuration files.

Refer to "Steps to Perform Dynamic Reconfiguration" for a complete list of steps to perform dynamic reconfiguration using the IP guided reconfiguration flow with multiple reconfiguration profiles enabled.

To perform a PMA reconfiguration such as TX PLL switching, CGB divider switching, or reference clock switching, you must use the flow described in "Steps to Perform Dynamic Reconfiguration".

You can use the multiple reconfiguration profiles feature without using the embedded reconfiguration streamer feature. When using the multiple reconfiguration profiles feature by itself, you must write the user logic to reconfigure all the entries that are different between the profiles while moving from one profile to another.

Note: You must ensure that none of the profiles in the Native PHY IP and Transmit PLL IP Core Parameter Editor gives error messages, or the IP generation fails. The Native PHY IP core and Transmit PLL IP core only validates the current active profile dynamically. For example, if you store a profile with error messages in the Native PHY IP or Transmit PLL IP Core Parameter Editor and load another profile without any error messages, the error messages disappear in the IP. You are allowed to generate the IP, but the generation fails. For timing closure for each profile, please refer to "Timing Closure Recommendations" section.

Related Information

- Steps to Perform Dynamic Reconfiguration on page 405
- Timing Closure Recommendations on page 428

6.3.1. Configuration Files

The Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY and Transmit PLL IP cores optionally allow you to save the parameters you specify for the IP instances as configuration files. The configuration file stores addresses and data values for that specific IP instance.

The configuration files are generated during IP generation. They are located in the `<IP instance name>/reconfig/` subfolder of the IP instance. The configuration data is available in the following formats:

- **SystemVerilog packages**: <name>**.sv**
- **C Header files**: <name>**.h**
- **Memory Initialization File (MIF)**: <name>**.mif**

Select one or more of the configuration file formats on the **Dynamic Reconfiguration** tab of the Transceiver Native PHY or Transmit PLL parameter editor to store the configuration data. All configuration file formats generated for a particular IP instance contain the same address and data values. The contents of the configuration files can be used to reconfigure from one transceiver PLL configuration to another.

Note: The addresses and bit settings of EMIB for a chosen configuration are available in the configuration files generated by the Native PHY IP.

The configuration files generated by Native PHY IP also include the PMA analog settings specified in the **Analog PMA settings** tab of the Native PHY IP Parameter Editor. The analog settings selected in the Native PHY IP Parameter Editor are used to include these settings and their dependent settings in the selected configuration files.

Example 1. SystemVerilog Configuration File

```
    27'h008FF04,
// [26:16]-DPRIO address=0x008;
// [15:8]-bit mask=0xFF;
// [7:7]- hssi_tx_pcs_pma_interface_pldif_datawidth_mode=pldif_data_10bit(1'h0);
// [6:5]-hssi_tx_pcs_pma_interface_tx_pma_data_sel=ten_g_pcs(2'h0);
// [4:4]-hssi_tx_pcs_pma_interface_prbs_gen_pat=prbs_gen_dis(1'h0);
// [3:0]-hssi_tx_pcs_pma_interface_sq_wave_num=sq_wave_default(4'h4);
…

localparam HSSI_TX_PCS_PMA_INTERFACE_PLDIF_DATAWIDTH_MODE_VALUE = "pldif_data_10bit";
localparam HSSI_TX_PCS_PMA_INTERFACE_PLDIF_DATAWIDTH_MODE_ADDR_OFST = 8;
localparam HSSI_TX_PCS_PMA_INTERFACE_PLDIF_DATAWIDTH_MODE_ADDR_FIELD_OFST = 7;
localparam HSSI_TX_PCS_PMA_INTERFACE_PLDIF_DATAWIDTH_MODE_ADDR_FIELD_HIGH = 7;
localparam HSSI_TX_PCS_PMA_INTERFACE_PLDIF_DATAWIDTH_MODE_ADDR_FIELD_SIZE = 1;
localparam HSSI_TX_PCS_PMA_INTERFACE_PLDIF_DATAWIDTH_MODE_ADDR_FIELD_BITMASK =
    32'h00000080;
localparam HSSI_TX_PCS_PMA_INTERFACE_PLDIF_DATAWIDTH_MODE_ADDR_FIELD_VALMASK =
    32'h00000000;
localparam HSSI_TX_PCS_PMA_INTERFACE_PLDIF_DATAWIDTH_MODE_ADDR_FIELD_VALUE = 1'h0;
```

Note: DPRIO refers to Avalon memory-mapped interface registers.

The SystemVerilog configuration files contain two parts. The first part consists of a data array of 27-bit hexadecimal values. The second part consists of parameter values. For the data array, each 27-bit hexadecimal value is associated with a comment that describes the various bit positions.

Table 158. Mapping of SystemVerilog Configuration File Line

Bit Position	Description
[26:16]	The channel or PLL address.
[15:8]	The channel or PLL bit mask. The bit mask exposes the bits that are configured in either the Transceiver Native PHY or the transmit PLL IP cores.
[7:0]	Feature bit values.

For example, a value of 27'h008FF04 represents an address of 0x008 and a bit mask of 0xFF. The four features that reside at address 0x008 are:

- `hssi_tx_pcs_pma_interface_pldif_datawidth_mode` with a value of 1'h0
- `hssi_tx_pcs_pma_interface_tx_pma_data_sel` with a value of 2'h0
- `hssi_tx_pcs_pma_interface_prbs_gen_pat` with a value of 1'h0
- `hssi_tx_pcs_pma_interface_sq_wave_num` with a value of 4'h4

Writing to bit 7 of address 0x008 changes the `hssi_tx_pcs_pma_interface_pldif_datawidth_mode` feature.

The MIF file and C header file are set up similarly to the SystemVerilog package file. Multiple transceiver features may reside at the same address. Also, a single transceiver feature may span across multiple addresses.

You can generate multiple configurations (up to 8) of the transceiver Native PHY IP Core, PLL IP Core, or both. One configuration defines the base transceiver or PLL configuration and the other configurations define the modified or target configurations. Use the IP Parameter Editor to create base and modified configurations of the Transceiver Native PHY or PLL IP core, according to the following table.

Table 159. Transceiver Native PHY or PLL IP Parameters (Base and Modified Configurations)

Native PHY or PLL Instance	Required Parameter Settings	Saved In
Base Configuration	• Click **Interface Protocols ➤ Transceiver PHY ➤ Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY** for the Native PHY IP core. Or, select one of the supported transmit PLL IP cores under **PLL**. Enable all options required for the base configuration, such as data rate, PCS options, and PMA options. • Enable all ports to be used by the modified configuration. For example, if the bitslip feature is not required in the base configuration, but required in modified configuration, then you must enable the `tx_std_bitslipboundarysel` port. Reconfiguring between Standard PCS, Enhanced PCS, and PCS Direct requires that you turn on **Enable datapath and interface reconfiguration**. The **Transceiver configuration rules** define the initial mode of the PHY instance. • On the **Dynamic Reconfiguration** tab, turn on **Enable dynamic reconfiguration** and specify the **Configuration Options**. This flow requires that you turn on **Configuration file** option.	• *<Native PHY Base Instance Name>/* `reconfig/altera_xcvr_native_s10_reconfig_parameters.sv` contains all transceiver register addresses and their bit value for that transceiver configuration. Or • *<PLL Base Instance Name>/* `reconfig/altera_xcvr_<type>_pll_s10_reconfig_parameters.sv` contains all PLL register addresses and their bit value for that PLL configuration.
Modified Configuration	• Click **Interface Protocols ➤ Transceiver PHY ➤ Intel Stratix 10 L-Tile/H-Tile Transceiver Native PHY**. Or, select one of the supported transmit PLL IP cores under **PLL**. Enable all options required for the modified configuration, such as data rate, PCS options, and PMA options. • Enable all ports that are used by the modified configuration. Reconfiguring between Standard PCS, Enhanced PCS, and PCS Direct requires **Enable datapath and interface reconfiguration** be enabled. The **Transceiver configuration rules** define the mode of the PHY instance. • On the **Dynamic Reconfiguration** tab, turn on **Enable dynamic reconfiguration** and specify the same **Configuration Options** as the base instance.	• *<Native PHY Modified Instance Name>/* `reconfig/altera_xcvr_native_s10_reconfig_parameters.sv` contains all transceiver register addresses and their bit value for that transceiver configuration. Or • *<PLL Modified Instance Name>/* `reconfig/altera_xcvr_<type>_pll_s10_reconfig_parameters.sv` contains all PLL register addresses and their bit value for that PLL configuration.

Note: You can generate the base and modified configuration files in the same or different folders. If you use the same folder, each configuration name must be unique.

Intel recommends following the flow described in the "Steps to Perform Dynamic Reconfiguration" section when performing dynamic reconfiguration of either the Native PHY IP core, transmit PLL IP core, or both.

6.3.2. Embedded Reconfiguration Streamer

You can optionally enable the embedded reconfiguration streamer in the Native PHY, Transmit PLL, or both IP cores to automate the reconfiguration operation. The embedded reconfiguration streamer is a feature block that can perform Avalon memory-mapped interface transactions to access channel/Transmit PLL configuration registers in the transceiver. When you enable the embedded streamer, the Native PHY/ Transmit PLL IP cores embed HDL code for reconfiguration profile storage and reconfiguration control logic in the IP files.

For the Transmit PLL IP, you can initiate the reconfiguration operation by writing to the control registers of the PLL using reconfiguration interface. Control and status signals of the streamer block are memory mapped in the PLL's soft control and status registers.

For the Native PHY IP, you can initiate the reconfiguration operation by writing to the control registers of the channel using reconfiguration interface. Control and status signals of the streamer block are memory mapped in the PHY's soft control and status registers. These embedded reconfiguration control and status registers are replicated for each channel.

Note: You cannot merge reconfiguration interfaces across multiple IP cores when the embedded reconfiguration streamer is enabled. Refer to "Dynamic Reconfiguration Interface Merging Across Multiple IP Blocks" section for more details.

For example, if the Native PHY IP core has four channels—logical channel 0 to logical channel 3—and you want to reconfigure logical channel 3 using the embedded reconfiguration streamer, you must write to the control register of logical channel 3 using the reconfiguration interface with the appropriate bit settings.

Note: The addresses and bit settings of EMIB for a chosen configuration are available in the configuration files generated by the Native PHY IP.

The configuration files generated by Native PHY IP also include the **Analog PMA settings** tab of the Native PHY IP Parameter Editor. The analog settings selected in the Native PHY IP Parameter Editor are used to include these settings and their dependent settings in the selected configuration files.

Refer to "Steps to Perform Dynamic Reconfiguration" for a complete list of steps to perform dynamic reconfiguration using the IP guided reconfiguration flow with embedded streamer enabled. To perform a PMA reconfiguration such as TX PLL switching, CGB divider switching, or reference clock switching, use the reconfiguration flow for special cases described in "Steps to Perform Dynamic Reconfiguration".

Refer to *Logical View of the L-Tile/H-Tile Transceiver Registers* for more details on Embedded reconfiguration streamer registers and bit settings.

Related Information

- Logical View of the L-Tile/H-Tile Transceiver Registers on page 450
- Steps to Perform Dynamic Reconfiguration on page 405
- Dynamic Reconfiguration Interface Merging Across Multiple IP Blocks on page 424

6.4. Arbitration

In Intel Stratix 10 devices, there are two levels of arbitration:

Figure 234. Arbitration in Intel Stratix 10 Transmit PLL

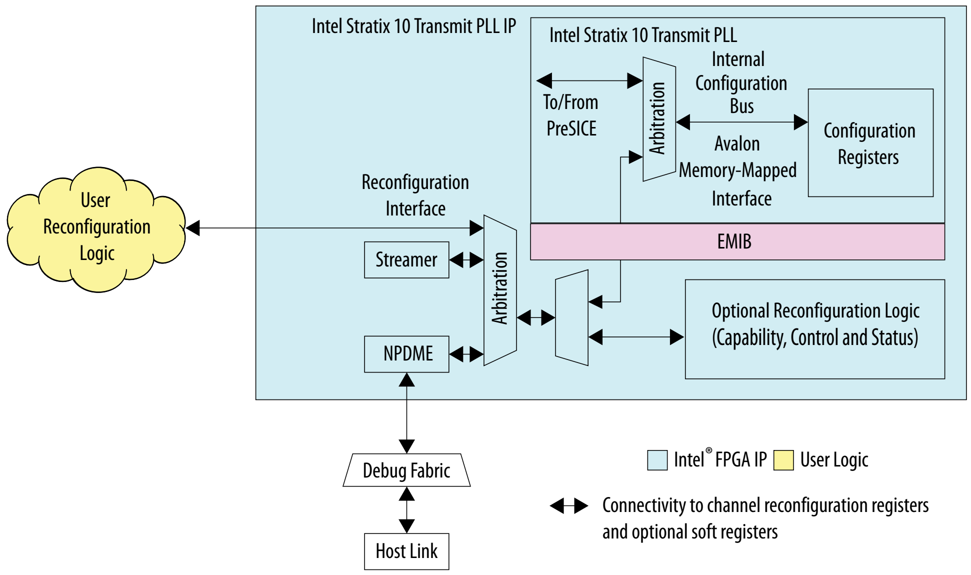
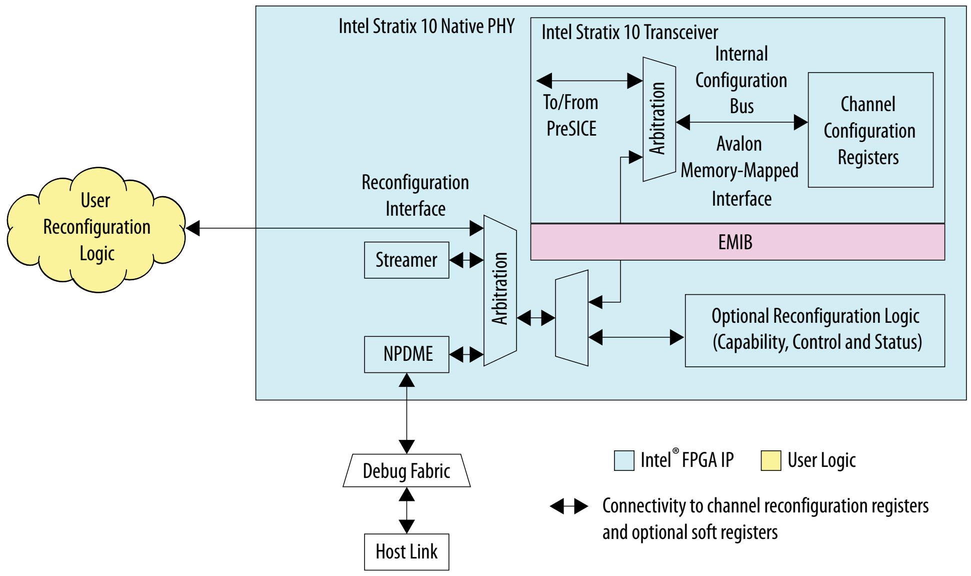

Intel Stratix 10 Transmit PLL IP
Intel Stratix 10 Transmit PLL
Internal Configuration Bus
To/From PreSICE
Arbitration
Configuration Registers
Avalon Memory-Mapped Interface
Reconfiguration Interface
User Reconfiguration Logic
EMIB
Streamer
Arbitration
Optional Reconfiguration Logic (Capability, Control and Status)
NPDME
Debug Fabric
Host Link
Intel® FPGA IP
User Logic
Connectivity to channel reconfiguration registers and optional soft registers

Figure 235. Arbitration in Intel Stratix 10 Native PHY


Intel Stratix 10 Native PHY
Intel Stratix 10 Transceiver
Internal Configuration Bus
To/From PreSICE
Arbitration
Channel Configuration Registers
Avalon Memory-Mapped Interface
Reconfiguration Interface
User Reconfiguration Logic
EMIB
Streamer
Arbitration
Optional Reconfiguration Logic (Capability, Control and Status)
NPDME
Debug Fabric
Host Link
Intel® FPGA IP
User Logic
Connectivity to channel reconfiguration registers and optional soft registers

- Reconfiguration interface arbitration with the PreSICE calibration engine

  When you have control over the internal configuration bus, refer to the second level of arbitration: Arbitration between multiple masters within the Native PHY/PLL IPs.

  For more details about arbitration between the reconfiguration interface and PreSICE, refer to the *Calibration* chapter.

- Arbitration between multiple masters within the Native PHY/PLL IPs

  Below are the feature blocks that can access the programmable registers:

  — Embedded reconfiguration streamer

  — NPDME

  — User reconfiguration logic connected to the reconfiguration interface

  When the internal configuration bus is not owned by the PreSICE, which feature block has access depends on which of them are enabled.

  These feature blocks arbitrate for control over the programmable space of each transceiver channel/PLL. Each of these feature blocks can request access to the programmable registers of a channel/PLL by performing a read or write operation to that channel/PLL. For any of these feature blocks to be used, you must first have control over the internal configuration bus. You must ensure that these feature blocks have completed all the read/write operations before you return the bus access to PreSICE.

  The embedded reconfiguration streamer has the highest priority, followed by the reconfiguration interface, followed by the NPDME. When two feature blocks are trying to access the same transceiver channel on the same clock cycle, the feature block with the highest priority is given access. The only exception is when a lower-priority feature block is in the middle of a read/write operation and a higher-priority feature block tries to access the same channel/PLL. In this case, the higher-priority feature block must wait until the lower-priority feature block finishes the read/write operation.

  Note: When you enable NPDME in your design, you must

  — connect an Avalon memory-mapped interface master to the reconfiguration interface

  — OR connect the `reconfig_clock`,`reconfig_reset` signals and ground the `reconfig_write`, `reconfig_read`, `reconfig_address` and `reconfig_writedata` signals of the reconfiguration interface. If the reconfiguration interface signals are not connected appropriately, there is no clock or reset for the NPDME, and the NPDME does not function as expected.

Related Information

Calibration on page 433

6.5. Recommendations for Dynamic Reconfiguration

Recommendations for Channels

- When reconfiguring PLLs across data rates, Intel recommends that you hold the channel transmitter (analog and digital) associated with the PLL in reset during reconfiguration and recalibration of the PLL.
- When reconfiguring channels across data rates or protocol modes, Intel recommends that you hold the channel transmitter (analog and digital) in reset during reconfiguration and recalibration of the channel transmitter.
- When reconfiguring channels across data rates or protocol modes, Intel recommends that you hold the channel receiver (analog and digital) in reset during reconfiguration and recalibration of the channel receiver.
- When performing reconfiguration on channels not involving data rate or protocol mode change, Intel recommends that you hold the channel transmitter (digital only) in reset during reconfiguration.
- When performing reconfiguration on channels not involving data rate or protocol mode change, Intel recommends that you hold the channel receiver (digital only) in reset during reconfiguration.

6.6. Steps to Perform Dynamic Reconfiguration

You can dynamically reconfigure blocks in the transceiver channel or PLL through the reconfiguration interface. The following procedure shows the steps required to reconfigure the channel and PLL blocks.

1. Check the **Enable Dynamic Reconfiguration** option in the **Dynamic Reconfiguration** tab in the Native PHY IP.
2. Select the desired configuration file type under **Configuration Files** option in the Native PHY IP.
3. Enable the desired dynamic reconfiguration features (such as multiple reconfiguration profiles, or feature blocks (such as embedded reconfiguration streamer and NPDME).
4. If you are using:
   a. Direct reconfiguration flow—Refer to the *Logical View of the L-Tile/H-Tile Transceiver Registers* for feature address and valid value of write data for the feature.
   b. IP guided reconfiguration flow—Note the settings of the base configuration and generate the corresponding configuration files. Also observe the settings of the modified configuration and generate the corresponding configuration files. Find out the differences in settings between the base and modified configurations.
   c. IP guided reconfiguration flow using multiple profiles—Create and store the parameter settings between the various configurations or profiles using configuration files. Find out the differences in settings between the various configurations or profiles using configuration files.
   d. IP guided reconfiguration flow using the embedded streamer—Refer to the *Logical View of the L-Tile/H-Tile Transceiver Registers* of the embedded reconfiguration streamer to stream the desired profile settings.

e. Reconfiguration flow for special cases—Refer to the lookup registers to be accessed for each special case, such as TX PLL switching, TX PLL reference clock switching, and RX CDR reference clock switching.

5. Assert the required channel resets (if necessary). Refer to *Recommendations for Dynamic Reconfiguration* for details on which resets need to be asserted.
6. If you are reconfiguring across data rates or protocol modes or enabling/disabling PRBS, place the channels in reset.
7. If you have background calibration enabled, disable it by setting channel offset address 0x542[0] to 0x0.
   You disabled it successfully if 0x542[0] = 0x0, 0x481[2] = 0x0, or `reconfig_waitrequest` is low.
8. You must perform this step only if you are reconfiguring fPLL/ATX PLL/CDR/CMU PLL. Otherwise, go to step 11. Request PreSICE to configure the fPLL/ATX PLL/CDR/CMU PLL in preparation for reconfiguration by setting the `pre_reconfig` bit for the fPLL/ATX PLL/CDR/CMU PLL.
   a. 1’b1: Request PreSICE to configure the fPLL/ATX PLL/CDR/CMU PLL in reconfiguration mode.
   b. 1'b0: Reconfiguration mode not requested.
9. You must perform this step only if you are reconfiguring fPLL/ATX PLL/CDR/CMU PLL. Otherwise, go to step 11. Also make sure you have performed step 8 before performing this step. Return the internal configuration bus access to PreSICE by writing a 0x01 to address 0x000 of the fPLL/ATX PLL/CDR/CMU PLL, and wait for PreSICE to complete the operation by monitoring the `pll_cal_busy` or `rx_cal_busy` signal or reading the `pll_cal_busy` or `rx_cal_busy` signal status from the status registers.
10. You must perform this step if you are reconfiguring the fPLL/ATX PLL/CDR/CMU PLL. Otherwise, go to step 11. Also, make sure you have performed step 9 before performing this step. Request internal configuration bus arbitration from PreSICE.
11. Perform the necessary reconfiguration using the flow described in the following sections: *Direct Reconfiguration Flow*, *Native PHY or PLL IP Guided Reconfiguration Flow*, and *Reconfiguration Flow for Special Cases*.
12. Perform all necessary reconfiguration. If reconfiguration involved datarate or protocol mode changes, you may have to reconfigure the PMA analog parameters of the channels. Refer to *Changing PMA Analog Parameters* for more details.
13. If reconfiguration involved datarate or protocol mode changes, request recalibration, and wait for the calibration to complete. Calibration is complete when `tx_cal_busy` or `rx_cal_busy` or `pll_cal_busy` is deasserted. For more details about calibration registers and the steps to perform recalibration, refer to *Calibration*.
14. If desired, enable background calibration by setting channel offset address 0x542[0] to 0x1.
    - The background calibration feature is only available for H-tile production devices starting with Intel Quartus Prime Design Suite 18.1 and if the data rate is >= 17.5 Gbps.
    - Refer to *Background Calibration* for more information.

Related Information

- Recommendations for Dynamic Reconfiguration on page 405

- Direct Reconfiguration Flow on page 409
- Native PHY IP or PLL IP Core Guided Reconfiguration Flow on page 410
- Reconfiguration Flow for Special Cases on page 411
- Changing Analog PMA Settings on page 418
- Calibration on page 433
- Background Calibration on page 442
- Logical View of the L-Tile/H-Tile Transceiver Registers on page 450

6.6.1. Channel Reconfiguration

1. If you have background calibration enabled, disable it by setting channel offset address 0x542[0] to 0x0.
   You disabled it successfully if 0x542[0] = 0x0, 0x481[2] = 0x0, or `reconfig_waitrequest` is low.
2. **Pause Traffic:** Assert the required channel resets (if necessary). Refer to the section *Recommendations for Dynamic Reconfiguration* for details on which resets need to be asserted. If you are reconfiguring across data rates or protocol modes or enabling/disabling PRBS, place the channels in reset.
3. **Modify:** Perform the necessary reconfiguration using the flow described in *Direct Reconfiguration Flow*, *Native PHY or PLL IP Guided Reconfiguration Flow*, and *Reconfiguration Flow for Special Cases*.
4. **Re-align:** Request recalibration, if reconfiguration involved data rate or protocol mode change, and wait for the calibration to complete. Calibration is complete when `tx/rx/pll_cal_busy` is deasserted. For more details about calibration registers and the steps to perform recalibration, refer to the *Calibration* chapter. If you reconfigured:
   a. TX simplex channel for data rate change—you must recalibrate the channel TX.
   b. RX simplex channel for data rate change—you must recalibrate the channel RX.
   c. Duplex channel for data rate change—you must recalibrate the channel RX, followed by the TX.
5. **Return Control:** After you have performed all necessary reconfiguration, return the internal configuration bus access to PreSICE with a direct write of 0x01 to offset address 0x000. You may have to reconfigure the PMA analog parameters of the channels. Refer to the *Changing PMA Analog Parameters* section for more details.
6. **Resume Traffic:** Deassert analog resets followed by digital resets. Refer to "Recommendations for Dynamic Reconfiguration" for details on the resets that needs to be deasserted.
7. If desired, enable background calibration by setting channel offset address 0x542[0] to 0x1.
   - The background calibration feature is only available for H-tile production devices starting with Intel Quartus Prime Design Suite 18.1 and if the data rate is >= 17.5 Gbps.
   - Refer to *Background Calibration* for more information.

Note: You cannot merge multiple reconfiguration interfaces across multiple IP blocks (merging independent instances of simplex TX/RX into the same physical location or merging separate CMU PLL and TX channel into the same physical location) when you enable the optional reconfiguration logic registers.

Related Information

- Recommendations for Dynamic Reconfiguration on page 405
- Native PHY IP or PLL IP Core Guided Reconfiguration Flow on page 410
- Direct Reconfiguration Flow on page 409
- Reconfiguration Flow for Special Cases on page 411
- Changing Analog PMA Settings on page 418
- Calibration on page 433
- Background Calibration on page 442

6.6.2. PLL Reconfiguration

1. For CMU PLL and CDR reconfiguration, if you have background calibration enabled, disable it by setting channel offset address 0x542[0] to 0x0. Skip this step if you are reconfiguring the ATX PLL or fPLL.
   You disabled it successfully if 0x542[0] = 0x0, 0x481[2] = 0x0, or `reconfig_waitrequest` is low.
2. **Pause Traffic:** Assert the required channel resets (if necessary). Refer to the section *Recommendations for Dynamic Reconfiguration* for details on which resets need to be asserted.
3. **Prepare PLL:** Request PreSICE to configure (set `pre_reconfig` bit) the fPLL/ATX PLL/CMU PLL/CDR in preparation for reconfiguration. To request PreSICE, read modify write to address 0x100, to modify `pre_reconfig` bit to value 1 for the fPLL/ATX PLL/CMU PLL/CDR.
   - 1’b1: Request PreSICE to configure the fPLL/ATX PLL/CMU PLL/CDR in reconfiguration mode.
   - 1'b0: Reconfiguration mode not requested.

Table 160. `pre_reconfig` **bitmapping**

TX PLL	Offset[bit]
ATX_PLL	0x100[1]
fPLL	0x100[0]
CMU PLL/CDR	0x100[3]

4. **Return Control:** Return the internal configuration bus access to PreSICE. To return control, direct write at address 0x000 with value 0x01 for fPLL/ATX PLL/CMU PLL/CDR. Wait for PreSICE to complete operation by monitoring the `pll_cal_busy/rx_cal_busy` signal or reading the `pll_cal_busy/rx_cal_busy` signal status from the status registers.
5. **Modify**: Perform the necessary reconfiguration using the flow described in *Direct Reconfiguration Flow*, *Native PHY or PLL IP Guided Reconfiguration Flow*, and *Reconfiguration Flow for Special Cases*.

6. **Re-align:** Request recalibration, if reconfiguration involved data rate or protocol mode change, and wait for the calibration to complete. Calibration is complete when `tx/rx/pll_cal_busy` is deasserted. For more details about calibration registers and the steps to perform recalibration, refer to the *Calibration* chapter. If you reconfigured PLL for data rate change—you must recalibrate the PLL and the channel TX.
7. **Return Control:** After you have performed all necessary reconfiguration and recalibration, return the internal configuration bus access to PreSICE. To return bus arbitration, direct write 0x01 to offset address 0x000. You may have to reconfigure the PMA analog parameters of the channels. Refer to *Changing PMA Analog Parameters* for more details.
8. **Resume Traffic:** Deassert analog resets followed by digital resets. Refer to "Recommendations for Dynamic Reconfiguration" for details on the resets that needs to be deasserted.
9. For CMU PLL and CDR reconfiguration and if desired, enable background calibration by setting channel offset address 0x542[0] to 0x1. Skip this step if you are reconfiguring the ATX PLL or fPLL.
    - The background calibration feature is only available for H-tile production devices starting with Intel Quartus Prime Design Suite 18.1 and if the data rate is >= 17.5 Gbps.
    - Refer to *Background Calibration* for more information.

Related Information

- Recommendations for Dynamic Reconfiguration on page 405
- Direct Reconfiguration Flow on page 409
- Native PHY IP or PLL IP Core Guided Reconfiguration Flow on page 410
- Reconfiguration Flow for Special Cases on page 411
- Changing Analog PMA Settings on page 418
- Calibration on page 433
- Background Calibration on page 442

6.7. Direct Reconfiguration Flow

Use this flow to perform dynamic reconfiguration when you know exactly which parameter and value to change for the transceiver channel or PLL. You can use this flow to change the PMA analog settings, enable/disable PRBS generator, and verifier hard blocks of the transceiver channel.

To perform dynamic reconfiguration using direct reconfiguration flow:

1. For PMA channel related registers, perform the necessary steps from steps 1 to 6 in *Channel Reconfiguration*, and, for PLL related registers, steps 1 to 9 in *PLL Reconfiguration* under *Steps to Perform Dynamic Reconfiguration*.
2. Read from the PMA analog feature address of the channel you want to change. For example, to change pre-emphasis 1st post-tap, read and store the value of address 0x105.
3. Perform a read-modify-write to feature address with a valid value. For example, to change the pre-emphasis 1st post-tap, write 5'b00001 to address 0x105.

Related Information

- Steps to Perform Dynamic Reconfiguration on page 405
- Channel Reconfiguration on page 407
- PLL Reconfiguration on page 408
- Changing Analog PMA Settings on page 418

6.8. Native PHY IP or PLL IP Core Guided Reconfiguration Flow

Use the Native PHY IP core or PLL IP core guided reconfiguration flow to perform dynamic reconfiguration when you need to change multiple parameters or parameters in multiple addresses for the transceiver channel or PLL. You can use this flow to change data rates, change clock divider values, or switch from one PCS datapath to another. You must generate the required configuration files for the base and modified Transceiver Native PHY IP core or PLL IP core configurations.

The configuration files contain addresses and bit values of the corresponding configuration. Compare the differences between the base and modified configuration files. The differences between these files indicate the addresses and bit values that must change to switch from one configuration to another. Perform read-modify-writes for the bit values that are different from the base configuration to obtain the modified configuration.

To perform dynamic reconfiguration using the IP Guided Reconfiguration Flow:

1. For PMA channel related registers, perform the necessary steps from steps 1 to 6 in *Channel Reconfiguration*, and, for PLL related registers, steps 1 to 9 in *PLL Reconfiguration* under *Steps to Perform Dynamic Reconfiguration*.
2. Perform a read-modify-write to all addresses and bit values that are different from the base configuration.

Note: If reconfiguration involved data rate or protocol mode changes, you may need to reconfigure the PMA analog parameters of the channels. Refer to the *Changing PMA Analog Parameters* section for more details.

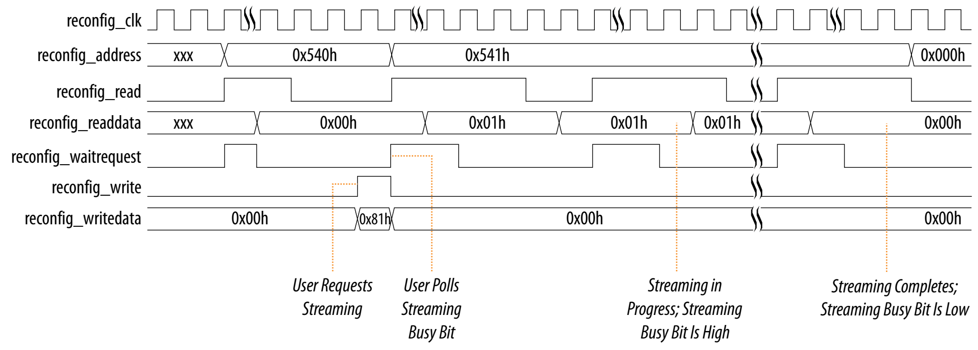
The bit values that must be changed to obtain the new configuration may span across multiple addresses, such as when switching between Standard, Enhanced, and PCS Direct data paths. It is difficult to manually compare these values for the base and modified configurations and then build logic to stream the different values in the modified configuration. You can use the multiple profiles feature of the Native PHY/Transmit PLL IP cores to store the parameter settings (MIF configuration file) to memory. With the configuration content saved, you can read from the memory and write the content to the target channel for reconfiguration. Optionally, you can also use the embedded reconfiguration streamer feature of the Native PHY/Transmit PLL IP cores, which includes the logic to store the individual profile information and logic to perform streaming. Using the embedded reconfiguration streamer, you can reduce the number of read-modify-write operations to obtain the modified configuration.

To perform dynamic reconfiguration using the Embedded Reconfiguration Streamer:

1. Perform the necessary steps from steps 1 to 13 in *Steps to Perform Dynamic Reconfiguration*.
2. Perform a read-modify-write to streamer control register with the appropriate bits.
3. Poll the streamer status register bit at regular intervals. The status register bits indicate when the reconfiguration is complete.

Note: If reconfiguration involved data rate or protocol mode changes, you may need to reconfigure the PMA analog parameters of the channels. Refer to the *Changing PMA Analog Parameters* section for more details.

Figure 236. Timing Diagram for Embedded Streamer Reconfiguration



Related Information

- Steps to Perform Dynamic Reconfiguration on page 405
- Channel Reconfiguration on page 407
- PLL Reconfiguration on page 408
- Changing Analog PMA Settings on page 418

6.9. Reconfiguration Flow for Special Cases

Dynamic reconfiguration can be performed on logical operations such as switching between multiple transmit PLLs or multiple reference clocks. In these cases, configuration files alone cannot be used. Configuration files are generated during IP generation and do not contain information on the placement of PLLs or reference clocks.

To perform dynamic reconfiguration on logical operations, you must use lookup registers that contain information about logical index to physical index mapping. Lookup registers are read-only registers. Use these lookup registers to perform a read-modify-write to the selection MUXes to switch between PLLs or reference clocks.

To perform dynamic reconfiguration using reconfiguration flow for special cases:

1. For PMA channel related registers, perform the necessary steps from steps 1 to 6 in *Channel Reconfiguration*, and, for PLL related registers, steps 1 to 9 in *PLL Reconfiguration* under *Steps to Perform Dynamic Reconfiguration*.
2. Read from the desired lookup register. Refer to the Switching Transmitter PLL on page 412 and Switching Reference Clocks on page 414 sections for information about lookup registers.
3. Perform Logical Encoding (only required for Transmitter PLL switching).
4. Perform read-modify-write to the required feature address with the desired/ encoded value.

Related Information

- Channel Reconfiguration on page 407
- PLL Reconfiguration on page 408

6.9.1. Switching Transmitter PLL

Dynamically switching data rates increases system flexibility to support multiple protocols. You can change the transceiver channel data rate by switching from one transmit PLL to another. To switch between transmit PLLs, you must reconfigure the local CGB MUX select lines of the channel by performing a channel reconfiguration. You can clock transceiver channels with up to four different transmitter PLLs. You can use the reconfiguration interface on the Native PHY IP core to specify which PLL drives the transceiver channel. The PLL switching method is the same, regardless of the number of transmitter PLLs involved.

Before initiating the PLL switch procedure, ensure that your Transceiver Native PHY instance defines more than one transmitter PLL input. Specify the **Number of TX PLL clock inputs per channel** parameter on the **TX PMA** tab during Transceiver Native PHY parameterization.

Refer to *Logical View of the L-Tile/H-Tile Transceiver Registers* for details on the registers and bits. The number of exposed `tx_serial_clk` bits varies according to the number of transmitter PLLs you specify. Use the Native PHY reconfiguration interface for this operation.

Table 161. Lookup Registers for Transmit PLL switching

Transceiver Native PHY Port	Description	Address	Bits
`tx_serial_clk0`	Represents logical PLL0. Lookup register `x117[3:0]` stores the mapping from logical PLL0 to the physical PLL.	0x117 (Lookup Register)	[3:0]
`tx_serial_clk1`	Represents logical PLL1. Lookup register `x117[7:4]` stores the mapping from logical PLL1 to the physical PLL.	0x117 (Lookup Register)	[7:4]

continued...

Transceiver Native PHY Port	Description	Address	Bits
`tx_serial_clk2`	Represents logical PLL2. Lookup register `x118[3:0]` stores the mapping from logical PLL2 to the physical PLL.	0x118 (Lookup Register)	[3:0]
`tx_serial_clk3`	Represents logical PLL3. Lookup register `x118[7:4]` stores the mapping from logical PLL3 to the physical PLL.	0x118 (Lookup Register)	[7:4]
N/A	PLL selection MUX	0x111	[7:0]

When performing a PLL switch, you must specify the lookup register address and bit values you want to switch to. The following procedure describes selection of a specific transmitter PLL when more than one PLL is connected to a channel. To change the data rate of the CDR, follow the detailed steps for reconfiguring channel and PLL blocks. After determining the logical PLL to switch to, follow this procedure to switch to the desired transmitter PLL:

1. Perform the necessary steps from steps 1 to 10 in *Steps to Perform Dynamic Reconfiguration*.
2. Read from the appropriate lookup register address and save the required 4-bit pattern. For example, switching to logical PLL1 requires saving bits [7:4] of address 0x117.
3. Encode the 4-bit value read in the previous step into an 8-bit value according to the following table:

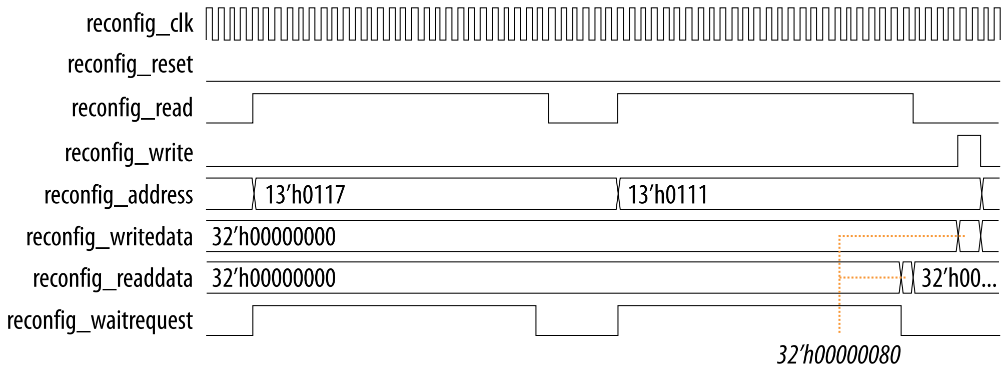
Table 162. Logical PLL Encoding

4-bit Logical PLL Bits	8-bit Mapping to Address 0x111
[3..0]	*{~logical_PLL_offset_readdata[3], logical_PLL_offset_readdata[1:0],logical_PLL_offset_readdata[3], logical_PLL_offset_readdata[3:0] }*
[7..4]	*{~logical_PLL_offset_readdata[7], logical_PLL_offset_readdata[5:4],logical_PLL_offset_readdata[7], logical_PLL_offset_readdata[7:4] }*

Note: For example, if reconfiguring to logical PLL1 then bits [7:4] are encoded to an 8-bit value {~bit[7], bit[5:4], bit[7], bit[7:4]}.

4. Perform a read-modify-write to bits[7:0] of address 0x111 using the encoded 8-bit value.
5. Perform the necessary steps from steps 12 to 14 in *Steps to Perform Dynamic Reconfiguration*.

Figure 237. TX PLL Switching



Related Information

- Logical View of the L-Tile/H-Tile Transceiver Registers on page 450
- Steps to Perform Dynamic Reconfiguration on page 405

6.9.2. Switching Reference Clocks

You can dynamically switch the input clock source for the ATX PLL, the fPLL, the CDR, and the CMU.

6.9.2.1. ATX Reference Clock Switching

You can use the reconfiguration interface on the ATX PLL instance to specify which reference clock source drives the ATX PLL. The ATX PLL supports clocking up to five different reference clock sources.

Before initiating a reference clock switch, ensure that your ATX PLL instance defines more than one reference clock source. Specify the **Number of PLL reference clocks** parameter on the **PLL** tab during ATX PLL parameterization.

The number of exposed `pll_refclk` ports varies according to the number of reference clocks you specify. Use the ATX PLL reconfiguration interface for this operation.

Table 163. Lookup Registers for Switching ATX PLL Reference Clock Inputs

Transceiver ATX PLL Port	Description	Address	Bits
`pll_refclk0`	Represents logical `refclk0`. Lookup register `x113[7:0]` stores the mapping from logical `refclk0` to the physical refclk.	0x113 (Lookup Register)	[7:0]
`pll_refclk1`	Represents logical `refclk1`. Lookup register `x114[7:0]` stores the mapping from logical `refclk1` to the physical refclk.	0x114 (Lookup Register)	[7:0]
`pll_refclk2`	Represents logical `refclk2`. Lookup register `x115[7:0]` stores the mapping from logical `refclk2` to the physical refclk.	0x115 (Lookup Register)	[7:0]

continued...

Send Feedback

Transceiver ATX PLL Port	Description	Address	Bits
`pll_refclk3`	Represents logical `refclk3`. Lookup register `x116[7:0]` stores the mapping from logical `refclk3` to the physical refclk.	0x116 (Lookup Register)	[7:0]
`pll_refclk4`	Represents logical `refclk4`. Lookup register `x117[7:0]` stores the mapping from logical `refclk4` to the physical refclk.	0x117 (Lookup Register)	[7:0]
N/A	ATX refclk selection MUX.	0x112	[7:0]

When performing a reference clock switch, you must specify the lookup register address and respective bits of the replacement clock. After determining the ATX PLL, follow this procedure to switch to the selected reference clock:

1. Perform the necessary steps from steps 1 to 10 in *Steps to Perform Dynamic Reconfiguration*.
2. Read from the lookup register address and save the required 8-bit pattern. For example, switching to logical `refclk2` requires use of bits`[7:0]` at address `0x115`.
3. Perform a read-modify-write to bits `[7:0]` at address `0x112` using the 8-bit value obtained from the lookup register.
4. Perform the necessary steps from steps 12 to 14 in *Steps to Perform Dynamic Reconfiguration*.

Related Information

Steps to Perform Dynamic Reconfiguration on page 405

6.9.2.2. fPLL Reference Clock Switching

You can use the reconfiguration interface on the fPLL instance to specify which reference clock source drives the fPLL. The fPLL supports clocking by up to five different reference clock sources.

Before initiating a reference clock switch, ensure that your fPLL instance defines more than one reference clock source. Specify the **Number of PLL reference clocks** parameter on the **PLL** tab during fPLL parameterization.

The number of exposed `pll_refclk` ports varies according to the number of reference clocks you specify. Use the fPLL reconfiguration interface for this operation.

Table 164. Register Map for Switching fPLL Reference Clock Inputs

Transceiver fPLL Port	Description	Address	Bits
`pll_refclk0`	Represents logical `refclk0` for MUX_0. Lookup register `x117[4:0]` stores the mapping from logical `refclk0` to the physical refclk for MUX_0.	0x117 (Lookup Register)	[7:0]
`pll_refclk1`	Represents logical `refclk1` for MUX_0. Lookup register `x118[4:0]` stores the mapping from logical `refclk1` to the physical refclk for MUX_0.	0x118 (Lookup Register)	[7:0]
`pll_refclk2`	Represents logical `refclk2` for MUX_0. Lookup register `x119[4:0]` stores the mapping from logical `refclk2` to the physical refclk for MUX_0.	0x119 (Lookup Register)	[7:0]

continued...

Transceiver fPLL Port	Description	Address	Bits
`pll_refclk3`	Represents logical `refclk3` for MUX_0. Lookup register x11A[4:0] stores the mapping from logical `refclk3` to the physical refclk for MUX_0.	0x11A (Lookup Register)	[7:0]
`pll_refclk4`	Represents logical `refclk4` for MUX_0. Lookup register x11B[4:0] stores the mapping from logical `refclk4` to the physical refclk for MUX_0.	0x11B (Lookup Register)	[7:0]
N/A	fPLL refclk selection MUX_0.	0x114	[7:0]
`pll_refclk0`	Represents logical `refclk0` for MUX_1. Lookup register x11D[4:0] stores the mapping from logical `refclk0` to the physical refclk for MUX_1.	0x11D (Lookup Register)	[7:0]
`pll_refclk1`	Represents logical `refclk1` for MUX_1. Lookup register x11E[4:0] stores the mapping from logical `refclk1` to the physical refclk for MUX_1.	0x11E (Lookup Register)	[7:0]
`pll_refclk2`	Represents logical `refclk2` for MUX_1. Lookup register x11F[4:0] stores the mapping from logical `refclk2` to the physical refclk for MUX_1.	0x11F (Lookup Register)	[7:0]
`pll_refclk3`	Represents logical `refclk3` for MUX_1. Lookup register x120[4:0] stores the mapping from logical `refclk3` to the physical refclk for MUX_1.	0x120 (Lookup Register)	[7:0]
`pll_refclk4`	Represents logical `refclk4` for MUX_1. Lookup register x121[4:0] stores the mapping from logical `refclk4` to the physical refclk for MUX_1.	0x121 (Lookup Register)	[7:0]
N/A	fPLL refclk selection MUX_1.	0x11C	[7:0]

Specify the logical reference clock and respective address and bits of the replacement clock when performing a reference clock switch. Follow this procedure to switch to the selected reference clock:

1. Perform the necessary steps from steps 1 to 10 in *Steps to Perform Dynamic Reconfiguration*.
2. Read from the lookup register for MUX 0 and save the required 8-bit pattern. For example, switching to logical `refclk3` requires use of bits[7:0] at address 0x11A.
3. Perform a read-modify-write to bits [7:0] at address 0x114 using the 8-bit value obtained from the lookup register.
4. Read from the lookup register for MUX 1 and save the required 8-bit pattern. For example, switching to logical `refclk3` requires use of bits[7:0] at address 0x120.
5. Perform a read-modify-write to bits [7:0] at address 0x11C using the 8-bit value obtained from the lookup register.
6. Perform the necessary steps from steps 12 to 14 in *Steps to Perform Dynamic Reconfiguration*.

Related Information

Steps to Perform Dynamic Reconfiguration on page 405

6.9.2.3. CDR and CMU Reference Clock Switching

You can use the reconfiguration interface to specify which reference clock source drives the CDR and CMU PLL. The CDR and CMU support clocking by up to five different reference clock sources.

Before initiating a reference clock switch, ensure that your CDR and CMU defines more than one reference clock source. For the CDR, specify the parameter on the **RX PMA** tab during the Native PHY IP parameterization. For the CMU, specify the **Number of PLL reference clocks** under the **PLL** tab when parameterizing the CMU PLL.

The number of exposed `rx_cdr_refclk` (CDR) or `pll_refclk` (CMU) varies according to the number of reference clocks you specify. Use the CMU reconfiguration interface for switching the CMU reference clock.

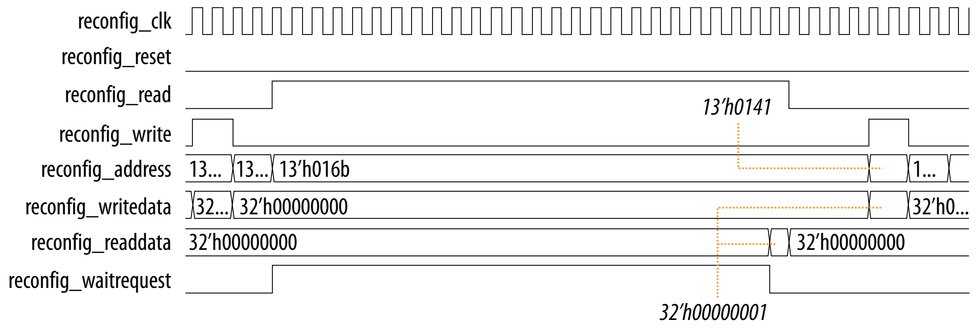
Table 165. Lookup Registers for Switching CDR Reference Clock Inputs

Native PHY Port	Description	Address	Bits
`cdr_refclk0`	Represents logical `refclk0`. Lookup register x16A`[7:0]` stores the mapping from logical `refclk0` to the physical refclk.	0x16A (Lookup Register)	[7:0]
`cdr_refclk1`	Represents logical `refclk1`. Lookup register x16B`[7:0]` stores the mapping from logical `refclk1` to the physical refclk.	0x16B (Lookup Register)	[7:0]
`cdr_refclk2`	Represents logical `refclk2`. Lookup register x16C`[7:0]` stores the mapping from logical `refclk2` to the physical refclk.	0x16C (Lookup Register)	[7:0]
`cdr_refclk3`	Represents logical `refclk3`. Lookup register x16D`[7:0]` stores the mapping from logical `refclk3` to the physical refclk.	0x16D (Lookup Register)	[7:0]
`cdr_refclk4`	Represents logical `refclk4`. Lookup register x16E`[7:0]` stores the mapping from logical `refclk4` to the physical refclk.	0x16E (Lookup Register)	[7:0]
N/A	CDR refclk selection MUX.	0x141	[7:0]

When performing a reference clock switch, note the logical reference clock to switch to and the respective address and bits. After determining the logical reference clock, follow this procedure to switch to the selected CDR reference clock:

1. Perform the necessary steps from steps 1 to 10 in *Steps to Perform Dynamic Reconfiguration*.
2. Read from the lookup register and save the required 8-bit pattern. For example, switching to logical `refclk3` requires saving bits`[7:0]` at address `0x16D`.
3. Perform a read-modify-write to bits `[7:0]` at address `0x141` using the 8-bit value obtained from the lookup register.
4. Perform the necessary steps from steps 12 to 14 in *Steps to Perform Dynamic Reconfiguration*.

Figure 238. CDR Reference Clock Switching



Related Information

Steps to Perform Dynamic Reconfiguration on page 405

6.9.3. Reconfiguring Between GX and GXT Channels

All GXT channels can be reconfigured to GX channels (≤ 17.4Gbps).

The receive datapath can be reconfigured by changing any of the following values in the Native PHY:

- Changing the reference clock source
- Changing the CDR's M/N/L counter values

The transmit datapath can be reconfigured by either of the following methods:

- An ATX PLL drives the GXT clock lines, and a different ATX PLL/fPLL drives the GX clock lines. The Native PHY IP core has two TX clock inputs selected.
  - Change the TX serial clock source in the Native PHY IP core to select between GXT and GX datarates.
- A single ATX PLL drives the GXT clock line and GX clock lines. Both the `tx_serial_clk` and `tx_serial_clk_gt` output ports must be enabled in the ATX PLL. The Native PHY IP core is configured to have two TX clock inputs and are connected to the serial clock ports in the ATX PLL.
  - The ATX PLL is reconfigured when switching between GXT and GX datarates. The ATX PLL needs to be recalibrated after reconfiguration.
  - The Native PHY IP core switches between the two `tx_serial_clk` sources.

The Native PHY IP core needs to be recalibrated if the receive or transmit datarate changes. The Native PHY recalibration must occur after the ATX PLL has locked if a single ATX PLL is used for the GXT and GX datarate for transmit (the second method above).

Note: The **TX local division factor** parameter in the Native PHY IP core cannot be used to switch between GXT and GX datarates when the GXT datarate is a multiple of the GX datarate.

6.10. Changing Analog PMA Settings

As referenced in the "PMA Functions" chapter, you can use the reconfiguration interface to change your analog PMA settings.

The PMA analog settings can be broadly divided into the following groups:

- PMA analog settings that are channel or system dependent:
  - These settings may vary from channel to channel based on channel loss or other factors
  - You can set these PMA analog settings based on IBIS-AMI or Advanced Link Analyzer simulations
  - Settings under this category include: VOD, TX pre-emphasis, VGA, CTLE, DFE

Refer to the *Logical View of the L-Tile/H-Tile Transceiver Registers* for the attribute names and addresses of the relevant registers for reconfiguration.

- PMA analog settings that are transceiver protocol-type, data rate dependent, or both:
  - These settings may vary for each transceiver protocol-type, data rate, or both in your design. You can set these PMA Analog settings using direct reconfiguration flow by performing RMWs to the respective registers of each channel (OR).
  - These are important to reconfigure when changing datarates.
  - Settings under this category include: slew rate, equalizer bandwidth, compensation enable.

Refer to the *Logical View of the L-Tile/H-Tile Transceiver Registers* for the attribute names and addresses of the relevant registers for reconfiguration.

Related Information

Logical View of the L-Tile/H-Tile Transceiver Registers on page 450

6.11. Ports and Parameters

The reconfiguration interface is integrated in the Native PHY instance and the TX PLL instances. Instantiate the Native PHY and the TX PLL IP cores in Qsys by clicking **Tools ➤ IP Catalog**. You can define parameters for IP cores by using the IP core-specific parameter editor. To expose the reconfiguration interface ports, select the **Enable dynamic reconfiguration** option when parameterizing the IP core.

You can share the reconfiguration interface among all the channels by turning on **Share reconfiguration interface** when parameterizing the IP core. When this option is enabled, the IP core presents a single reconfiguration interface for dynamic reconfiguration of all channels. Address bits [10:0] provide the register address in the reconfiguration space of the selected channel. The remaining address bits of the reconfiguration address specify the selected logical channel. For example, if there are four channels in the Native PHY IP instance, `reconfig_address[10:0]` specifies the address and `reconfig_address[12:11]` are binary encoded to specify the four channels. For example, 2'b01 in `reconfig_address[12:11]` specifies logical channel 1.

The following figure shows the signals available when the Native PHY IP core is configured for four channels and the **Share reconfiguration interface** option is enabled.

Figure 239. Signals Available with Shared Native PHY Reconfiguration Interface

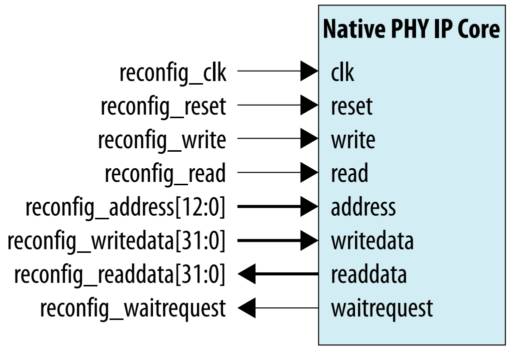


Table 166. Reconfiguration Interface Ports with Shared Native PHY Reconfiguration Interface

The reconfiguration interface ports when **Share reconfiguration interface** is enabled. <*N*> represents the number of channels.

Port Name	Direction	Clock Domain	Description
`reconfig_clk`	Input	N/A	Avalon clock. The clock frequency is up to 150 MHz.
`reconfig_reset`	Input	`reconfig_clk`	Resets the Avalon interface. Asynchronous assertion and synchronous deassertion.
`reconfig_write`	Input	`reconfig_clk`	Write enable signal. Signal is active high.
`reconfig_read`	Input	`reconfig_clk`	Read enable signal. Signal is active high.
`reconfig_address[log2<N>+10:0]`	Input	`reconfig_clk`	Address bus. The lower 11 bits specify address and the upper bits specify the channel.
`reconfig_writedata[31:0]`	Input	`reconfig_clk`	A 32-bit data write bus. Data to be written into the address indicated by `reconfig_address`.
`reconfig_readdata[31:0]`	Output	`reconfig_clk`	A 32-bit data read bus. Valid data is placed on this bus after a read operation. Signal is valid after `reconfig_waitrequest` goes high and then low.
`reconfig_waitrequest`	Output	`reconfig_clk`	A one-bit signal that indicates the Avalon interface is busy. Keep the Avalon command asserted until the interface is ready to proceed with the read/write transfer. The behavior of this signal depends on whether the feature **Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE** is enabled or not. For more details, refer to the *Calibration* section.

When **Share reconfiguration interface** is off, the Native PHY IP core provides an independent reconfiguration interface for each channel. For example, when a reconfiguration interface is not shared for a four-channel Native PHY IP instance, `reconfig_address[10:0]` corresponds to the reconfiguration address bus of logical channel 0, `reconfig_address[21:11]` correspond to the reconfiguration address bus of logical channel 1, `reconfig_address[32:22]` corresponds to the reconfiguration address bus of logical channel 2, and `reconfig_address[43:33]` correspond to the reconfiguration address bus of logical channel 3.

The following figure shows the signals available when the Native PHY is configured for four channels and the **Share reconfiguration interface** option is not enabled.

Figure 240. Signals Available with Independent Native PHY Reconfiguration Interfaces

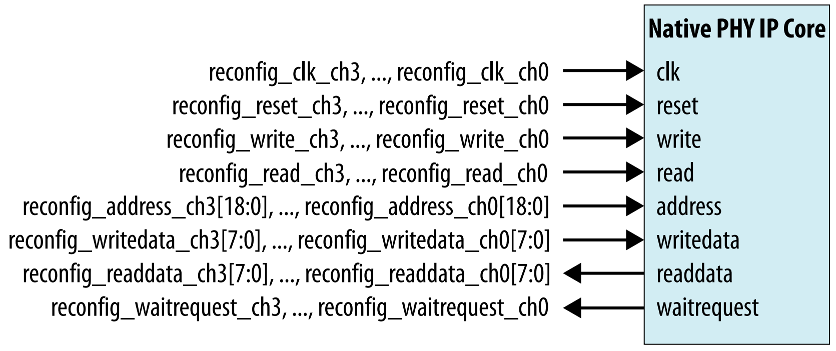

Native PHY IP Core
reconfig_clk_ch3, ..., reconfig_clk_ch0
clk
reconfig_reset_ch3, ..., reconfig_reset_ch0
reset
reconfig_write_ch3, ..., reconfig_write_ch0
write
reconfig_read_ch3, ..., reconfig_read_ch0
read
reconfig_address_ch3[18:0], ..., reconfig_address_ch0[18:0]
address
reconfig_writedata_ch3[7:0], ..., reconfig_writedata_ch0[7:0]
writedata
reconfig_readdata_ch3[7:0], ..., reconfig_readdata_ch0[7:0]
readdata
reconfig_waitrequest_ch3, ..., reconfig_waitrequest_ch0
waitrequest

Table 167. Reconfiguration Interface Ports with Independent Native PHY Reconfiguration Interfaces

The reconfiguration interface ports when **Share reconfiguration interface** is disabled. <*N*> represents the number of channels.

Port Name	Direction	Clock Domain	Description
`reconfig_clk[N-1:0]`	Input	N/A	Avalon clock for each channel. The clock frequency is up to 150 MHz.
`reconfig_reset[N-1:0]`	Input	`reconfig_clk`	Resets the Avalon interface for each channel. Asynchronous to assertion and synchronous to deassertion.
`reconfig_write[N-1:0]`	Input	`reconfig_clk`	Write enable signal for each channel. Signal is active high.
`reconfig_read[N-1:0]`	Input	`reconfig_clk`	Read enable signal for each channel. Signal is active high.
`reconfig_address[N*11-1:0]`	Input	`reconfig_clk`	A 11-bit address bus for each channel.
`reconfig_writedata[N*32-1:0]`	Input	`reconfig_clk`	A 32-bit data write bus for each channel. Data to be written into the address indicated by the corresponding address field in `reconfig_address`.
`reconfig_readdata[N*32-1:0]`	Output	`reconfig_clk`	A 32-bit data read bus for each channel. Valid data is placed on this bus after a read operation. Signal is valid after `waitrequest` goes high and then low.
`reconfig_waitrequest[N-1:0]`	Output	`reconfig_clk`	A one-bit signal for each channel that indicates the Avalon interface is busy. Keep the Avalon command asserted until the interface is ready to proceed with the read/ write transfer. The behavior of this signal depends on whether the feature **Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE** is enabled or not. For more details, refer to the *Calibration* section.

Table 168. Avalon Interface Parameters

The following parameters are available in the **Dynamic Reconfiguration** tab of the Transceiver Native PHY and TX PLL parameter editors.

Parameter	Value	Description
Enable dynamic reconfiguration	On / Off	Available in Native PHY and TX PLL IP parameter editors. Enables the reconfiguration interface. Off by default. The reconfiguration interface is exposed when this option is enabled.
Share reconfiguration interface	On / Off	Available in Native PHY IP parameter editor only. Enables you to use a single reconfiguration interface to control all channels. Off by default. If enabled, the uppermost bits of `reconfig_address` identifies the active channel. The lower 11 bits specify the reconfiguration address. Binary encoding is used to identify the active channel (available only for Transceiver Native PHY). Enable this option if the Native PHY is configured with more than one channel.
Enable Native PHY Debug Master Endpoint	On / Off	Available in Native PHY and TX PLL IP parameter editors. When enabled, the Native PHY Debug Master Endpoint (NPDME) is instantiated and has access to the Avalon memory-mapped interface of the Native PHY. You can access certain test and debug functions using System Console with the NPDME. Refer to the *Embedded Debug Features* section for more details about NPDME.
Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE	On / Off	When enabled, `reconfig_waitrequest` does not indicate the status of Avalon memory-mapped interface arbitration with PreSICE. The Avalon memory-mapped interface arbitration status is reflected in a soft status register bit. This feature requires that the **Enable control and status registers** feature under **Optional Reconfiguration Logic** be enabled. Refer to *Arbitration* for more details on this feature. Refer to the *Calibration* chapter for more details about calibration.
Enable rcfg_tx_digitalreset_release_ctrl port	On / Off	Available in Native PHY IP parameter editor only. Enables the `rcfg_tx_digitalreset_release_ctrl` port that dynamically controls the TX PCS reset release sequence. This port is mandatory when reconfiguring to/from TX PCS Gearbox ratio of *:67. Deassert this signal at least 30 ns before deasserting `tx_digitalreset` when the TX PCS Gearbox ratio is configured to *:67. In other modes, assert this signal at least 30 ns before deasserting `tx_digitalreset`.
Enable capability registers	On / Off	Available in Native PHY and TX PLL IP parameter editors. Enables capability registers. These registers provide high-level information about the transceiver channel's /PLL's configuration.
Set user-defined IP identifier	User-specified	Available in Native PHY and TX PLL IP parameter editors. Sets a user-defined numeric identifier that can be read from the `user_identifier` offset when the capability registers are enabled.
Enable control and status registers	On / Off	Available in Native PHY and TX PLL IP parameter editors. Enables soft registers for reading status signals and writing control signals on the PHY /PLL interface through the NPDME or reconfiguration interface.
Enable PRBS soft accumulators	On / Off	Available in Native PHY IP parameter editor only. Enables soft logic to perform PRBS bit and error accumulation when using the hard PRBS generator and verifier.
Configuration file prefix	User-specified	Available in Native PHY and TX PLL IP parameter editors. Specifies the file prefix used for generating configuration files. Use a unique prefix for configuration files for each variant of the Native PHY and PLL.

continued...

Parameter	Value	Description
Generate SystemVerilog package file	On / Off	Available in Native PHY and TX PLL IP parameter editors. Creates a SystemVerilog package file that contains the current configuration data values for all reconfiguration addresses. Disabled by default.
Generate C header file	On / Off	Available in Native PHY and TX PLL IP parameter editors. Creates a C header file that contains the current configuration data values for all reconfiguration addresses. Disabled by default.
Generate MIF (Memory Initialize File)	On / Off	Available in Native PHY and TX PLL IP parameter editors. Creates a MIF file that contains the current configuration data values for all reconfiguration addresses. Disabled by default.
Enable multiple reconfiguration profiles	On / Off	Available in Native PHY and Transmit PLL IP parameter editors only. Use the Parameter Editor to store multiple configurations. The parameter settings for each profile are tabulated in the Parameter Editor.
Enable embedded reconfiguration streamer	On / Off	Available in Native PHY and Transmit PLL IP parameter editors only. Embeds the reconfiguration streamer into the Native PHY/ Transmit PLL IP cores and automates the dynamic reconfiguration process between multiple predefined configuration profiles.
Generate reduced reconfiguration files	On / Off	Available in Native PHY and Transmit PLL IP parameter editors only. Enables the Native PHY and Transmit PLL IP cores to generate reconfiguration files that contain only the attributes that differ between multiple profiles.
Number of reconfiguration profiles	1 to 8	Available in Native PHY and Transmit PLL IP parameter editors only. Specifies the number of reconfiguration profiles to support when multiple reconfiguration profiles are enabled.
Selected reconfiguration profile	0 to 7	Available in Native PHY and Transmit PLL IP parameter editors only. Selects which reconfiguration profile to store when you click **Store profile**.
Store configuration to selected profile	N/A	Available in Native PHY and Transmit PLL IP parameter editors only. Stores the current Native PHY and Transmit PLL parameter settings to the profile specified by the **Selected reconfiguration profile** parameter.
Load configuration from selected profile	N/A	Available in Native PHY and Transmit PLL IP parameter editors only. Loads the current Native PHY/Transmit PLL IP with parameter settings from the stored profile specified by the **Selected reconfiguration profile** parameter.
Clear selected profile	N/A	Available in Native PHY and Transmit PLL IP parameter editors only. Clears the stored Native PHY/Transmit PLL IP parameter settings for the profile specified by the **Selected reconfiguration profile** parameter. An empty profile defaults to the current parameter settings of the Native PHY/Transmit PLL. In other words, an empty profile reflects the Native PHY/Transmit PLL current parameter settings.
Clear all profiles	N/A	Available in Native PHY and Transmit PLL IP parameter editors only. Clears the Native PHY/Transmit PLL IP parameter settings for all the profiles.
Refresh selected_profile	N/A	Available in Native PHY and Transmit PLL IP parameter editors only. Equivalent to clicking the **Load configuration from selected profile** and **Store configuration to selected profile** buttons in sequence. This operation loads the parameter settings from stored profile specified by the **Selected reconfiguration profile** parameter and then stores the parameters back to the profile.

Related Information

- Calibration on page 433

- Arbitration on page 403
- Embedded Debug Features on page 426
- Changing Analog PMA Settings on page 418

6.12. Dynamic Reconfiguration Interface Merging Across Multiple IP Blocks

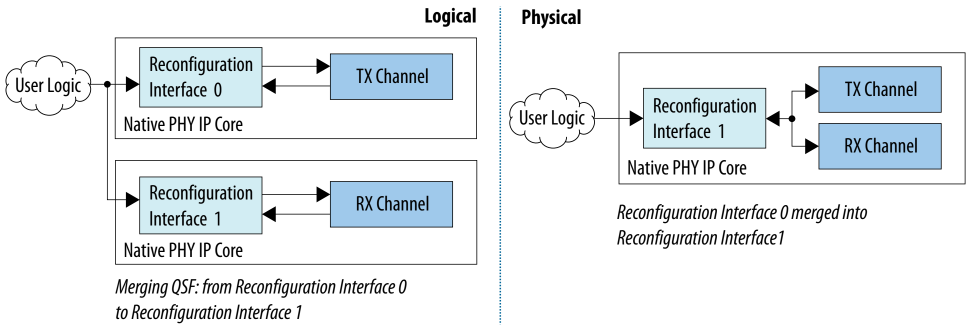
Dynamic reconfiguration interfaces may need to be shared between multiple IP blocks to maximize transceiver channel utilization. The Native PHY provides the ability to create channels that are either simplex or duplex instances. However, each physical transceiver channel in Intel Stratix 10 devices is fully duplex.

You can share the reconfiguration interfaces across different IP blocks by manually making a QSF assignment. There are two cases where a dynamic reconfiguration interface might need to be shared between multiple IP blocks:

- Independent instances of simplex receivers and transmitters in the same physical location
- Separate CMU PLL and TX channel in the same physical location

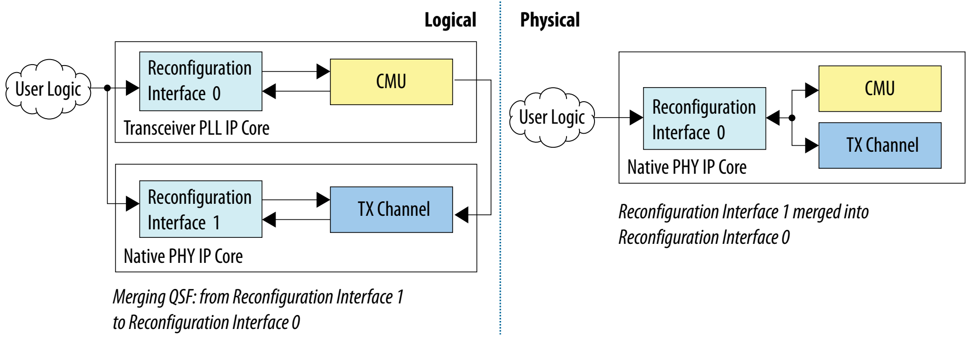
The following example shows one Native PHY IP instance of a TX-only channel and another instance of an RX-only channel.

Figure 241. Independent Instances of Simplex TX/RX in the Same Physical Location


Logical
Physical
User Logic
Reconfiguration Interface 0
TX Channel
Native PHY IP Core
Reconfiguration Interface 1
RX Channel
Native PHY IP Core
Merging QSF: from Reconfiguration Interface 0 to Reconfiguration Interface 1
User Logic
Reconfiguration Interface 1
TX Channel
RX Channel
Native PHY IP Core
Reconfiguration Interface 0 merged into Reconfiguration Interface1

The following example shows one Native PHY IP instance of a TX-only channel and an instance of a CMU PLL.

Figure 242. Separate CMU PLL and TX Channel in the Same Physical Location


Logical
Physical
User Logic
Reconfiguration Interface 0
CMU
Transceiver PLL IP Core
Reconfiguration Interface 1
TX Channel
Native PHY IP Core
Merging QSF: from Reconfiguration Interface 1 to Reconfiguration Interface 0
User Logic
Reconfiguration Interface 0
CMU
TX Channel
Native PHY IP Core
Reconfiguration Interface 1 merged into Reconfiguration Interface 0

Rules for Merging Reconfiguration Interfaces Across Multiple IP Cores

To merge reconfiguration interfaces across multiple IP blocks, you must follow these rules:

1. The control signals for the reconfiguration interfaces of the IP blocks must be driven by the same source. The `reconfig_clk`, `reconfig_reset`, `reconfig_write`, `reconfig_read`, `reconfig_address`, and `reconfig_writedata` ports of the two interfaces to be merged must be driven from the same source.
2. You must make a QSF assignment to manually specify which two reconfiguration interfaces are to be merged.
    a. Use the **XCVR_RECONFIG_GROUP** assignment.
    b. Set the **To** field of the assignment to either the reconfiguration interfaces of the instances to be merged or to the pin names. The reconfiguration interface has the string **inst_ct1_xcvr_avmm1**.
    c. Assign the two instances to be merged to the same reconfiguration group.

You cannot merge multiple reconfiguration interfaces when NPDME, optional reconfiguration logic, or embedded reconfiguration streamer are enabled in the Native PHY IP core. [48]

You cannot merge the TX and RX channels when the **Shared reconfiguration interface** parameter is enabled in the Native PHY IP core Parameter Editor. You can merge channels only if the reconfiguration interfaces are independent.

Refer to the following two examples to merge reconfiguration interfaces.

Example 2. Using reconfiguration interface names

This example shows how to merge a transmit-only Native PHY instance with a receive-only instance using the reconfiguration interface names. These instances are assigned to reconfiguration group 0.

For Native PHY 0—transmit-only instance:

```
set_instance_assignment -name XCVR_RECONFIG_GROUP 0 -to
topdesign:topdesign_inst|<TX only instance name>*ct1_hssi_avmm1_if_inst-
>inst_ct1_xcvr_avmm1
```

For Native PHY 1—receive-only instance to be merged with Native PHY 0:

```
set_instance_assignment -name XCVR_RECONFIG_GROUP 0 -to
topdesign:topdesign_inst|<RX only instance name>*ct1_hssi_avmm1_if_inst-
>inst_ct1_xcvr_avmm1
```

Example 3. Using pin names

This example shows how to merge a transmit-only Native PHY instance with a receive-only instance using pin names. These instances are assigned to reconfiguration group 1.

(48) Please refer to *Calibration* section on how to calibrate when those features are not available.

For Native PHY 0—transmit-only instance:

```
set_instance_assignment -name XCVR_RECONFIG_GROUP 1 -to tx[0]
```

For Native PHY 1—receive-only instance to be merged with Native PHY 0:

```
set_instance_assignment -name XCVR_RECONFIG_GROUP 1 -to rx[0]
```

Related Information

Calibration on page 433

6.13. Embedded Debug Features

The Intel Stratix 10 Transceiver Native PHY, ATX PLL, fPLL, and CMU PLL IP cores provide the following optional debug features to facilitate embedded test and debug capability:

- Native PHY Debug Master Endpoint (NPDME)
- Optional Reconfiguration Logic

6.13.1. Native PHY Debug Master Endpoint (NPDME)

The NPDME is a JTAG-based Avalon memory-mapped interface master that provides access to the transceiver and PLL registers through the system console. You can enable NPDME using the **Enable Native PHY Debug Master Endpoint** option available under the **Dynamic Reconfiguration** tab in the Native PHY and PLL IP cores. When using NPDME, the Quartus Prime software inserts the debug interconnect fabric to connect with USB, JTAG, or other net hosts. Select the **Share Reconfiguration Interface** parameter when the Native PHY IP instance has more than one channel. The transceiver Toolkit uses NPDME and it is a useful tool in debugging transceiver links.

When you enable NPDME in your design, you must:

- connect an Avalon memory-mapped interface master to the reconfiguration interface.
- OR connect the `reconfig_clk`, `reconfig_reset` signals and ground the `reconfig_write`, `reconfig_read`, `reconfig_address` and `reconfig_write` data signals of the reconfiguration interface. If the reconfiguration interface signals are not connected appropriately, there is no clock or reset for the NPDME, and the NPDME does not function as expected.

  Refer to the example connection below:

  .reconfig_clk (mgmt_clk),

  .reconfig_reset (mgmt_reset),

  .reconfig_write (1'b0),

  .reconfig_address (11'b0),

  .reconfig_read (1'b0),

  .reconfig_writedata (32'b0),

Note: You cannot merge multiple reconfiguration interfaces when NPDME is enabled.

Related Information

Debugging Transceiver Links

6.13.2. Optional Reconfiguration Logic

The Intel Stratix 10 Transceiver Native PHY, ATX PLL, fPLL, and CMU PLL IP cores contain soft logic for debug purposes known as the Optional Reconfiguration Logic. This soft logic provides a set of registers that enable you to determine the state of the Native PHY and PLL IP cores.

You can enable the following optional reconfiguration logic options in the transceiver Native PHY and PLL IP cores:

- Capability registers
- Control and status registers
- PRBS Soft Accumulators

Related Information

Dynamic Reconfiguration Parameters on page 60

6.13.2.1. Capability Registers

The capability registers provide high level information about the transceiver channel and PLL configuration.

The capability registers capture a set of chosen capabilities of the PHY and PLL that cannot be reconfigured.

Note: You can find a full list of capability registers in *Capability Registers* section. Refer to the *Logical View of the L-Tile/H-Tile Transceiver Registers* for the names and addresses of these registers.

Related Information

- Capability Registers on page 437
- Logical View of the L-Tile/H-Tile Transceiver Registers on page 450

6.13.2.2. Control and Status Registers

Control and status registers are optional registers that memory-map the status outputs from and control inputs to the Native PHY and PLL.

Refer to *Logical View of the L-Tile/H-Tile Transceiver Registers* for details on Control and Status registers.

Related Information

Logical View of the L-Tile/H-Tile Transceiver Registers on page 450

6.13.2.3. PRBS Soft Accumulators

This feature enables soft logic to perform bit and error accumulation when using the hard PRBS generator and verifier.

Refer to the "Debug Functions" section for more information, and to the *Logical View of the L-Tile/H-Tile Transceiver Registers* for the complete list of registers needed to use this feature.

Related Information

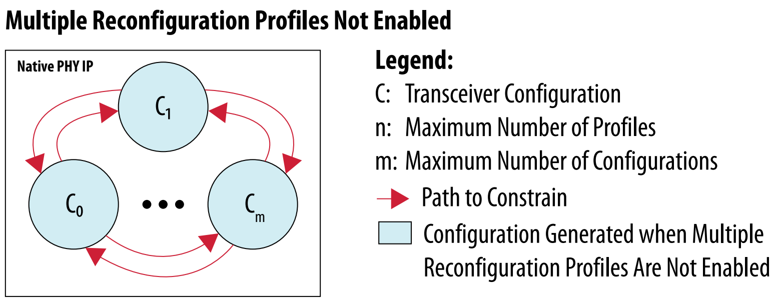
- Debug Functions on page 154
- Logical View of the L-Tile/H-Tile Transceiver Registers on page 450

6.14. Timing Closure Recommendations

You must close timing for the reconfiguration profiles in the following cases:

1. When the multiple reconfiguration profile is not enabled. You must manually include constraints for all the modified and target configuration. See example below:

Figure 243. Multiple Reconfiguration Profile Not Enabled

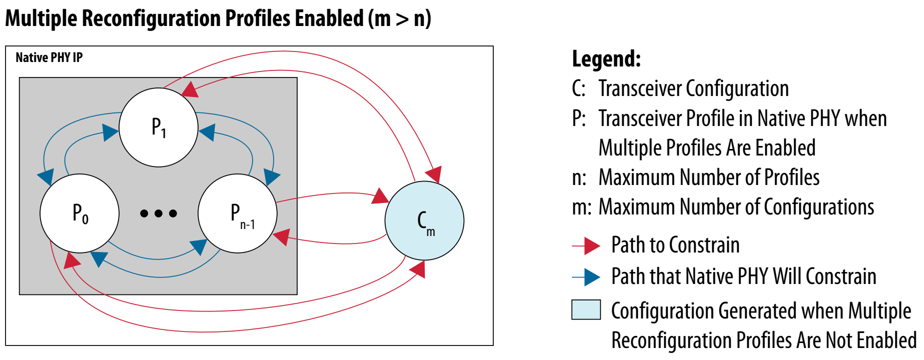

Multiple Reconfiguration Profiles Not Enabled
Native PHY IP
C_1
C_0
C_m
Legend:
C: Transceiver Configuration
n: Maximum Number of Profiles
m: Maximum Number of Configurations
Path to Constrain
Configuration Generated when Multiple Reconfiguration Profiles Are Not Enabled

2. When the multiple reconfiguration profile is enabled, and

   Case 1: More than 8 configurations are required:

   When you have exhausted maximum configurations (eight) supported by Native PHY IP. You must manually include constraints for all the modified and target configuration outside of the 8 configurations supported by the Native PHY IP.

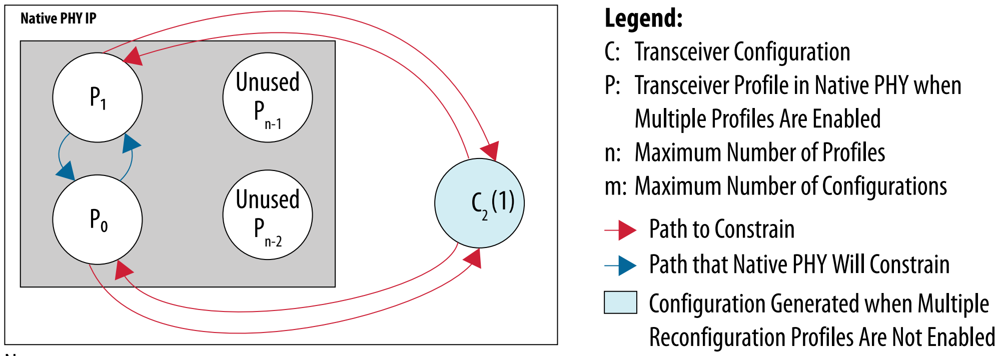
Figure 244. Multiple Reconfiguration Profile is Enabled (n=8)


Multiple Reconfiguration Profiles Enabled (m > n)
Native PHY IP
P_1
P_0
P_{n-1}
C_m
Legend:
C: Transceiver Configuration
P: Transceiver Profile in Native PHY when Multiple Profiles Are Enabled
n: Maximum Number of Profiles
m: Maximum Number of Configurations
Path to Constrain
Path that Native PHY Will Constrain
Configuration Generated when Multiple Reconfiguration Profiles Are Not Enabled

   Case 2: Some configurations are generated with Multiple Reconfiguration Profile disabled.

Figure 245.

Multiple Reconfiguration Profiles Enabled (n = 8)


Native PHY IP
P_1
Unused P_{n-1}
P_0
Unused P_{n-2}
C_2 (1)
Legend:
C: Transceiver Configuration
P: Transceiver Profile in Native PHY when Multiple Profiles Are Enabled
n: Maximum Number of Profiles
m: Maximum Number of Configurations
Path to Constrain
Path that Native PHY Will Constrain
Configuration Generated when Multiple Reconfiguration Profiles Are Not Enabled

Note:

(1) You must manually include constraints for all the modified and target configurations that were generated with Multiple Reconfiguration Profile disabled.

Note: Native PHY IP, supports up to eight reconfiguration profiles and generates appropriate SDC constraints for all paths.

Intel recommends that you enable the multiple reconfiguration profiles feature in the Native PHY IP core if any of the modified or target configurations involve changes to PCS settings.

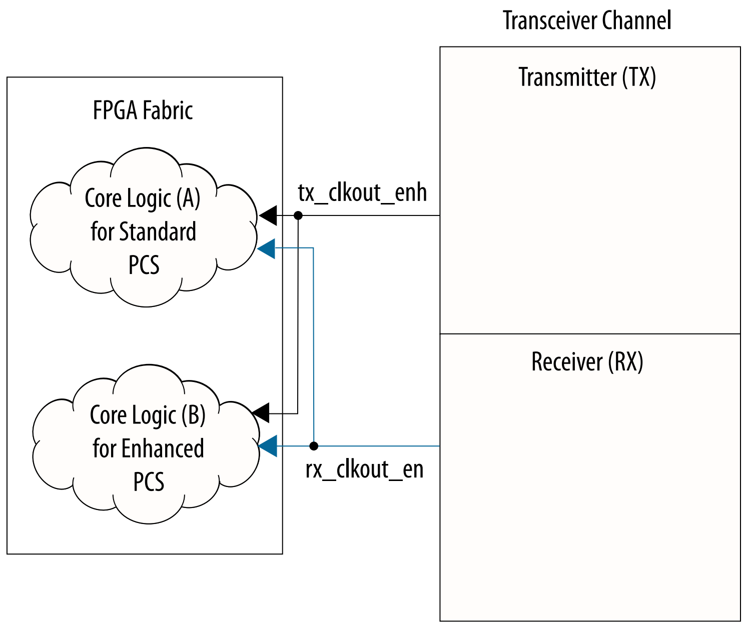
Using multiple reconfiguration profiles is optional if the reconfiguration involves changes to only PMA settings such as PLL switching, CGB divider switching, and refclk switching.

When supporting dynamic reconfiguration, you must:

- Include constraints to create the extra clocks for all modified or target configurations at the PCS-FPGA fabric interface. Clocks for the base configuration are created by the Quartus Prime software. These clocks enable the Intel Quartus Prime Pro Edition to perform static timing analysis for all the transceiver configuration profiles and their corresponding FPGA fabric core logic blocks.
- Include the necessary false paths between the PCS – FPGA fabric interface and the core logic.

For example, you can perform dynamic reconfiguration to switch the datapath from Standard PCS to Enhanced PCS using the multiple reconfiguration profiles feature. In the following example, the base configuration uses the Standard PCS (data rate = 1.25 Gbps, PCS-PMA width = 10) and drives core logic A in the FPGA fabric. The target or modified configuration is configured to use the Enhanced PCS (data rate = 12.5 Gbps, PCS-PMA width = 64) and drives core logic B in the FPGA fabric.

Figure 246. Using Multiple Reconfiguration Profiles



In the above example, you must

- create the `tx_clkout` clock that is used to clock the core logic B in the FPGA fabric.
- Based on how the clocks are connected in the design, you might have to include additional constraints to set false paths from the registers in the core logic to the clocks. For example,

```
set_false_path -from [get_clocks {tx_clkout_enh}] -to [get_registers <Core
Logic A>]
```

```
set_false_path -from [get_clocks {rx_clkout_enh}] -to [get_registers <Core
Logic A>]
```

```
set_false_path -from [get_clocks {tx_clkout}] -to [get_registers <Core
Logic B>]
```

```
set_false_path -from [get_clocks {rx_clkout}] -to [get_registers <Core
Logic B>]
```

6.15. Unsupported Features

The following features are not supported by either the Transceiver Native PHY IP core or the PLL IP reconfiguration interface:

- Reconfiguration from a bonded configuration to a non-bonded configuration, or vice versa
- Reconfiguration from a bonded protocol to another bonded protocol
- Reconfiguration from PCIe (with Hard IP) to PCIe (without Hard IP) or non-PCIe bonded protocol switching
- Master CGB reconfiguration

- Switching between two master CGBs
- Serialization factor changes on bonded channels
- TX PLL switching on bonded channels
- Reconfiguration between FIFO modes
- Reconfiguration of an ATX PLL from GX to GXT mode if the adjacent master CGB is being used

Note: Transceiver Native PHY IP non-bonded configuration to another Transceiver Native PHY IP non-bonded configuration is supported.

6.16. Transceiver Register Map

The transceiver register map provides a list of available PCS, PMA, EMIB and PLL addresses that are used in the reconfiguration process.

To avoid an illegal configuration, use the register map in conjunction with a transceiver configuration file generated by the Intel Stratix 10 Native PHY/Transmit PLL IP core. This configuration file includes details about the registers that are set for a specific transceiver configuration. Refer to a valid transceiver configuration file for legal register values and combinations.

Related Information

Logical View of the L-Tile/H-Tile Transceiver Registers on page 450

6.17. Reconfiguration Interface and Dynamic Revision History

Document Version	Changes
2020.03.03	Made the following change: • In *Multiple Reconfiguration Profiles*, clarified that fPLL in **Core** mode does not support the dynamic reconfiguration feature.
2019.10.02	Made the following change: • Simplified the `reconfig_clk` specification to "up to 150 MHz."
2019.05.08	Made the following change: • Removed note in *Reconfiguring Between GX and GXT Channels*.
2019.03.22	Made the following change: • Added steps to disable the re-enable background calibration in *Steps to Perform Dynamic Reconfiguration*, *Channel Reconfiguration*, and *PLL Reconfiguration*.
2018.07.06	Made the following changes: • Modified *Direct Reconfiguration Flow*, *Native PHY IP or PLL IP Core Guided Reconfiguration Flow*, and *Reconfiguration Flow* to reflect changes to *Steps to Perform Dynamic Reconfiguration*. • Added steps 5 to 13 to *Steps to Perform Dynamic Reconfiguration*. • Added clarification that you can dynamically reconfigure both channels and PLLs to *Interacting with the Reconfiguration Interface*.
2018.03.16	Made the following changes:

continued...

Document Version	Changes
	• Added topics "Channel Reconfiguration" and "PLL Reconfiguration". • Cases added to close timing for reconfiguration profiles. • Added a new diagram "Reconfigurable Interfaces". • Combined three topics (Configuration Files, Multiple Reconfiguration Profiles, Embedded Reconfiguration Profiles) in 1 "Topic Name TBD".
2017.06.06	Made the following changes: • Added a new table "`pre_reconfig` bit mapping" in "Steps to Dynamic Reconfiguration" topic. • Topic "Reconfiguring Between GX and GXT Channels" removed for this release. • Topic "Changing Analog PMA Settings" updated. • Example code added in "Native PHY Debug Master Endpoint (NPDME)" topic.
2016.12.21	Initial release

7. Calibration

Transceivers include both analog and digital blocks that require calibration to compensate for process, voltage, and temperature (PVT) variations. Intel Stratix 10 transceivers use hardened Precision Signal Integrity Calibration Engine (PreSICE) to perform calibration routines. Each transceiver tile has a PreSICE engine. If you are using more than 24 transceiver channels in your design, the transceiver calibrates on a per tile basis. This means that channel 0 in tile 1 and channel 0 in tile 2 are calibrated concurrently, etc.

Power-up Calibration, Background Calibration and User Recalibration are the main types of calibration.

- Power-up calibration occurs automatically at device power-up. It runs during device configuration.
- After you enable background calibration, background calibration continuously runs in the background.
- Use dynamic reconfiguration to trigger user recalibration. In this case, you are responsible for enabling the required calibration sequence.

Note: Only H-Tile production devices support background calibration.

When you upgrade from a previous version of the Intel Quartus Prime software to Intel Quartus Prime software version 18.1 or above, the Native PHY IP automatically upgrades any H-tile GXT design transceiver link operating at datarates ≥ 17.5 Gbps to **Enable background calibration**. Refer to *Background Calibration* for more information about specific requirements and how to control the background calibration.

Intel Stratix 10 devices use the OSC_CLK_1 pin to provide the transceiver calibration clock source. You must provide a 25, 100, or 125 MHz free running and stable clock to OSC_CLK_1.

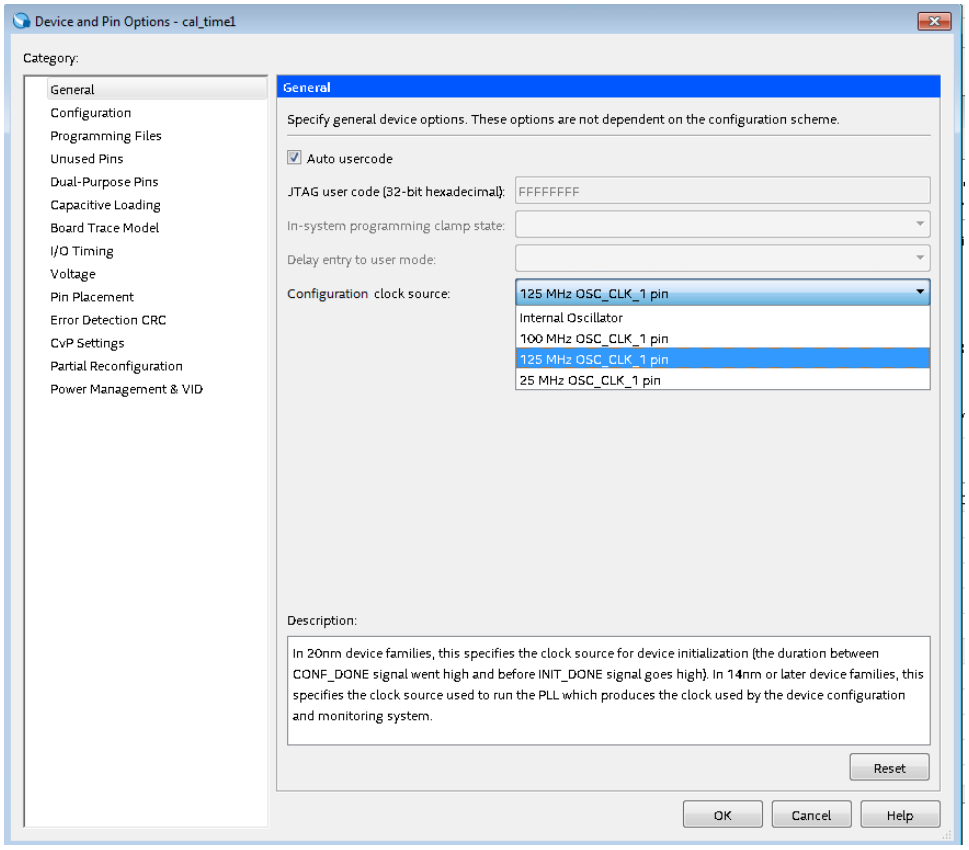
The FPGA's **Internal Oscillator** cannot be used for transceiver calibration. Do not select this clock source as the **Configuration clock source** in the Intel Quartus Prime settings.

In addition to providing this clock, you must also choose the appropriate frequency in **Quartus assignments**:

Quartus assignments ➤ Device ➤ Device and Pin Options ➤ Configuration clock source

1. Scroll down menu
2. Choose **25**, **100**, or **125** MHz OSC_CLK_1 pin option.

Figure 247. Calibration Clock Options



Specify general device options. These options are not dependent on the configuration scheme.

☑ Auto usercode

JTAG user code (32-bit hexadecimal): FFFFFFFF

In-system programming clamp state:

Delay entry to user mode:

Configuration clock source: 125 MHz OSC_CLK_1 pin

- Internal Oscillator
- 100 MHz OSC_CLK_1 pin
- 125 MHz OSC_CLK_1 pin
- 25 MHz OSC_CLK_1 pin

Description:

In 20nm device families, this specifies the clock source for device initialization (the duration between CONF_DONE signal went high and before INIT_DONE signal goes high). In 14nm or later device families, this specifies the clock source used to run the PLL which produces the clock used by the device configuration and monitoring system.

Note: You see the selected clock source in .qsf as follows:

```
set_global_assignment -name DEVICE_INITIALIZATION_CLOCK
OSC_CLK_1_125MHz
```

There is a PLL inside the FPGA that receives the clock from OSC_CLK_1 and provides a 250-MHz calibration clock to PreSICE. All reference clocks driving transceiver PLLs (ATX PLL, fPLL, CDR/CMU PLL) must have a stable frequency and be free running before the start of FPGA configuration (pull the FPGA's nCONFIG input high). For more information about OSC_CLK_1 pin requirements, refer to the *Intel Stratix 10 GX and SX Device Family Pin Connection Guidelines*.

Related Information

- Background Calibration on page 442
- Intel Stratix 10 GX and SX Device Family Pin Connection Guidelines

7.1. Reconfiguration Interface and Arbitration with PreSICE (Precision Signal Integrity Calibration Engine)

In Intel Stratix 10 devices, calibration is performed using the PreSICE. The PreSICE includes an Avalon memory-mapped interface to access the transceiver channel and PLL programmable registers. This Avalon memory-mapped interface includes a communication mechanism that enables you to request specific calibration sequences from the calibration controller.

The PreSICE Avalon memory-mapped interface and user Avalon memory-mapped interface reconfiguration both share an internal configuration bus. This bus is arbitrated to gain access to the transceiver channel and PLL programmable registers, and the calibration registers.

There are two ways to check which one has access to the internal configuration bus:

- Use the dynamic reconfiguration interface `reconfig_waitrequest`
- Use capability registers

The Native PHY IP core and PLL default setting is to use `reconfig_waitrequest`. When PreSICE controls the internal configuration bus, the `reconfig_waitrequest` from the internal configuration bus is high. When user access is granted, the `reconfig_waitrequest` signal from the internal configuration bus goes low.

To use the capability registers to check bus arbitration:

1. Select **Enable dynamic reconfiguration** from the **Dynamic Reconfiguration** tab in the PHY and PLL GUI.
2. Select both the **Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE** and **Enable control and status registers** options.

Reading the capability register 0x481[2] identifies what is controlling the channel access. Reading the capability register 0x480[2] identifies what is controlling the ATX and fPLL access.

Note: When **Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE** and **Enable control and status registers** are enabled, `reconfig_waitrequest` is not asserted high when PreSICE controls the internal configuration bus.

To return the internal configuration bus to PreSICE:

- **Write 0x1 to offset address 0x0[0] if any calibration bit is enabled from offset address 0x100.**
- **Write 0x3 to offset address 0x0[1:0] if no calibration bit has been enabled from offset address 0x100.**

To check if the calibration process is running, do one of the following:

- Monitor the `pll_cal_busy`, `tx_cal_busy`, and `rx_cal_busy` Native PHY output signals.
- Read the `tx/rx/pll_cal_busy` signal status from the capability registers.

The `tx/rx/pll_cal_busy` signals remain asserted as long as the calibration process is running. To check whether or not calibration is complete, you can read the capability registers or check the `tx/rx/pll_cal_busy` signals. The PMA `tx_cal_busy` and `rx_cal_busy` are from the same internal node, which cannot be separated from the hardware. The capability register 0x481[5:4] can enable or disable `tx_cal_busy` or `rx_cal_busy` individually. Using capability register 0x481[5:4] to isolate `tx_cal_busy` and `rx_cal_busy` is not supported in simplex TX and RX merging into a signal physical channel. See details in *Capability Registers*.

Related Information

- Capability Registers on page 437
- *Avalon Interface Specifications*

7.2. Calibration Registers

The Intel Stratix 10 transceiver PMA and PLLs include the following types of registers for calibration:

- Avalon memory-mapped interface arbitration registers—enables you to request internal configuration bus access
- Calibration enable registers—provide a convenient way to recalibrate the PMA or PLL
- Capability registers—provide calibration and arbitration status updates through the Avalon memory-mapped interface reconfiguration

7.2.1. Avalon Memory-Mapped Interface Arbitration Registers

Table 169. Avalon Memory-Mapped Interface Arbitration Registers

Bit	Offset Address	Description
[0]	0x0[49]	This bit arbitrates the control of the Avalon memory-mapped interface. • Set this bit to 0 to request control of the internal configuration bus by user. • Set this bit to 1 to pass the internal configuration bus control to PreSICE.
[1]	0x0	This bit indicates whether or not calibration is done. This is the inverted `cal_busy` signal. You can write to this bit; however, if you accidentally write 0x0 without enabling any calibration bit in 0x100, PreSICE may not set this bit to 0x1, and `cal_busy` remains high. Channel reset is triggered if `cal_busy` is connected to the reset controller. • 0x1 = calibration complete • 0x0 = calibration not complete. The `cal_busy` signal is activated two clock cycles after you write 0x0 to this bit.

Note: During calibration when `reconfig_waitrequest` is high, you can not read offset address 0x0.

7.2.2. User Recalibration Enable Registers

(49) The transceiver channel, ATX PLL, and fPLL use the same offset address.

7.2.2.1. Transceiver Channel Calibration Registers

Table 170. Transceiver Channel PMA Calibration Enable Registers

Bit	PMA Calibration Enable Register Offset Address 0x100
0	PMA RX calibration enable[50] Set to 1 to enable calibration.
1	PMA TX calibration enable Set to 1 to enable calibration.

7.2.2.2. ATX PLL/fPLL/CMU PLL Calibration Registers

During calibration when PreSICE is controlling the internal configuration bus, you cannot read from or write to calibration enable registers.

To enable calibration, you must perform a read-modify-write on offset address 0x100.

1. Read the offset address 0x100.
2. Keep the value from MSB[7:1] and set LSB[0] to 1.
3. Write the new value to the offset address 0x100.

Table 171. ATX PLL/fPLL/CMU PLL Calibration Enable Registers

Bit	TX PLL Calibration Enable Register Offset Address 0x100
0	Set to 1 to enable ATX PLL calibration. Set to 1 to enable CMU PLL calibration.
1	Set to 1 to enable fPLL calibration.

7.2.3. Capability Registers

Capability registers allow you to read calibration status through Avalon memory-mapped interface reconfiguration. They are soft logic and reside in the FPGA fabric.

Reading capability registers does not require bus arbitration. You can read them during the calibration process.

To use capability registers to check calibration status, you must enable the capability registers when generating the Native PHY or PLL IP cores. To enable the capability registers, select the **Enable capability registers** option in the **Dynamic Reconfiguration** tab.

The `tx_cal_busy` and `rx_cal_busy` signals from the hard PHY are from the same hardware and change the state (high/low) concurrently during calibration. The register bits 0x481[5:4] are defined to solve this issue. This prevents a TX channel being affected by RX calibration, or an RX channel being affected by TX calibration. If you want `rx_cal_busy` unchanged during the TX calibration, you must set 0x481[5] to 0x0 before returning the bus to PreSICE. The channel RX is not reset due to the TX calibration. If you want`tx_cal_busy` unchanged during the RX calibration, you must set 0x481[4] to 0x0 before returning the bus to PreSICE. The channel TX is not reset due to the RX calibration. If you accidentally write 0x00 to 0x481[5:4], the `tx_cal_busy` or `rx_cal_busy` output ports are never activated to high. Neither of

(50) CDR and CMU PLL calibration are part of RX PMA calibration.

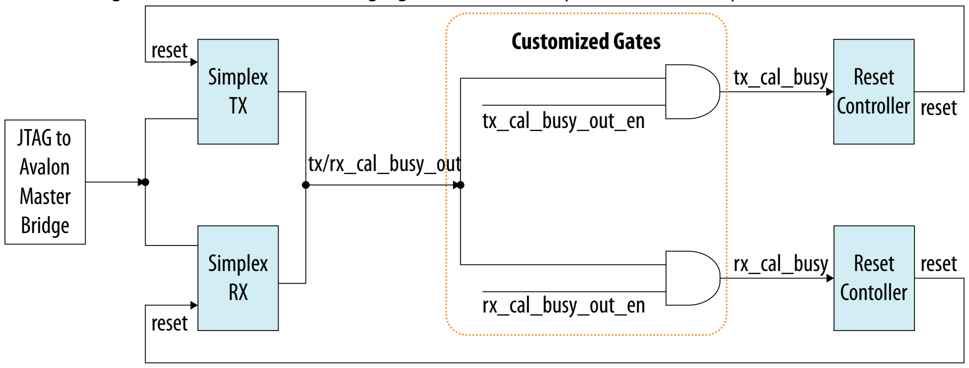
the 0x481[1:0] registers assert either. This feature cannot be enabled when channel merging is involved or when a TX simplex and a RX simplex are merged into a single physical channel.

Note: When you merge a TX Simplex and a RX Simplex into the same physical channel, you cannot enable the background calibration feature.

Rules to Build Customized Gating Logic to Separate tx_cal_busy and rx_cal_busy signals

Figure 248. An Example of an AND Gate used as Customized Logic

The customized gates shown in the following figure are an example and not a unique solution



The capability register is not available for merging a Simplex TX and a Simplex RX signal into the same physical channel. The `tx_cal_busy_out` and `rx_cal_busy_out` signals share the same port. So, you should build customized gating logic to separate them.

- The `tx_cal_busy_out_en` signal enables the `tx_cal_busy` output.
- The `rx_cal_busy_out_en` signal enables the `rx_cal_busy` output.
- At power up, `tx_cal_busy_out_en` and `rx_cal_busy_out_en` should be set to "1".
- At normal operation:
  - When the RX is calibrating, setting `tx_cal_busy_out_en` to "0" and `rx_cal_busy_out_en` to "1" disables tx_cal_busy, so the TX does not reset while RX is calibrating.
  - When the TX is calibrating, setting `rx_cal_busy_out_en` to "0" and `tx_cal_busy_out_en` to "1" disables `rx_cal_busy`, so the RX does not reset while TX is calibrating.

You can use the PMA 0x481[2] register to check bus arbitration through Avalon memory-mapped interface reconfiguration. This feature is available whether or not **Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE** is enabled in the **Dynamic Reconfiguration** tab. The ATX PLL and fPLL use the 0x480[2] register for bus arbitration status.

Table 172. PMA Capability Registers for Calibration Status

Bit	Description
0x481[5]	PMA channel `rx_cal_busy` output enable.[51] The power up default value is 0x1. 0x1: The `rx_cal_busy` output and 0x481[1] are asserted high whenever PMA TX or RX calibration is running. 0x0: The `rx_cal_busy` output or 0x481[1] are never asserted high.
0x481[4]	PMA channel `tx_cal_busy` output enable. The power up default value is 0x1. 0x1: The `tx_cal_busy` output and 0x481[0] are asserted high whenever PMA TX or RX calibration is running. 0x0: The `tx_cal_busy` output or 0x481[0] are never asserted high.
0x481[2]	PreSICE Avalon memory-mapped interface control. This register is available to check who controls the bus, no matter if, separate `reconfig_waitrequest` from the status of Avalon memory-mapped interface arbitration with PreSICE is enabled or not. 0x1: PreSICE is controlling the internal configuration bus. 0x0: The user has control of the internal configuration bus.
0x481[1]	PMA channel `rx_cal_busy` active high 0x1: PMA RX calibration is running 0x0: PMA RX calibration is done
0x481[0]	PMA channel `tx_cal_busy` active high 0x1: PMA TX calibration is running 0x0: PMA TX calibration is done

Table 173. ATX PLL Capability Registers for Calibration Status

Bit	Description
0x480[2]	PreSICE Avalon memory-mapped interface control. This register is available to check who controls the bus, no matter if, separate `reconfig_waitrequest` from the status of Avalon memory-mapped interface arbitration with PreSICE is enabled or not. 0x1: PreSICE is controlling the internal configuration bus. 0x0: The user has control of the internal configuration bus.
0x480[1]	ATX PLL `pll_cal_busy` 0x1: ATX PLL calibration is running 0x0: ATX PLL calibration is done

Table 174. fPLL Capability Registers for Calibration Status

Bit	Description
0x480[2]	PreSICE Avalon memory-mapped interface control 0x1: PreSICE is controlling the internal configuration bus. This register is available to check who controls the bus, no matter if, separate `reconfig_waitrequest` from the status of Avalon memory-mapped interface arbitration with PreSICE is enabled or not. 0x0: The user has control of the internal configuration bus.
0x480[1]	fPLL `pll_cal_busy` 0x1: fPLL calibration is running 0x0: fPLL calibration is done

[51] CDR and CMU PLL calibration are part of RX PMA calibration.

Table 175. CMU PLL Capability Registers for Calibration Status

Bit	Description
0x480[2]	PreSICE Avalon memory-mapped interface control. This register is available to check who controls the bus, no matter if, separate `reconfig_waitrequest` from the status of Avalon memory-mapped interface arbitration with PreSICE is enabled or not. 0x1: PreSICE is controlling the internal configuration bus. 0x0: The user has control of the internal configuration bus.
0x480[1]	CMU PLL `pll_cal_busy` 0x1: CMU PLL calibration is running 0x0: CMU PLL calibration is done

7.2.4. Rate Switch Flag Register

The rate switch flag is for clock data recovery (CDR) charge pump calibration. Each SOF has CDR default charge pump settings. After power up, these settings are loaded into the PreSICE memory space. If you stream in a whole new memory initialization file (`.MIF`), the charge pump settings are stored into the Avalon memory-mapped interface reconfiguration space. During RX PMA calibration (including CDR), PreSICE needs to know which set of CDR charge pump setting to use.

The default value of 0x100[3] is 0x0, PreSICE uses the settings in its memory space. If after a rate change you set 0x100[3]=0x1, PreSICE uses the setting from the Avalon memory-mapped interface reconfiguration register. The rate switch flag only tells PreSICE where to obtain the CDR charge pump settings for CDR calibration. The rate switch flag should be used only when there is a rate change.

Multiple MIF files are required for rate change and reconfiguration. When the CDR charge pump setting registers 0x139[7] and 0x133[7:5] in the new MIF you want to stream in are different from the previous MIF, you must recalibrate with 0x100[3] = 0x1. If you stream in the whole MIF, the 0x100[3] is set to the correct value inside the MIF. If you stream in a reduced MIF, you must check whether or not CDR charge pump setting registers 0x139[7] and 0x133[7:5] are inside the reduced MIF. If the reduced MIF has these updated registers, you must set register 0x100[3]=0x1. If the reduced MIF does not include these updated registers, you need to set 0x100[3]=0x0.

Table 176. Rate Switch Flag Register for CDR Calibration

Bit	Description
0x100[3]	Rate switch flag register. Power up default value is 0x0. 0x0, PreSICE uses the default CDR charge pump bandwidth from the default memory space. 0x1, PreSICE uses the CDR charge pump bandwidth setting from the Avalon memory-mapped interface space register space.

Related Information

Steps to Perform Dynamic Reconfiguration on page 405
When you perform dynamic reconfiguration to CDR or CMU PLL, please refer to this topic.

7.3. Power-up Calibration

After the device is powered up, PreSICE automatically initiates the calibration process during device programming. The time required to complete the calibration process after device power-up can vary by device. All `tx/rx/pll_cal_busy` signals are high after device power up. These `tx/rx/pll_cal_busy` signals are de-asserted by PreSICE at the completion of the calibration process. You must ensure that the transceiver reset sequence in your design waits for the calibration to complete before resetting the transceiver PLLs and the transceiver channels.

The PreSICE may still control the internal configuration bus even after power-up calibration is complete. Intel recommends that you wait until all `tx/rx/pll_cal_busy` signals are low before requesting any access.

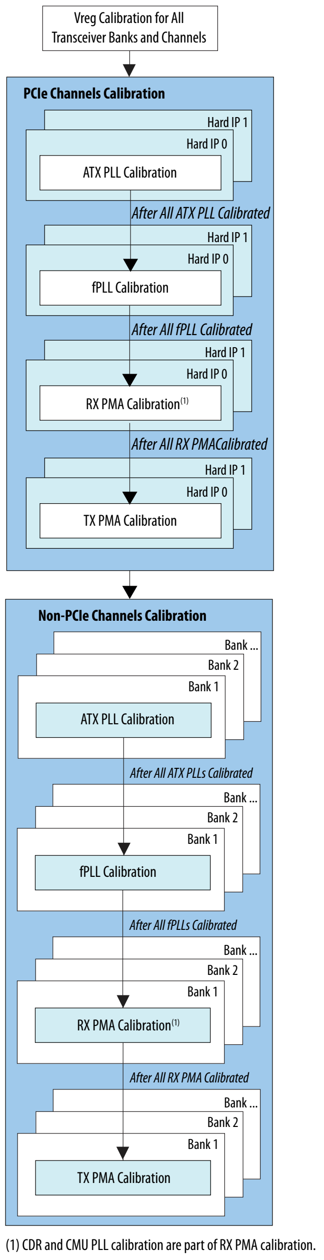
All power-up calibration starts from the clock network regulator and voltage regulator calibration for all banks and channels.

For applications using both PCIe Hard IP and non-PCIe channels, the power-up calibration sequence is:

1. Clock network calibration for all tiles.
2. Voltage regulator calibration for all banks, channels, and PLLs.
3. Wait for PCIe reference clocks to toggle.
4. PCIe HIP0 calibration (if used).
5. PCIe HIP1 calibration (if used).
6. Calibration of all non-PCIe Hard IP channels in the calibration sequence.

Note: For successful device configuration and proper power-up calibration for your PCIe link, you must ensure that the PCIe reference clock is available and stable during power-up. User recalibration for PCIe links is not supported.

Figure 249. Power-up Calibration Sequence for PCIe Hard IP and non-PCIe Channels


Vreg Calibration for All Transceiver Banks and Channels
PCIe Channels Calibration
Hard IP 1
Hard IP 0
ATX PLL Calibration
After All ATX PLL Calibrated
fPLL Calibration
After All fPLL Calibrated
RX PMA Calibration(1)
After All RX PMACalibrated
TX PMA Calibration
Non-PCIe Channels Calibration
Bank ...
Bank 2
Bank 1
ATX PLL Calibration
After All ATX PLLs Calibrated
fPLL Calibration
After All fPLLs Calibrated
RX PMA Calibration(1)
After All RX PMA Calibrated
TX PMA Calibration
(1) CDR and CMU PLL calibration are part of RX PMA calibration.

7.4. Background Calibration

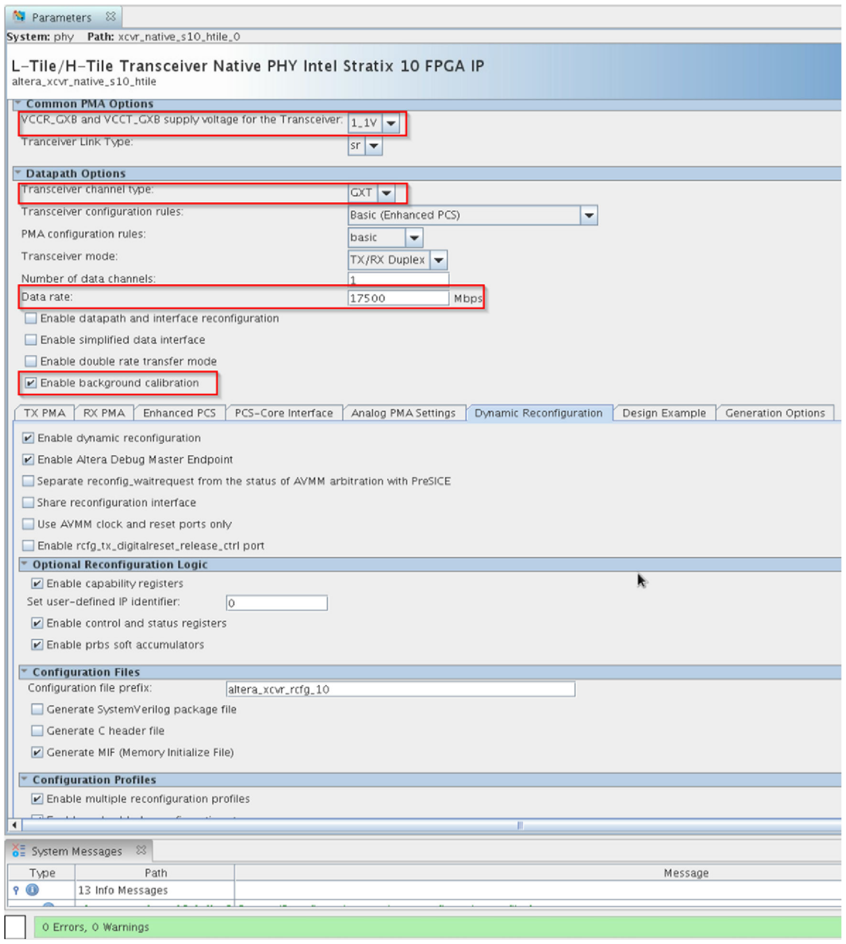
For better performance of applications running at datarates equal to or greater than 17.5 Gbps, enable background calibration in the Native PHY IP.

Note: Only H-Tile production devices support background calibration.

When you upgrade from a previous version of the Intel Quartus Prime software to Intel Quartus Prime software version 18.1 or later, the Native PHY IP automatically enables the **Enable background calibration** feature.

1. Turn on **Enable background calibration**.

Figure 250. Enable Background Calibration



Parameters

System: phy Path: xcvr_native_s10_htile_0

L-Tile/H-Tile Transceiver Native PHY Intel Stratix 10 FPGA IP
altera_xcvr_native_s10_htile

Common PMA Options
VCCR_GXB and VCCT_GXB supply voltage for the Transceiver: 1_1V
Tranceiver Link Type: sr

Datapath Options
Transceiver channel type: GXT
Transceiver configuration rules: Basic (Enhanced PCS)
PMA configuration rules: basic
Transceiver mode: TX/RX Duplex
Number of data channels: 1
Data rate: 17500 Mbps
☐ Enable datapath and interface reconfiguration
☐ Enable simplified data interface
☐ Enable double rate transfer mode
☑ Enable background calibration

TX PMA | RX PMA | Enhanced PCS | PCS-Core Interface | Analog PMA Settings | Dynamic Reconfiguration | Design Example | Generation Options

☑ Enable dynamic reconfiguration
☑ Enable Altera Debug Master Endpoint
☐ Separate reconfig_waitrequest from the status of AVMM arbitration with PreSICE
☐ Share reconfiguration interface
☐ Use AVMM clock and reset ports only
☐ Enable rcfg_tx_digitalreset_release_ctrl port

Optional Reconfiguration Logic
☑ Enable capability registers
Set user-defined IP identifier: 0
☑ Enable control and status registers
☑ Enable prbs soft accumulators

Configuration Files
Configuration file prefix: altera_xcvr_rcfg_10
☐ Generate SystemVerilog package file
☐ Generate C header file
☑ Generate MIF (Memory Initialize File)

Configuration Profiles
☑ Enable multiple reconfiguration profiles

System Messages

Type	Path	Message
	13 Info Messages	

0 Errors, 0 Warnings

Background calibration is always running, so PreSICE always controls Avalon memory-mapped interface.

2. If you dynamically reconfigure the GX rate to GXT rate, choose **1_1v** for **VCCR_GXB and VCCT_GXB supply voltage for the Transceiver** even when you are at the GX rate.

3. If you turn on the **Share reconfiguration interface** option for a multichannel PHY configuration, you must disable background calibration by setting 0 to 0x542[0] for every channel to get Avalon memory-mapped interface access. For example, if the **Share reconfiguration Interface** option is enabled for three transceiver channels, you must disable the background calibration for all three channels before you have access to the Avalon memory-mapped interface on these channels.

7.5. User Recalibration

Power up calibration automatically calibrates all PLLs and transceiver channels used in your application. User recalibration is required if the following conditions are met:

- During device power up, OSC_CLK_1 is asserted and running stable, but the transceiver reference clock remains de-asserted until after the power up process is complete.
- During device power up, OSC_CLK_1 and the transceiver reference clock are asserted and running stable. When the device power up process is complete, the transceiver reference clock changes frequency. When this happens, either the transceiver reference clock could become unstable or your application requires a different transceiver reference clock during normal operation, which could cause a data rate change.
- After device power up in normal operation, you reconfigure the transceiver data rate.
- If you use the CDR CMU as a TX PLL, you must recalibrate the PMA TX of the channel which uses the CDR CMU as a TX PLL.
- If you recalibrate the TX PLL due to an unstable reference clock during power up calibration, you must recalibrate the PMA TX after TX PLL recalibration.
- If the TX PLL and CDR share the same reference clock which is unstable during power up calibration, you must recalibrate the TX PLL, PMA TX and PMA RX. The PMA RX calibration includes CDR calibration.
- Recalibrate the fPLL if the fPLL is connected as a second PLL (downstream cascaded PLL). This is important especially if the first PLL output clock is not stable.

You must also reset the transceivers after performing a user recalibration. For example, if you perform a data rate auto-negotiation that involves PLL reconfiguration and PLL and channel interface switching, then you must reset the transceivers.

The proper reset sequence is required after calibration. Intel recommends you use the Intel Stratix 10 Transceiver Reset Controller IP which has `tx_cal_busy` and `rx_cal_busy` inputs and follow Intel's recommended reset sequence. You need to connect `tx_cal_busy` and `rx_cal_busy` from the Native PHY IP core outputs to the reset controller inputs in your design. Reset upon calibration is automatically processed when you perform user recalibration.

You can initiate the recalibration process by writing to the specific recalibration registers. You must enable capability registers when generating the Native PHY IP or PLL IP cores to have access to the 0x480 or 0x481 registers.

Related Information

Recommended Reset Sequence on page 320

7.5.1. Recalibrating a Duplex Channel (Both PMA TX and PMA RX)

Note: Address refers to the channel offset address.

Disable background calibration if you have it on.

1. Write 0x0 to channel offset address 0x542[0] to disable background calibration.

You have control if 0x542[0] = 0x0 or 0x481[2] = 0x0 or if `reconfig_waitrequest` is low.

Recalibrating

2. Perform a RMW operation on 0x03 with mask 0x03 to address 0x100. This sets the PMA TX and PMA RX calibration enable bit.
3. Write 0x01 to address 0x00. This lets the PreSICE perform the calibration.
4. Loop read 0x481[1:0] until you see both bits become 0x0.
   - PMA RX calibration is complete when 0x481[1] = 0x0.
   - PMA TX calibration is complete when 0x481[0] = 0x0.

Enable background calibration.

5. Write 0x1 to channel offset address 0x542[0].

7.5.2. Recalibrating the PMA RX Only in a Duplex Channel

Note: Address refers to the channel offset address.

Disable background calibration if you have it on.

1. Write 0x0 to channel offset address 0x542[0] to disable background calibration.

   You have control if 0x542[0] = 0x0 or 0x481[2] = 0x0 or if `reconfig_waitrequest` is low.

Recalibrating

2. Perform a RMW operation on 0x00 with mask 0x10 to address 0x481. This sets bit 4 to mask out `tx_cal_busy`.
3. Perform a RMW operation on 0x01 with mask 0x01 to address 0x100. This sets the PMA RX calibration enable bit.
4. Write 0x01 to address 0x00. This lets the PreSICE perform the calibration.
5. Loop read 0x481[1] until you see it become 0x0.
   - PMA RX calibration is complete when 0x481[1] = 0x0.
6. Perform a RMW operation on 0x10 to address 0x481. This sets bit 4 to 0x1 to enable `tx_cal_busy`.

Enable background calibration

7. Write 0x1 to channel offset address 0x542[0].

7.5.3. Recalibrating the PMA TX Only in a Duplex Channel

Note: Address refers to the channel offset address.

Disable background calibration if you have it on

1. Write 0x0 to channel offset address 0x542[0] to disable background calibration.

You have control if 0x542[0] = 0x0 or 0x481[2] = 0x0 or if `reconfig_waitrequest` is low.

Recalibrating

2. Perform a RMW operation on 0x00 with mask 0x20 to address 0x481. This sets bit 5 to 0 to mask out `rx_cal_busy`.
3. Perform a RMW operation on 0x02 with mask 0x02 to address 0x100. This sets the PMA TX calibration enable bit.
4. Write 0x01 to address 0x00. This lets the PreSICE perform the calibration.
5. Loop read 0x481[0] until you see it become 0x0.
   - PMA TX calibration is complete when 0x481[0] = 0x0.
6. Perform a RMW operation on 0x20 to address 0x481. This sets bit 5 to 0x1 to enable `rx_cal_busy`.

Enable background calibration

7. Write 0x1 to channel offset address 0x542[0].

7.5.4. Recalibrating a PMA Simplex RX Without a Simplex TX Merged into the Same Physical Channel

Note: Address refers to the simplex RX channel offset address.

Disable background calibration if you have it on.

1. Write 0x0 to channel offset address 0x542[0] to disable background calibration.

   You have control if 0x542[0] = 0x0 or 0x481[2] = 0x0 or if `reconfig_waitrequest` is low.

Recalibrating

2. Perform a RMW operation on 0x00 with mask 0x10 to address 0x481. This sets bit 4 to 0 to mask out `tx_cal_busy`.
3. Perform a RMW operation on 0x01 with mask 0x01 to address 0x100. This sets the PMA RX calibration enable bit.
4. Write 0x01 to address 0x00. This lets the PreSICE perform the calibration.
5. Loop read 0x481[1] until you see it become 0x0.
   - PMA RX calibration is complete when 0x481[1] = 0x0.
6. Perform a RMW operation on 0x10 to address 0x481. This sets bit 4 to 0x1 to enable `tx_cal_busy`.

Enable background calibration.

7. Write 0x1 to channel offset address 0x542[0].

7.5.5. Recalibrating a PMA Simplex TX Without a Simplex RX Merged into the Same Physical Channel

Note: Address refers to the simplex TX channel offset address.

Disable background calibration if you have it on

1. Write 0x0 to channel offset address 0x542[0] to turn background calibration off.

You have control if 0x542[0] = 0x0 or 0x481[2] = 0x0 or if `reconfig_waitrequest` is low.

Recalibrating

2. Perform a RMW operation on 0x00 with mask 0x20 to address 0x481. This sets bit 5 to 0 to mask out `rx_cal_busy`.
3. Perform a RMW operation on 0x02 with mask 0x02 to address 0x100. This sets the PMA TX calibration enable bit.
4. Write 0x01 to address 0x00. This lets the PreSICE perform the calibration.
5. Loop read 0x481[0] until you see it become 0x0.
    - PMA TX calibration is complete when 0x481[0] = 0x0.
6. Perform a RMW operation on 0x20 with mask 0x20 to address 0x481. This sets bit 5 to 0x1 to enable `rx_cal_busy`.

Enable background calibration

7. Write 0x1 to channel offset address 0x542[0].

7.5.6. Recalibrating Only a PMA Simplex RX in a Simplex TX Merged Physical Channel

Note: Address refers to the simplex RX channel offset address. When you merge a TX simplex and a RX simplex into the same physical channel, you cannot enable the background calibration feature.

1. Set your design to mask out `tx_cal_busy`.
2. Perform a RMW operation on 0x01 with mask 0x01 to address 0x100. This sets the PMA RX calibration enable bit.
3. Write 0x01 to address 0x00. This lets the PreSICE perform the calibration. This activates `rx_cal_busy` and `reconfig_waitrequest` both high.
4. Loop check `rx_cal_busy` and `reconfig_waitrequest`.
    - PMA RX calibration is complete when `rx_cal_busy` is low.
    - PreSICE returns control to you when `reconfig_waitrequest` is low.
5. Set your design to remove the `tx_cal_busy` mask.

7.5.7. Recalibrating Only a PMA Simplex TX in a Simplex RX Merged Physical Channel

Note: Address refers to the simplex TX channel offset address. When you merge a TX simplex and a RX simplex into the same physical channel, you cannot enable the background calibration feature.

1. Set your design to mask out `rx_cal_busy`.
2. Perform a RMW operation on 0x02 with mask 0x02 to address 0x100. This sets the PMA TX calibration enable bit.
3. Write 0x01 to address 0x00. This lets the PreSICE perform the calibration. This activates `tx_cal_busy` and `reconfig_waitrequest` both high.
4. Loop check `rx_cal_busy` and `reconfig_waitrequest`.

- PMA RX calibration is complete when `rx_cal_busy` is low.
- PreSICE returns control to you when `reconfig_waitrequest` is low.

5. Set your design to remove the `rx_cal_busy` mask.

7.5.8. Recalibrating the fPLL

Note: Address refers to the fPLL offset address.

1. Perform a RMW operation on 0x02 with mask 0x02 to address 0x100. This sets the fPLL calibration enable bit.
2. Write 0x01 to address 0x00. This lets the PreSICE perform the calibration.
3. Loop read 0x480[1] until you see it become 0x0.
   fPLL calibration is complete when 0x480[1] = 0x0.

7.5.9. Recalibrating the ATX PLL

Note: Address refers to the ATX PLL offset address.

1. Perform a RMW operation on 0x01 with mask 0x01 to address 0x100. This sets the ATX PLL calibration enable bit.
2. Write 0x01 to address 0x00. This lets the PreSICE perform the calibration.
3. Loop read 0x480[1] until you see it become 0x0.
   ATX PLL calibration is complete when 0x480[1] = 0x0.

7.5.10. Recalibrating the CMU PLL When it is Used as a TX PLL

Note: Address refers to the simplex RX channel offset address.

Disable background calibration if you have it on.

1. Write 0x0 to channel offset address 0x542[0] to disable background calibration. You have control if 0x542[0] = 0x0 or 0x481[2] = 0x0 or if `reconfig_waitrequest` is low.

Recalibrating

2. Perform a RMW operation on 0x01 with mask 0x01 to address 0x100. This sets the CMU PLL calibration enable bits.
3. Write 0x01 to address 0x00. This lets the PreSICE perform the calibration.
4. Loop read 0x480[1] until you see the bit become 0x0.
   - CMU PLL calibration is complete when 0x480[1] = 0x0.

Enable background calibration.

5. Write 0x1 to channel offset address 0x542[0].

7.6. Calibration Revision History

Document Version	Changes
2020.03.03	Made the following change: • Added "for successful device configuration" to the PCIe note in *Power-up Calibration*.
2019.10.25	Made the following change: • Added the requirement that: The FPGA's internal oscillator cannot be used for transceiver calibration. Do not select this clock source as the **Configuration clock source** in the Intel Quartus Prime settings.
2019.03.22	Made the following changes: • Removed the first two steps in *Recalibrating Only a PMA Simplex RX in a Simplex TX Merged Physical Channel*, *Recalibrating Only a PMA Simplex TX in a Simplex RX Merged Physical Channel*, *Recalibrating the fPLL*, *Recalibrating the ATX PLL*, and *Recalibrating the CMU PLL when it is Used as a TX PLL*. • Added the 0x481[2] = 0x0 option to confirm control of calibration to *Recalibrating a Duplex Channel (Both PMA TX and PMA RX)*, *Recalibrating the PMA RX Only in a Duplex Channel*, *Recalibrating the PMA TX Only in a Duplex Channel*, *Recalibrating a PMA Simplex RX without a Simplex TX Merged into the Same Physical Channel*, *Recalibrating a PMA Simplex TX without a Simplex RX Merged into the Same Physical Channel*, and *Recalibrating the CMU PLL When it is Used as a TX PLL*.
2018.10.05	Made the following changes: • Added *Background Calibration*. • In *User Recalibration* sections, added instructions for turning background calibration on and off. • Added a note to the *Power-up Calibration* section. • Updated the note in *Avalon Memory-Mapped Interface Arbitration Registers*. • Added a note to the *Capability Registers* section. • Added a note to the *Calibration* section. • Added the "CMU PLL Capability Registers for Calibration Status" table. • Updated the steps in the "Recalibrating the CMU PLL when it is Used as a TX PLL" section.
2018.07.06	Made the following changes: • Deleted bit [1] from the *Avalon Memory-Mapped Interface Arbitration Registers*. • Updated the instructions to return the internal configuration bus to PreSICE in *Reconfiguration Interface and Arbitration with PreSICE (Precision Signal Integrity Calibration Engine)*. • Changed the calibration clock the PLL provides from a 200 to a 250-MHz in *Calibration*. • Added PMA TX calibration to *Recalibrating the CMU PLL when it is Used as a TX PLL*. • Added clarification: Each transceiver tile has a PreSICE engine. If you are using more than 24 transceiver channels in your design, the transceiver calibrates on a per tile basis. This means that channel 0 in tile 1 and channel 0 in tile 2 are calibrated concurrently, etc. • Added a footnote to the "PMA Capability Registers for Calibration Status" and "Transceiver Channel PMA Calibration Enable Registers" tables: CDR and CMU PLL calibration are part of RX PMA calibration. • Changed "Device initialization clock source" to "Configuration clock source" in *Calibration* and the "Calibration Clock Options" figure. • Make it even more obvious that the clocks must be present and at the correct frequency. • Removed duplicate paragraphs in *User Recalibration*. • Deleted the "Power-up Calibration Sequence for Non-PCIe Hard IP (HIP) Channels" figure. • Changed the offset address for returning the internal configuration bus to PreSICE.
2017.02.17	Made the following changes: • Updated the value of the capability register from 0x281 to 0x481 and 0x280 to 0x480 respectively. • Updated the Power-up Calibration sequence for non-PCIe and PCIe HIP channels. • Updated the use conditions to recalibrate the transceivers or the PLLs.
2016.12.21	Initial release

A. Logical View of the L-Tile/H-Tile Transceiver Registers

The term, logical view, refers to the various transceiver blocks categorized by their function. For each function, the relevant register tables list the registers according to the sequence you must use to configure them.

A.1. ATX_PLL Logical Register Map

Stratix 10 ATX_PLL Register Map Summary

Feature	Feature Description
ATX PLL Calibration	ATX PLL Calibration allows users to optimize the ATX PLL performance when changing data rates.
Optional Reconfiguration Logic ATX PLL- Capability	Enables ATX PLL capabilities to be readable.
Optional Reconfiguration Logic ATX PLL- Control & Status	Enables users to read the status of ATX PLL functions and reset the ATX PLL.
Embedded Streamer (ATX PLL)	Enables logic in ATX PLL to store the individual profile information and perform .mif streaming for reconfiguration application.

A.1.1. ATX PLL Calibration

ATX PLL Calibration allows users to optimize the ATX PLL performance when changing data rates.

Name	Address	Type	Attribute Name	Encodings
Internal configuration bus arbitration register for ATX PLL	0x000[0]	read-write	pcs_arbiter_ctrl	This bit arbitrates the control of internal configuration bus Write 1'b0 to control the internal configuration bus. Write 1'b1 to pass the internal configuration bus control to PreSICE.
ATX PLL Calibration Status	0x000[1]	read-write	pcs_cal_done	Status for calibration done or not done This is the inverted cal_busy signal. 1'b1: calibration done 1'b0: calibration not done
ATX PLL calibration enable	0x100[0]	read-write	lc_calibration	ATX PLL calibration enable 1'b1: ATX PLL calibration enable on 1'b0: ATX PLL calibration enable off
Request PreSICE to configure the ATX PLL in preparation for reconfiguration	0x100[1]	read-write	pre_reconfig	Request PreSICE to configure the ATX PLL in preparation for reconfiguration: 1'b1: Request PreSICE to configure the PLL in reconfiguration mode 1'b0: Reconfiguration mode not requested

A.1.2. Optional Reconfiguration Logic ATX PLL- Capability

Enables ATX PLL capabilities to be readable.

Name	Address	Type	Attribute Name	Encodings

IP Identifier	`0x400[7:0]`
	`0x401[7:0]`
	`0x402[7:0]`
	`0x403[7:0]`
		`read-only`	`atx_address_id`	Unique identifier for the PLL instance.

Status Register Enabled	`0x404[0]`	`read-only`	`atx_status_register_enable`	Indicates if the status registers have been enabled. 1'b1 indicates the feature is enabled.
Control Register Enabled	`0x405[0]`	`read-only`	`atx_control_register_enable`	Indicates if the control registers have been enabled. 1'b1 indicates the feature is enabled.
Master CGB enabled	`0x410[0]`	`read-only`	`atx_mcgb_enable`	Indicates if the Master CGB is enabled. 1'b1 indicates the master CGB is enabled.

A.1.3. Optional Reconfiguration Logic ATX PLL- Control & Status

Enables users to read the status of ATX PLL functions and reset the ATX PLL.

Name	Address	Type	Attribute Name	Encodings
PLL Locked Status	`0x480[0]`	`read-only`	`atx_pll_locked`	Indicates if the ATX PLL is locked. 1'b1 indicates the ATX PLL is locked.
PLL Calibration Busy Status	`0x480[1]`	`read-only`	`atx_cal_busy`	Indicates the ATX calibration status. 1'b1 indicates the ATX is currently being calibrated.
Avalon memory-mapped interface Bus Busy Status	`0x480[2]`	`read-only`	`atx_avmm_busy`	Shows the status of internal configuration bus arbitration. When set to 1'b1, PreSICE has control of the internal configuration bus. When set to 1'b0, you have control of the internal configuration bus. Refer to the *Arbitration* section for more details.
PLL Power-down	`0x4E0[0]`	`read-write`	`atx_pll_powerdown`	Drives the PLL power-down when the Override is set
Override PLL Power-down	`0x4E0[1]`	`read-write`	`atx_override_pll_powerdown`	Ensures the PLL listens to the NPDME pll_powerdown register. 1'b1 indicates the receiver listens to the NPDME `pll_powerdown`.

A.1.4. Embedded Streamer (ATX PLL)

Enables logic in ATX PLL to store the individual profile information and perform .mif streaming for dynamic reconfiguration.

Name	Address	Type	Attribute Name	Encodings
Configuration profile select	`0x540[2:0]`	`read-write`	`atx_cfg_sel`	Binary encoding of the configuration profile to stream Profile 0: 3'b000 Profile 1: 3'b001

continued...

Name	Address	Type	Attribute Name	Encodings
				Profile 2: 3'b010 Profile 3: 3'b011 Profile 4: 3'b100 Profile 5: 3'b101 Profile 6: 3'b110 Profile 7: 3'b111
Start streaming	0x540[7]	`read-write`	`atx_cfg_load`	Set to 1'b1 to initiate streaming, self-clearing bit
Busy status bit	0x541[0]	`read-only`	`atx_rcfg_busy`	Bit is set to: 1'b1: streaming is in progress 1'b0: streaming is complete

A.2. CMU_PLL Logical Register Map

Stratix 10 CMU_PLL Register Map Summary

Feature	Feature Description
CDR/CMU and PMA Calibration	Enables user to optimize CDR/CMU and PMA performance when changing the data rate and reference clock.
Optional Reconfiguration Logic CMU PLL- Capability	Enables CMU PLL capabilities to be readable.
Optional Reconfiguration Logic CMU PLL- Control & Status	Enables users to read the status of CMU PLL functions and reset the CMU PLL.
Embedded Streamer (CMU PLL)	Enables logic in CMU PLL to store the individual profile information and perform streaming.

A.2.1. CDR/CMU and PMA Calibration

Enables user to optimize CDR/CMU and PMA performance when changing the data rate, reference clock, or both.

Name	Address	Type	Attribute Name	Encodings
Internal configuration bus arbitration register	0x000[0]	`read-write`	`pcs_arbiter_ctrl`	This bit arbitrates the control of internal configuration bus Write 1'b0 to control the internal configuration bus by user. Write 1'b1 to pass the internal configuration bus control to PreSICE.
PMA Calibration Status	0x000[1]	`read-write`	`pcs_cal_done`	Status for calibration done or not done. This is an OR operation, and if both TX and RX calibration are enabled, PreSICE calibrates RX first, then TX. After both have completed, the calibration is complete. This is inverted `cal_busy` signal. 1'b1: calibration done 1'b0: calibration not done
PMA RX calibration enable	0x100[0]	`read-write`	`pm_cr2_tx_rx_uc_rx_cal`	PMA RX calibration enable 1'b1: RX calibration enable on 1'b0: RX calibration enable off

continued...

Send Feedback

Name	Address	Type	Attribute Name	Encodings
PMA TX calibration enable	0x100[1]	read-write	uc_tx_cal	PMA TX calibration enable 1'b1: TX calibration enable on 1'b0: TX calibration enable off
Request PreSICE to configure the CDR/CMU PLL in preparation for reconfiguration	0x100[3]	read-write	pre_reconfig	Rate switch flag register when you use the PMA as a RX channel. The power up default value is 0x0. 0x0: PreSICE uses the default CDR charge pump bandwidth from the default memory space. 0x1: PreSICE uses the CDR charge pump bandwidth setting from the Avalon memory-mapped interface space register space. Request PreSICE to configure the PLL in preparation for reconfiguration when the PMA is configured as a CMU PLL: 1'b1: Request PreSICE to configure the PLL in reconfiguration mode 1'b0: Reconfiguration mode not requested
Background calibration	0x542[0]	read-write	enable_background_cal	Request background calibration when the feature is enabled in the Native PHY IP core: • 1'b1: Enabled Background Calibration • 1'b0: Disable Background Calibration

A.2.2. Optional Reconfiguration Logic CMU PLL- Capability

Enables CMU PLL capabilities to be readable.

Name	Address	Type	Attribute Name	Encodings
IP Identifier	0x400[7:0] 0x401[7:0] 0x402[7:0] 0x403[7:0]	read-only	cmu_address_id	Unique identifier for the PLL instance.
Status Register Enabled	0x404[0]	read-only	cmu_status_register_enable	Indicates if the status registers have been enabled. 1'b1 indicates the feature is enabled.
Control Register Enabled	0x405[0]	read-only	cmu_control_register_enable	Indicates if the control registers have been enabled. 1'b1 indicates the feature is enabled.

A.2.3. Optional Reconfiguration Logic CMU PLL- Control & Status

Enables users to read the status of CMU PLL functions and reset the CMU PLL.

Name	Address	Type	Attribute Name	Encodings
PLL Locked Status	0x480[0]	read-only	cmu_pll_locked	Indicates if the CMU PLL is locked. 1'b1 indicates the CMU PLL is locked.
PLL Calibration Busy Status	0x480[1]	read-only	cmu_cal_busy	Indicates the CMU PLL calibration status. 1'b1 indicates the CMU PLL is currently being calibrated.

continued...

Name	Address	Type	Attribute Name	Encodings
Avalon memory-mapped interface Bus Busy Status	0x480[2]	`read-only`	`cmu_avmm_busy`	Shows the status of internal configuration bus arbitration. When set to 1'b1, PreSICE has control of the internal configuration bus. When set to 1'b0, you have control of the internal configuration bus. Refer to the *Arbitration* section for more details.
PLL Power-down	0x4E0[0]	`read-write`	`cmu_pll_powerdown`	Drives the PLL power-down when the Override is set
Override PLL Power-down	0x4E0[1]	`read-write`	`cmu_override_pll_powerdown`	Ensures the PLL listens to the NPDME pll_powerdown register. 1'b1 indicates the receiver listens to the NPDME `pll_powerdown`.

A.2.4. Embedded Streamer (CMU PLL)

Enables logic in CMU PLL to store the individual profile information and perform streaming.

Name	Address	Type	Attribute Name	Encodings
Configuration profile select	0x540[2:0]	`read-write`	`cmu_cfg_sel`	Binary encoding of the configuration profile to stream Profile 0: 3'b000 Profile 1: 3'b001 Profile 2: 3'b010 Profile 3: 3'b011 Profile 4: 3'b100 Profile 5: 3'b101 Profile 6: 3'b110 Profile 7: 3'b111
Start streaming	0x540[7]	`read-write`	`cmu_cfg_load`	Set to 1'b1 to initiate streaming, self-clearing bit
Busy status bit	0x541[0]	`read-only`	`cmu_rcfg_busy`	Bit is set to: 1'b1: streaming is in progress 1'b0: streaming is complete

A.3. FPLL Logical Register Map

Stratix 10 FPLL Register Map Summary

Feature	Feature Description
fPLL Calibration	fPLL Calibration allows users to optimize the fPLL performance when changing data rates.
Optional Reconfiguration Logic fPLL-Capability	Enables fPLL capabilities to be readable.
Optional Reconfiguration Logic fPLL-Control & Status	Enables users to read the status of fPLL functions and reset the fPLL.
Embedded Streamer (fPLL)	Enables logic in fPLL to store the individual profile information and perform .mif streaming for reconfiguration application.

A.3.1. fPLL Calibration

fPLL Calibration allows users to optimize the fPLL performance when changing data rates.

Name	Address	Type	Attribute Name	Encodings
Internal configuration bus arbitration register for FPLL	0x000[0]	`read-write`	`pcs_arbiter_ctrl`	This bit arbitrates the control of internal configuration bus. Write 1'b0 if you want to control the internal configuration bus Write 1'b1 to pass the internal configuration bus control to PreSICE.
fPLL Calibration Status	0x000[1]	`read-write`	`pcs_cal_done`	Status for whether or not calibration is done This is the inverted `cal_busy` signal. 1'b1: calibration done 1'b0: calibration not done
fPLL calibration enable	0x100[1]	`read-write`	`fpll_calibration`	fPLL calibration enable 1'b1: fPLL calibration enable on 1'b0: fPLL calibration enable off
Request PreSICE to configure the fPLL in preparation for reconfiguration	0x100[0]	`read-write`	`pre_reconfig`	Request PreSICE to configure the fPLL in preparation for reconfiguration ENCODINGS: 1'b1: Request PreSICE to configure the PLL in reconfiguration mode 1'b0: Reconfiguration mode not requested

A.3.2. Optional Reconfiguration Logic fPLL-Capability

Enables fPLL capabilities to be readable.

Name	Address	Type	Attribute Name	Encodings
IP Identifier	0x400[7:0] 0x401[7:0] 0x402[7:0] 0x403[7:0]	`read-only`	`fpll_address_id`	Unique identifier for the PLL instance.
Status Register Enabled	0x404[0]	`read-only`	`fpll_status_register_enable`	Indicates if the status registers have been enabled. 1'b1 indicates the feature is enabled.
Control Register Enabled	0x405[0]	`read-only`	`fpll_control_register_enable`	Indicates if the control registers have been enabled. 1'b1 indicates the feature is enabled.
Master CGB enabled	0x410[0]	`read-only`	`fpll_mcgb_enable`	Indicates if the Master CGB is enabled. 1'b1 indicates the master CGB is enabled.

A.3.3. Optional Reconfiguration Logic fPLL-Control & Status

Enables users to read the status of fPLL functions and reset the fPLL.

Name	Address	Type	Attribute Name	Encodings
PLL Locked Status	0x480[0]	read-only	fpll_pll_locke d	Indicates if the fPLL is locked. 1'b1 indicates the fPLL is locked.
PLL Calibration Busy Status	0x480[1]	read-only	fpll_cal_busy	Indicates the fPLL calibration status. 1'b1 indicates the fPLL is currently being calibrated.
Avalon memory-mapped interface Bus Busy Status	0x480[2]	read-only	fpll_avmm_busy	Shows the status of internal configuration bus arbitration. When 1'b1, PreSICE has control of the internal configuration bus. When 1'b0, you have control of the internal configuration bus. Refer to the `Arbitration` section for more details.
PLL Power-down	0x4E0[0]	read-write	fpll_pll_power down	Drives the PLL power-down when the Override is set.
Override PLL Power-down	0x4E0[1]	read-write	fpll_override_ pll_powerdown	Ensures the PLL listens to the NPDME pll_powerdown register. 1'b1 indicates the receiver listens to the NPDME `pll_powerdown`.

A.3.4. Embedded Streamer (fPLL)

Enables logic in fPLL to store the individual profile information and perform .mif streaming for reconfiguration application.

Name	Address	Type	Attribute Name	Encodings
Configuration profile select	0x540[2:0]	read-write	fpll_cfg_sel	Binary encoding of the configuration profile to stream Profile 0: 3'b000 Profile 1: 3'b001 Profile 2: 3'b010 Profile 3: 3'b011 Profile 4: 3'b100 Profile 5: 3'b101 Profile 6: 3'b110 Profile 7: 3'b111
Start streaming	0x540[7]	read-write	fpll_cfg_load	Set to 1'b1 to initiate streaming, self-clearing bit
Busy status bit	0x541[0]	read-only	fpll_rcfg_busy	Bit is set to: 1'b1: streaming is in progress 1'b0: streaming is complete

A.4. Channel Logical Register Map

Stratix 10 Channel Register Map Summary

Feature	Feature Description
Pre-emphasis	Boost high frequency component by de-emphasizing low frequency components to compensate the low-pass characteristics of the physical medium.
VOD	TX output differential swing.
TX Compensation	When enabled, it reduces the PDN induced ISI jitter.

continued...

Feature	Feature Description
Slew Rate	This attribute of the TX PMA is datarate dependent and is important when doing datarate reconfiguration.
Loopback	The PMA supports serial, pre-CDR reverse serial and post-CDR reverse serial loopback paths.
RX PMA	This attribute of the RX PMA is datarate dependent and is important when doing datarate reconfiguration.
Setting RX PMA Adaptation Modes on page 459	RX PMA adaptation supports the following modes: • Manual CTLE, Manual VGA, DFE Off • Adaptive CTLE, Adaptive VGA, DFE Off • Adaptive CTLE, Adaptive VGA, 1-Tap Adaptive DFE • Adaptive CTLE, Adaptive VGA, All-Tap Adaptive DFE
Manual CTLE	You can use the CTLE in Manual Mode. This section describes how to set the values.
Manual VGA	You can use the VGA in Manual Mode. This section describes how to set the values.
Adaptation Control - Start	In the adaptive modes, this section describes how you can start adaptation.
Adaptation Control - Stop	In the adaptive modes, this section describes how you can stop adaptation.
Adapted Value Readout	In the adaptive modes, this section describes how you can read the adapted values. An example is this sequence for reading adapted VGA value: STEP 1: Set pm_cr2_tx_rx_testmux_select to 4'b1011 STEP 2: Set adp_status_sel to 6'b000101 STEP 3: Read testmux[4:0] as the adapted VGA value.
CDR/CMU and PMA Calibration	Enables user to optimize CDR/CMU and PMA performance when changing the data rate and reference clock.
Reset	Enables the reset of the channel TX, RX PCS, and PMA.
Optional Reconfiguration Logic PHY-Capability	Enables Native PHY channels' capabilities to be readable.
Optional Reconfiguration Logic PHY-Control & Status	Enables users to read the status of channel and control the channels' behavior.
Embedded Streamer (Native PHY)	Enables logic in Native PHY to store the individual profile information and perform .mif streaming.
Square Wave Generator	Square wave generator can be used to provide a simple way to simulate traffic.
PRBS Generator	PRBS generator can be used to provide a simple way to simulate traffic.
PRBS Verifier	PRBS Verifier can be used to provide a simple and easy way to verify and characterize high-speed links. Note that you must also configure the registers under "Other registers needed for PRBS verifier" in order to use this feature.
EMIB-related registers needed for PRBS Verifier	Used only in conjunction with the PRBS Verifier feature.
Static Polarity Inversion	Static polarity inversion can be used to easily invert the polarity of the TX or RX serial data or both.
PRBS Soft Accumulators	PRBS soft accumulators can be used to count the number of accumulated bits and errors when the hard PRBS blocks are used.

A.4.1. Transmitter PMA Logical Register Map

A.4.1.1. Pre-emphasis

Boost high frequency component by de-emphasizing low frequency components to compensate the low-pass characteristics of the physical medium.

Pre-emphasis 1st post-tap magnitude	0x105[4:0]	read-write	pre_emp_switching_ctrl_1st_post_tap	Sets 1st post-tap magnitude. Direct mapped until 24. 5'b00000: 0 (until) 5'b11000: 24
Pre-emphasis 1st post-tap polarity	0x105[6]	read-write	pre_emp_sign_1st_post_tap	Set 1st post-tap polarity. 1'b1: Negative Polarity 1'b0: Positive Polarity
Pre-emphasis 1st pre-tap magnitude	0x107[4:0]	read-write	pre_emp_switching_ctrl_pre_tap_1t	Sets 1st pre-tap magnitude. Direct mapped until 15. 5'b00000: 0 (until) 5'b01111: 15
Pre-emphasis 1st pre-tap polarity	0x107[5]	read-write	pre_emp_sign_pre_tap_1t	Set 1st pre-tap polarity. 1'b1: Negative Polarity 1'b0: Positive Polarity

A.4.1.2. VOD

TX output differential swing.

Name	Address	Type	Attribute Name	Encodings
TX Output Swing level	0x109[4:0]	read-write	vod_output_swing_ctrl	Sets TX output swing level. Direct mapped. 5'b10001: 17 (600 mV) (until) 5'b11111: 31 (VCCT or Transmitter Power Supply Voltage)

A.4.1.3. TX Compensation

When enabled, it reduces the PDN induced ISI jitter.

Name	Address	Type	Attribute Name	Encodings
Transmitter High-Speed Compensation	0x109[5]	read-write	compensation_en	1'b0: Compensation OFF 1'b1: Compensation ON

A.4.1.4. Slew Rate

This attribute of the TX PMA is datarate dependent and is important when doing datarate reconfiguration.

Name	Address	Type	Attribute Name	Encodings
Slew Rate	0x10C[2:0]	read-write	slew_rate_ctrl	3'b000: Slowest setting until 3'b101: Fastest setting

A.4.2. Receiver PMA Logical Register Map

A.4.2.1. RX PMA

This attribute of the RX PMA is datarate dependent and is important when doing datarate reconfiguration.

Name	Address	Type	Attribute Name	Encodings
Equalization Bandwidth	0x11F[5:4]	read-write	eq_bw_sel	2'b00: Datarate ≤ 6.5Gbps 2'b01: 6.5Gbps < Datarate ≤ 12.5Gbps 2'b10: 12.5Gbps < Datarate ≤ 19.2Gbps 2'b11: 19.2Gbps < Datarate

A.4.2.2. Setting RX PMA Adaptation Modes

RX PMA adaptation supports reconfiguration among the following modes:

- Manual CTLE, Manual VGA, DFE Off
- Adaptive CTLE, Adaptive VGA, DFE Off
- Adaptive CTLE, Adaptive VGA, 1-Tap Adaptive DFE
- Adaptive CTLE, Adaptive VGA, All-Tap Adaptive DFE

Name	Address	Type	Attribute Name	Encodings
Bypass DFE	0x161[6]	read-write	adp_dfe_fxtap_bypass	1'b0: Do not bypass DFE 1'b1: Bypass DFE
Bypass DLEV	0x161[5]	read-write	adp_dlev_bypass	1'b0: Do not bypass DLEV 1'b1: Bypass DLEV
Bypass VGA	0x149[7]	read-write	adp_vga_bypass	1'b0: Do not bypass VGA 1'b1: Bypass VGA
Bypass CTLE - AC	0x15F[3]	read-write	adp_ctle_bypass_ac	1'b0: Do not bypass CTLE AC Gain 1'b1: Bypass CTLE AC Gain
Bypass CTLE - EQ	0x15C[7]	read-write	adp_ctle_bypass_dc	1'b0: Do not bypass CTLE EQ Gain 1'b1: Bypass CTLE EQ Gain
Enable DFE adaptation	0x148[1]	read-write	adp_dfe_en	1'b0: Disable 1'b1: Enable
Enable DLEV adaptation	0x148[3]	read-write	adp_dlev_en	1'b0: Disable 1'b1: Enable
Enable VGA adaptation	0x148[4]	read-write	adp_vga_en	1'b0: Disable 1'b1: Enable
Enable CTLE - AC adaptation	0x161[7]	read-write	adp_ac_ctle_en	1'b0: Disable 1'b1: Enable
Enable CTLE - EQ adaptation	0x148[5]	read-write	adp_dc_ctle_en	1'b0: Disable 1'b1: Enable
Bypass DFE taps 4-15	0x14C[6:5]	read-write	adp_dfe_tap_sel_en	2'b00: Bypass no DFE taps 2'b10: Bypass DFE taps 4-15 Other encodings: Not used

continued...

Name	Address	Type	Attribute Name	Encodings
Hold DFE taps 2-3	0x15D[3:1]	read-write	adp_dfe_hold_sel	3'b000: Do not hold DFE taps 3'b101: Hold DFE taps 2-3 Other encodings: Not used
Enable the edge samplers sampling the edges without DFE tap 1 correction	0x131[6]	read-write	pdb_edge_pre_h1	1'b0: Disable 1'b1: Enable
Enable the edge samplers sampling the edges with DFE tap 1 correction	0x131[7]	read-write	pdb_edge_pst_h1	1'b0: Disable 1'b1: Enable
Enable DFE taps 4-9	0x12F[6]	read-write	pdb_tap_4t9	1'b0: Disable 1'b1: Enable
Enable DFE taps 10-15	0x12F[7]	read-write	pdb_tap_10t15	1'b0: Disable 1'b1: Enable
Enable DFE tap summing node	0x150[5]	read-write	pdb_tapsum	1'b0: Disable 1'b1: Enable
Select CTLE AC and VGA adaptation algorithm	0x14A[6]	read-write	adp_ac_ctle_concurrent_mode_sel	No need to set this in manual mode. 1'b0: AC algorithm mode 1 (VGA adapts first, then AC) 1'b1: AC algorithm mode 2 (AC adapts first, then VGA) Other encodings: Not used
Select CTLE EQ adaptation algorithm	0x162[3:2]	read-write	adp_dc_ctle_mode_sel	No need to set this in manual mode. 2'b00: DC algorithm mode 1 (Window Minimization) 2'b10: DC algorithm mode 3 (DFE H2 Range) Other encodings: Not used
Select CTLE EQ adaptation algorithm	0x167[6:3]	read-write	adp_dc_ctle_mode0_win_start	No need to set this in manual mode. Controls window for EQ gain adaptation (direct-mapped). 4'b0000: 0 until 4'b1111: 15
Select CTLE EQ adaptation algorithm	0x167[2]	read-write	adp_dc_ctle_onetime	No need to set this in manual mode. Sets EQ gain to one-time. 1'b0: EQ gain is not one-time 1'b1: EQ gain is one-time
Select VGA low limit	0x158[6]	read-write	adp_vga_ctle_low_limit	No need to set this in manual mode. 1'b0: Sets VGA low limit to 0 1'b1: Sets VGA low limit to 4
Select DLEV target	0x166[4:0]	read-write	adp_vga_dlev_target	No need to set this in manual mode. Controls the VGA target value. The value in mV is calculated as follows: 4mV*(25 + direct-mapped value)

A.4.2.3. Manual CTLE

You can use the CTLE in Manual Mode. This section describes how to set the values.

Name	Address	Type	Attribute Name	Encodings
Manual CTLE AC Gain value	0x11A[7:4]	`read-write`	`ctle_ac_gain`	Sets manual AC Gain value. Direct mapped. 4'b0000: 0 (-2dB at the peak frequency) (until) 4'b1111: 15 (+10dB at the peak frequency) *Note:* The effect of the gain values in dB will vary across PVT.
Manual CTLE EQ Gain value	0x11E[7] 0x11C[4:0]	`read-write`	`ctle_eq_gain`	Sets manual EQ Gain value. Direct mapped. 6'b000000: 0 (0dB) (until) 6'b101111: 47 (-16dB) *Note:* The effect of the gain values in dB will vary across PVT.

A.4.2.4. Manual VGA

You can use the VGA in Manual Mode. This section describes how to set the values.

Name	Address	Type	Attribute Name	Encodings
Manual VGA Value	0x11F[0] 0x11A[3:0]	`read-write`	`vga_dc_gain`	Sets manual VGA Gain value. Direct mapped. 5'b00000: 0 (-5dB) (until) 5'b11111: 31 (+7dB) *Note:* The effect of the gain values in dB varies across PVT.

A.4.2.5. Adaptation Control - Start

In the adaptive modes, this section describes how you can start adaptation.

Name	Address	Type	Attribute Name	Encodings
STEP 1: Set adaptation reset to 0. Wait 100ms.	0x148[0]	`read-write`	`adp_rstn`	1'b0: Adaptation in reset 1'b1: Adaptation not in reset
STEP 2: Set adaptation reset to 1	0x148[0]	`read-write`	`adp_rstn`	1'b0: Adaptation in reset 1'b1: Adaptation not in reset
STEP 3: Set adaptation start to 1	0x14C[0]	`read-write`	`adp_adapt_start`	1'b0: Adaptation not started 1'b1: Adaptation started

A.4.2.6. Adaptation Control - Stop

In the adaptive modes, this section describes how you can stop adaptation.

Name	Address	Type	Attribute Name	Encodings
STEP 1: Set adaptation reset to 0. Wait 100ms.	0x148[0]	`read-write`	`adp_rstn`	1'b0: Adaptation in reset 1'b1: Adaptation not in reset
STEP 2: Set adaptation start to 0	0x14C[0]	`read-write`	`adp_adapt_start`	1'b0: Adaptation not started 1'b1: Adaptation started

A.4.2.7. Adapted Value Readout

In the adaptive modes, this section describes how you can read the adapted values. An example is this sequence for reading adapted VGA value:

- STEP 1: Set pm_cr2_tx_rx_testmux_select to 4'b1011
- STEP 2: Set adp_status_sel to 6'b000101
- STEP 3: Read testmux[4:0] as the adapted VGA value.

Name	Address	Type	Attribute Name	Encodings
STEP 1: Set testmux to read from adaptation block	0x171[4:1]	`read-write`	`pm_cr2_tx_rx_testmux_select`	4'b1011: Adaptation block
STEP 2: Select which signal from adaptation to read	0x149[5:0]	`read-write`	`adp_status_sel`	6'b000101: VGA 6'b000100: CTLE - EQ Gain 6'b000011: CTLE - AC Gain 6'b000111: DFE Tap # 1 6'b001000: DFE Tap # 2 6'b001001: DFE Tap # 3 6'b001010: DFE Tap # 4 6'b001011: DFE Tap # 5 6'b001100: DFE Tap # 6 6'b001101: DFE Tap # 7 6'b001110: DFE Tap # 8 6'b001110: DFE Tap # 9 6'b001111: DFE Tap # 10 6'b001111: DFE Tap # 11 6'b010000: DFE Tap # 12 6'b010000: DFE Tap # 13 6'b010001: DFE Tap # 14 6'b010001: DFE Tap # 15
STEP 3: Read this value from the testmux	0x17E[7:0]	`read-only`	`testmux`	VGA value = testmux[4:0] CTLE - EQ Gain value = testmux[5:0] CTLE - AC Gain value = testmux[6:3] DFE Tap # 1 sign = testmux[6:6] & magnitude= testmux[5:0] DFE Tap # 2 sign = testmux[5:5] & magnitude= testmux[4:0] DFE Tap # 3 sign = testmux[5:5] & magnitude= testmux[4:0] DFE Tap # 4 sign = testmux[4:4] & magnitude= testmux[3:0] DFE Tap # 5 sign = testmux[4:4] & magnitude= testmux[3:0] DFE Tap # 6 sign = testmux[4:4] & magnitude= testmux[3:0]

continued...

Name	Address	Type	Attribute Name	Encodings
				DFE Tap # 7 sign = testmux[4:4] & magnitude= testmux[3:0] DFE Tap # 8 sign = testmux[2:2] & magnitude= testmux[1:0] DFE Tap # 9 sign = testmux[6:6] & magnitude= testmux[5:3] DFE Tap # 10 sign = testmux[3:3] & magnitude= testmux[2:0] DFE Tap # 11 sign = testmux[7:7] & magnitude= testmux[6:4] DFE Tap # 12 sign = testmux[3:3] & magnitude= testmux[2:0] DFE Tap # 13 sign = testmux[7:7] & magnitude= testmux[6:4] DFE Tap # 14 sign = testmux[3:3] & magnitude= testmux[2:0] DFE Tap # 15 sign = testmux[7:7] & magnitude= testmux[6:4]

A.4.3. Pattern Generators and Checkers

A.4.3.1. Square Wave Generator

Square wave generator can be used to provide a simple way to simulate traffic.

Name	Address	Type	Attribute Name	Encodings
Square Wave Block Select	0x008[6:5] 0x006[2:0]	`read-write`	`tx_pma_data_sel`	Select Square Wave Pattern Generator. Register format {0x008[6:5], 0x006[2:0]} 5'b00101: Square Wave Pattern
Square Wave Clock Enable	0x006[7]	`read-write`	`sqwgen_clken`	1'b0: Square Wave Clock Disable 1'b1: Square Wave Clock Enable
Square Wave Pattern Select	0x008[3:0]	`read-write`	`sq_wave_num`	Select square wave pattern. The number "n" is the number of ones followed by the number of ones in the square wave. 4'b0001: n=1 4'b0100: n=4 4'b1000: n=6 4'b0110: n=8

A.4.3.2. PRBS Generator

PRBS generator can be used to provide a simple way to simulate traffic.

Name	Address	Type	Attribute Name	Encodings
PRBS Block Select	0x008[6:5] 0x006[2:0]	`read-write`	`tx_pma_data_sel`	Select PRBS or Square Wave Pattern Generator Block. Register format {0x008[6:5], 0x006[2:0]} 5'b00100: PRBS Pattern
PRBS_9 Width Select	0x006[3]	`read-write`	`prbs9_dwidth`	PRBS9 data width Selection. 1'b1: PRBS9 10 bits 1'b0: PRBS 64 bits

continued...

Name	Address	Type	Attribute Name	Encodings
PRBS Clock Enable	0x006[6]	`read-write`	`prbs_clken`	1'b0: PRBS Clock Disable 1'b1: PRBS Clock Enable
PRBS Pattern Select	0x008[4] 0x007[7:4]	`read-write`	`prbs_gen_pat`	Select PRBS pattern generator. Register format {0x008[4], 0x007[7:4]} 5'b00001: PRBS 7 5'b00010: PRBS 9 5'b00100: PRBS 15 5'b01000: PRBS 23 5'b10000: PRBS 31 5'b00000: Disable PRBS in 64-bit mode
Serializer Factor	0x110[2:0]	`read-write`	`ser_mode`	Serializer bit width mode. 3'b011: 64 bit 3'b100: 10 bit

A.4.3.3. PRBS Verifier

PRBS Verifier can be used to provide a simple and easy way to verify and characterize high-speed links. Note that you must also configure the registers in *Other registers needed for PRBS verifier* in order to use this feature.

Name	Address	Type	Attribute Name	Encodings
PRBS Verifier Clk Enable	0x00A[7]	`read-write`	`prbs_clken`	1'b0: PRBS Clock Disable 1'b1: PRBS Clock Enable
PRBS Error Mask Threshold	0x00B[3:2]	`read-write`	`rx_prbs_mask`	Masks out the initial errors seen by PRBS Verifier. 2'b11: PRBS Mask 1024 2'b00: PRBS Mask 128 2'b01: PRBS Mask 256 2'b10: PRBS Mask 512
PRBS Pattern Select	0x00C[0] 0x00B[7:4]	`read-write`	`prbs_ver`	Register format {0x00C[0], 0x00B[7:4]} : 5'b00001: PRBS 7 5'b00010: PRBS 9 5'b00100: PRBS 15 5'b01000: PRBS 23 5'b10000: PRBS 31 5'b00000: Disable PRBS in 64-bit mode
PRBS_9 Width Select	0x00C[3]	`read-write`	`prbs9_dwidth`	1'b1: PRBS 9 10 bits 1'b0: PRBS 9 64bits
Deserializer Factor	0x13F[3:0]	`read-write`	`deser_factor`	4'b0001: 10 bits 4'b1110: 64 bits

A.4.3.4. Other registers needed for PRBS Verifier

Used only in conjunction with the PRBS Verifier feature.

Name	Address	Type	Attribute Name	Encodings
Clkslip source select	0x00A[2]	read-write	clkslip_sel	1'b0: Source is PLD
Datapath mapping mode	0x210[4:0]	read-write	datapath_mapping_mode	5'b01001: 10G, 32-bit datapath with 1:1 FIFO
FIFO double write enable	0x214[0]	read-write	fifo_double_write	1'b0 = Single width mode
FIFO read clock select	0x322[6:5]	read-write	fifo_rd_clk_sel	2'b10: PLD_RX_CLK1 for FIFO read clock
FIFO double width mode	0x312[6]	read-write	fifo_double_read	1'b0: Single width mode
Word Marking Bit	0x212[7]	read-write	word_mark	1'b0: Disable
RX FIFO Full threshold	0x213[4:0]	read-write	rxfifo_full	5'b00111: RX FIFO full threshold
RX FIFO power saving mode	0x218[7:6]	read-write	rx_fifo_power_mode	2'b01: Full width, half depth
Phase comp mode read delay	0x213[7:5]	read-write	phcomp_rd_del	3'b010: Read delay 2
Adapter Loopback mode	0x218[0]	read-write	adapter_lpbk_mode	1'b0: DISABLE
EMIB Loopback mode	0x215[7]	read-write	aib_lpbk_mode	1'b0: DISABLE
FIFO write clock select	0x223[1:0]	read-write	fifo_wr_clk_sel	2'b00 = FIFO Write Clock Select pld_pcs_rx_clk_out
FIFO write clock select	0x322[4]	read-write	fifo_wr_clk_sel	1'b0: Uses rx_transfer_clk for FIFO write clock
FIFO mode	0x315[2:0]	read-write	rxfifo_mode	3'b000: Phase compensation
FIFO read allowed or not when empty	0x313[6]	read-write	fifo_stop_rd	1'b0: Read when empty
FIFO write allowed or not when full	0x313[7]	read-write	fifo_stop_wr	1'b0: Write when full
PLD clk1 delay path sel	0x321[4:1]	read-write	pld_clk1_delay_sel	4'b1100: Delay path 12
FIFO Partially empty threshold	0x313[5:0]	read-write	rxfifo_pempty	6'b000010: Partially empty threshold = 2
RX FIFO Write control	0x318[1]	read-write	rx_fifo_write_ctrl	1'b1: Keep writing when block lock is lost
RX FIFO Power saving mode	0x31A[4:2]	read-write	rx_fifo_power_mode	3'b001: Full width, single width mode
Custom pulse stretching amount for PLD async outputs	0x320[2:0]	read-write	stretch_num_stages	3'b010: 2 cycle stretch
EMIB clock select	0x322[1:0]	read-write	aib_clk1_sel	2'b01: Uses PLD_PCS_RX_CLK_OUT for EMIB clk
Word align	0x318[0]	read-write	word_align	1'b0: Disable word align

continued...

Name	Address	Type	Attribute Name	Encodings
Loopback mode	0x315[6]	`read-write`	`lpbk_mode`	1'b0: DISABLE
Data Valid mode	0x312[7]	`read-write`	`dv_mode`	1'b0: Data valid disable
FIFO empty threshold	0x311[5:0]	`read-write`	`rxfifo_empty`	6'b000000: RX FIFO empty threshold
FIFO Full threshold	0x312[5:0]	`read-write`	`rxfifo_full`	6'b000111: FIFO Full threshold
Deserializer EMIB clk x1	0x164[7]	`read-write`	`deser_aibck_x1`	1'b1: Sends x1 clock out

A.4.3.5. PRBS Soft Accumulators

PRBS soft accumulators can be used to count the number of accumulated bits and errors when the hard PRBS blocks are used.

Name	Address	Type	Attribute Name	Encodings
Counter enable	0x500[0]	`read-write`	`prbs_counter_en`	1'b1: Enabled 1'b0: Disabled
Reset	0x500[1]	`read-write`	`prbs_reset`	1'b1: Reset the error accumulators 1'b0: Error accumulators not in reset
Error Count Snapshot	0x500[2]	`read-write`	`prbs_snap`	1'b1: Capture snapshot 1'b0: Do not take snapshot
PRBS Done	0x500[3]	`read-only`	`prbs_done`	1'b1: Asserted, consecutive PRBS patterns received 1'b0: Not asserted, PRBS not detected
Accumulated Error Count[7:0]	0x501[7:0] 0x502[7:0] 0x503[7:0] 0x504[7:0] 0x505[7:0] 0x506[7:0] 0x507[1:0]	`read-only`	`prbs_acc_err_cnt`	The accumulated error count is stored in a 50 bit register. The first set of 8 bits are stored in this register. You need to use all the 50 bits to get the complete information on the number of the error count.
Accumulated Bit Count[7:0]	0x50D[7:0] 0x50E[7:0] 0x50F[7:0] 0x510[7:0] 0x511[7:0] 0x512[7:0] 0x513[1:0]	`read-only`	`prbs_acc_bit_cnt`	The accumulated bits are stored in a 50 bit register. The first set of 8 bits are stored here in this register. You need to use all the 50 bits to gather information on the number of accumulated bits.

A.4.4. Loopback

The PMA supports serial, pre-CDR reverse serial and post-CDR reverse serial loopback paths.

Name	Address	Type	Attribute Name	Encodings
Reverse Serial Loopback Path	0x144[1] 0x132[5:4]	`read-write`	`reverse_serial_loopback`	Sets transceiver diagnostic (pre-CDR) or post-CDR reverse loopback mode

continued...

Name	Address	Type	Attribute Name	Encodings
	0x137[7]	read-write	diag_loopback_enable	Encodings {0x11D[0], 0x132[5:4], 0x137[7], 0x144[1], 0x142[4]} : 6'b000000 - Disable reverse serial loopback 6'b100101 - Enable pre-CDR reverse serial loopback 6'b001010 - Enable post-CDR reverse serial loopback
	0x144[1] 0x142[4] 0x11D[0]	read-write	loopback_mode	

A.4.5. Optional Reconfiguration Logic PHY- Capability

Enables Native PHY channels' capabilities to be readable.

Name	Address	Type	Attribute Name	Encodings
IP Identifier	0x400 0x401 0x402 0x403	read-only	PHYIP_address_id	Unique identifier for the Native PHY instance.
Status Register Enabled	0x404[0]	read-only	PHYIP_status_register_enable	Indicates if the status registers have been enabled. 1'b1 indicates the feature is enabled.
Control Register Enabled	0x405[0]	read-only	PHYIP_control_register_enable	Indicates if the control registers have been enabled. 1'b1 indicates the feature is enabled.
Number of Channels	0x410[7:0]	read-only	PHYIP_chnls	Shows the number of channels specified for the Native PHY instance.
Channel Number	0x411[7:0]	read-only	PHYIP_chnl_num	Shows the unique channel number.
Duplex	0x412[1:0]	read-only	PHYIP_duplex	Shows transceiver mode: 2'b00: <unused> 2'b10: TX 2'b01: RX 2'b11: duplex

A.4.6. Optional Reconfiguration Logic PHY- Control & Status

Enables users to read the status of channel and control the channels' behavior.

Name	Address	Type	Attribute Name	Encodings
RX Locked to Data Status	0x480[0]	read-only	ch_rx_is_lockedtodata	Shows the status of the current channel's rx_is_lockedtodata signal. 1'b1 indicates the receiver is locked to the incoming data.
RX Locked to Reference Status	0x480[1]	read-only	ch_rx_is_lockedtoref	Shows the status of the current channel's rx_is_lockedtoref signal. 1'b1 indicates the receiver is locked to the reference clock.
TX Calibration Busy Status	0x481[0]	read-only	ch_tx_cal_busy	Shows the status of the transmitter calibration status. 1'b1 indicates the transmitter calibration is in progress.

continued...

Name	Address	Type	Attribute Name	Encodings
RX Calibration Busy Status	0x481[1]	`read-only`	`ch_rx_cal_busy`	Shows the status of the receiver calibration status. 1'b1 indicates the receiver calibration is in progress.
Avalon memory-mapped interface Bus Busy Status	0x481[2]	`read-only`	`ch_avmm_busy`	Shows the status of internal configuration bus arbitration. When 1'b1, PreSICE has control of the internal configuration bus. When 1'b0, you have control of the internal configuration bus. Refer to the *Arbitration* section for more details. Refer to the *Calibration* chapter for more details on calibration registers and performing user recalibration.
TX Calibration Busy Status Enable	0x481[4]	`read-write`	`ch_tx_cal_busy_enable`	PMA channel tx_cal_busy output enable. The power up default value is 0x1. 1'b1: The tx_cal_busy output and `ch_tx_cal_busy` 0x481[0] are asserted high whenever PMA TX or RX calibration is running. 1'b0: The tx_cal_busy output or `ch_tx_cal_busy` 0x481[0] is never asserted high.
RX Calibration Busy Status Enable	0x481[5]	`read-write`	`ch_rx_cal_busy_enable`	PMA channel rx_cal_busy output enable. The power up default value is 0x1. 1'b1: The rx_cal_busy output and `ch_rx_cal_busy` 0x481[1] are asserted high whenever PMA TX or RX calibration is running. 1'b0: The rx_cal_busy output or `ch_rx_cal_busy` 0x481[1] is never asserted high.
Set RX Lock to Data	0x4E0[0]	`read-write`	`ch_set_rx_locktodata`	Asserts the `set_rx_locktodata` signal to the receiver. See override`_set_rx_locktodata` row below. 1'b1 sets the NPDME `set_rx_locktodata` register.
Set RX Lock to Reference	0x4E0[1]	`read-write`	`ch_set_rx_locktoref`	Asserts the `set_rx_locktoref` signal to the receiver. See override`_set_rx_locktoref` row below. 1'b1 set the NPDME `set_rx_locktoref` register.
Override Set RX Lock to Data	0x4E0[2]	`read-write`	`ch_override_set_rx_locktodata`	Selects whether the receiver listens to the NPDME set_rx_locktodata register or the `rx_set_locktodata` port. 1'b1 indicates the receiver listens to the NPDME `set_rx_locktodata` register. 1'b0 indicates the receiver listens to the NPDME `set_rx_locktodata` port.
Override Set RX Lock to Reference	0x4E0[3]	`read-write`	`ch_override_set_rx_locktoref`	Selects whether the receiver listens to the NPDME set_rx_locktoref register or the rx_set_locktoref port. 1'b1 indicates the receiver listens to the NPDME set_rx_locktoref register. 1'b0 indicates the receiver listens to the NPDME set_rx_locktoref port.
RX Serial Loopback	0x4E1[0]	`read-write`	`ch_rx_seriallpbken`	Enables the `rx_seriallpbken` feature in the transceiver. 1'b1 enables serial loopback.

Send Feedback

A.4.7. Embedded Streamer (Native PHY)

Enables logic in Native PHY to store the individual profile information and perform .mif streaming.

Name	Address	Type	Attribute Name	Encodings
Configuration Profile Select	0x540[2:0]	read-write	PHYIP_cfg_sel	Selects the Configuration from the list of configurations specified by you in the Native PHY IP Parameter Editor. It represents the profile number between the GUI multi-profile generation. Profile 0: 3'b000 Profile 1: 3'b001 Profile 2: 3'b010 Profile 3: 3'b011 Profile 4: 3'b100 Profile 5: 3'b101 Profile 6: 3'b110 Profile 7: 3'b111
Broadcast enable	0x540[6]	read-write	PHYIP_bcast_en	Selection to broadcast the profile selected using register cfg_sel to the channels.1'b1- Broadcast the same profile to all the channels ,1'b0- Single Channel
Start Streaming	0x540[7]	read-write	PHYIP_cfg_load	Set to 1'b1 to initiate streaming, self-clearing bit
Busy Status Bit	0x541[0]	read-only	PHYIP_rcfg_bus y	This bit indicates the status of the profile streaming 1'b0: Streaming is complete 1'b1: Streaming is in progress

A.4.8. Static Polarity Inversion

Static polarity inversion can be used to easily invert the polarity of the TX or RX serial data or both.

Name	Address	Type	Attribute Name	Encodings
TX bit polarity inversion	0x007[2]	read-write	tx_static_pola rity_inversion	Invert TX bit polarity selection. 1'b1: Disable static polarity inversion 1'b0: Enable static polarity inversion
RX bit polarity inversion	0x00A[4]	read-write	rx_static_pola rity_inversion	Invert RX bit polarity selection. 1'b1: Disable static polarity inversion 1'b0: Enable static polarity inversion

A.4.9. Reset

Enables the reset of the channel TX, RX PCS, and PMA.

Name	Address	Type	Attribute Name	Encodings
RX Analog Reset	0x4E2[0]	read-write	rx_analogreset	Drives the rx_analogreset signal when the override is set 1'b1: reset asserted

continued...

Name	Address	Type	Attribute Name	Encodings
				1'b0: reset de-asserted
RX Digital Reset	0x4E2[1]	read-write	rx_digitalreset	Drives the `rx_digitalreset` signal when the override is set 1'b1: reset asserted 1'b0: reset de-asserted
TX Analog Reset	0x4E2[2]	read-write	tx_analogreset	Drives the `tx_analogreset` signal when the override is set 1'b1: reset asserted 1'b0: reset de-asserted
TX Digital Reset	0x4E2[3]	read-write	tx_digitalreset	Drives the `tx_digitalreset` signal when the override is set 1'b1: reset asserted 1'b0: reset de-asserted
NPDME RX Analog Reset	0x4E2[4]	read-write	override_rx_analogreset	Selects whether the receiver listens to the NPDME `rx_analogreset` register or the `rx_analogreset` port. 1'b1 indicates the receiver listens to the NPDME `rx_analogreset` register.
NPDME RX Digital Reset	0x4E2[5]	read-write	override_rx_digitalreset	Selects whether the receiver listens to the NPDME `rx_digitalreset` register or the `rx_digitalreset` port. 1'b1 indicates the receiver listens to the NPDME `rx_digitalreset` register.
NPDME TX Analog Reset	0x4E2[6]	read-write	override_tx_analogreset	Selects whether the receiver listens to the NPDME `tx_analogreset` register or the `tx_analogreset` port. 1'b1 indicates the receiver listens to the NPDME `tx_analogreset` register.
NPDME TX Digital Reset	0x4E2[7]	read-write	override_tx_digitalreset	Selects whether the receiver listens to the NPDME `tx_digitalreset` register or the `tx_digitalreset` port. 1'b1 indicates the receiver listens to the NPDME `tx_digitalreset` register.

A.4.10. CDR/CMU and PMA Calibration

Enables user to optimize CDR/CMU and PMA performance when changing the data rate, reference clock, or both.

Name	Address	Type	Attribute Name	Encodings
Internal configuration bus arbitration register	0x000[0]	read-write	pcs_arbiter_ctrl	This bit arbitrates the control of internal configuration bus Write 1'b0 to control the internal configuration bus by user. Write 1'b1 to pass the internal configuration bus control to PreSICE.
PMA Calibration Status	0x000[1]	read-write	pcs_cal_done	Status for calibration done or not done. This is an OR operation, and if both TX and RX calibration are enabled, PreSICE calibrates RX first, then TX. After both have completed, the calibration is complete. This is inverted `cal_busy` signal.

continued...

Name	Address	Type	Attribute Name	Encodings
				1'b1: calibration done 1'b0: calibration not done
PMA RX calibration enable	0x100[0]	read-write	pm_cr2_tx_rx_uc_rx_cal	PMA RX calibration enable 1'b1: RX calibration enable on 1'b0: RX calibration enable off
PMA TX calibration enable	0x100[1]	read-write	uc_tx_cal	PMA TX calibration enable 1'b1: TX calibration enable on 1'b0: TX calibration enable off
Request PreSICE to configure the CDR/CMU PLL in preparation for reconfiguration	0x100[3]	read-write	pre_reconfig	Rate switch flag register when you use the PMA as a RX channel. The power up default value is 0x0. 0x0: PreSICE uses the default CDR charge pump bandwidth from the default memory space. 0x1: PreSICE uses the CDR charge pump bandwidth setting from the Avalon memory-mapped interface space register space. Request PreSICE to configure the PLL in preparation for reconfiguration when the PMA is configured as a CMU PLL: 1'b1: Request PreSICE to configure the PLL in reconfiguration mode 1'b0: Reconfiguration mode not requested
Background calibration	0x542[0]	read-write	enable_background_cal	Request background calibration when the feature is enabled in the Native PHY IP core: • 1'b1: Enabled Background Calibration • 1'b0: Disable Background Calibration

A.5. Logical View Register Map of the L-Tile/H-Tile Transceiver Registers Revision History

Document Version	Changes
2018.10.05	Made the following changes: • Added the Background calibration feature to the "CDR/CMU and PMA Calibration" section.
2018.08.27	Made the following change: • Corrected the binary encoding for TX Output Swing level in the "VOD" section.
2018.07.12	Made the following change: • Updated the description of address 0x100[3] in the "CDR/CMU and PMA Calibration" section.
2018.07.06	Made the following changes: • Clarified the encodings for slew rate in the "Slew Rate" section. • Added the "Setting RX PMA Adaptation Modes" section. • Changed the encodings in the "Static Polarity Inversion" section.
2018.04.16	Made the following changes: • Remove the "Setting RX PMA Adaptation Modes" section.
2017.08.11	Made the following changes:

continued...

Document Version	Changes
	• Added new registers to the "Setting RX PMA Adaptation Modes" section. • Changed the instructions in STEP A and STEP C in the "Adaptation Control - Start" section.
2017.06.06	Initial release.