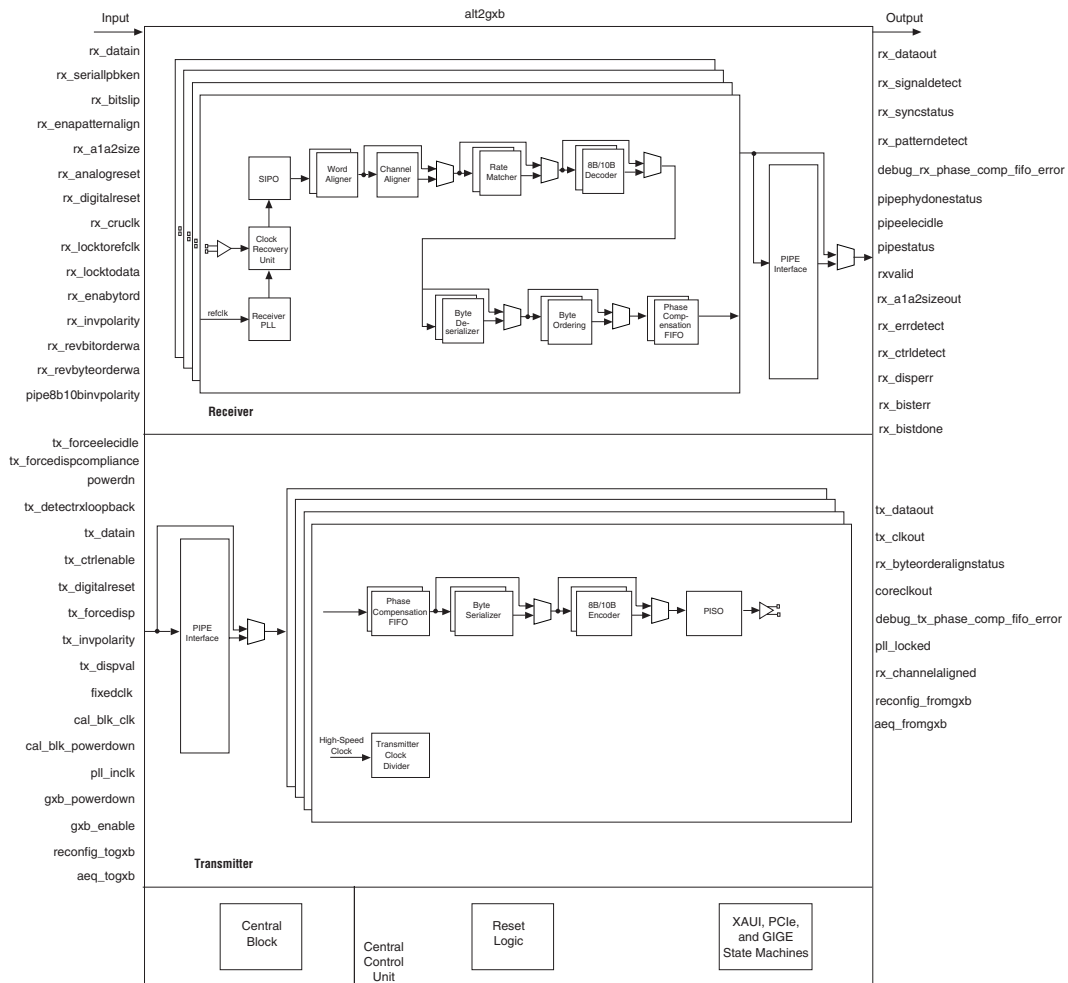


Introduction

This chapter provides detailed information about the architecture of Stratix® II GX devices. Figure 2-1 shows the Stratix II GX block diagram.

Figure 2-1. Stratix II GX Transceiver Block Diagram



Stratix II GX ALT2GXB Ports List

Table 2-1 provides information about the Stratix II GX ports.

<i>Table 2-1. Stratix II GX ALT2GXB Ports (Part 1 of 7)</i>			
Port Name	Input/Output	Description	Scope
Receiver Physical Coding Sublayer (PCS) Ports			
rx_dataout	Output	Receiver parallel data output. The bus width depends on the channel width multiplied by the number of channels per instance.	
rx_clkout	Output	Recovered clock from the receiver channel.	Channel
rx_coreclk	Output	Optional read clock port for the receiver phase compensation first-in first-out (FIFO). If not selected, Quartus II software automatically selects rx_clkout/tx_clkout as the read clock for receiver phase compensation FIFO. If selected, you must drive this port with a clock that is frequency locked to rx_clkout/tx_clkout.	Channel
rx_enapatternalign	Input	Enables word aligner to align to the comma. This port can either be edge or level sensitive based on the word aligner mode. In the double-width mode, this port is only edge-sensitive.	Channel
rx_bitslip	Input	Word aligner bit-slip control. The word aligner slips a bit of the current word boundary every rising edge of this signal.	Channel
rx_rlv	Output	Run-length violation indicator. A high pulse is given when the run length has detected a violation.	Channel
rx_byteorderalignstatus	Output	From byte ordering block. A high pulse is given when the byte ordering block has successfully aligned the bytes of the PCS output.	Channel
pipe8b10binvpolarity	Input	Physical Interface for PCI Express (PIPE) polarity inversion at the 8B/10B decoder input. This port inverts the data at the input to the 8B/10B decoder.	Channel

Table 2–1. Stratix II GX ALT2GXB Ports (Part 2 of 7)

Port Name	Input/Output	Description	Scope
pipestatus	Output	PIPE receiver status port. In case of multiple status signals, the lower number signal takes precedence. 000 — Received data OK 001 — 1 skip added (not supported) 010 — 1 skip removed (not supported) 011 — Receiver detected 100 — 8B/10B decoder error 101 — Elastic buffer overflow 110 — Elastic buffer underflow 111 — Received disparity error.	Channel
pipephydonestatus	Output	PIPE indicates a mode transition completion—power transition and <code>rx_detect</code> . A pulse is given.	Channel
rx_pipedatavalid	Output	PIPE valid data indicator on the <code>rx_dataout</code> port.	Channel
pipeelecidle	Output	PIPE signal detect for PCI Express.	Channel
rx_digitalreset	Input	Reset port for the receiver PCS block. This port resets all the digital logic in the receiver channel. The minimum pulse width is two parallel clock cycles.	Channel
rx_bisterr	Output	Built-in self test (BIST) block error flag. This port latches high if an error is detected. Assertion of <code>rx_digitalreset</code> resets the BIST verifier, which clears the error flag.	Channel
rx_bistdone	Output	Built-in self test verifier done flag. This port goes high if the receiver finishes reception of the test sequence.	Channel
rx_ctrlldetect	Output	Receiver control code indicator port. Indicates whether the data at the output of <code>rx_dataout</code> is a control or data word. Used with the 8B/10B decoder.	Channel
rx_errdetect	Output	8B/10B code group violation signal. Indicates that the data at the output of <code>rx_dataout</code> has a code violation or a disparity error. Used with disparity error signal to differentiate between a code group error and/or a disparity error. In addition, in XAUI mode, <code>rx_errdetect</code> is asserted in the corresponding byte position when ALT2GXB substitutes the received data with 9'b1FE because of XAUI protocol violations.	Channel

Table 2–1. Stratix II GX ALT2GXB Ports (Part 3 of 7)

Port Name	Input/Output	Description	Scope
rx_syncstatus	Output	Indicates when the word aligner either aligns to a new word boundary (in single-width mode the rx_patterndetect port is level sensitive), indicates that a resynchronization is needed (in single- or double-width mode the rx_patterndetect is edge sensitive), or indicates if synchronization is achieved or not (in single-width mode, the dedicated synchronization state machine is used).	Channel
rx_disperr	Output	8B/10B disparity error indicator port. Indicates that the data at the output of rx_dataout has a disparity error.	Channel
rx_patterndetect	Output	Indicates when the word aligner detects the alignment pattern in the current word boundary.	Channel
rx_ala2size	Input	Available only in SONET/SDH OC-12 and OC-48 modes to select between one of the following two word alignment options: 0 – 16-bit A1A2 1 – 32-bit A1A1A2A2	Channel
rx_ala2sizeout	Output	Available only in SONET/SDH OC-12 and OC-48 modes to indicate one of the following two word alignment options: 0 – 16-bit A1A2 1 – 32-bit A1A1A2A2	Channel
rx_invpolarity	Input	Available in all modes except (OIF) CEI PHY. Inverts the polarity of the received data at the input of the word aligner.	Channel
rx_revbitorderwa	Input	Available in Basic mode with bit-slip word alignment or dynamic reconfiguration enabled. Reverses the bit-order of the received data at a byte level at the output of the word aligner.	Channel
rx_revbyteorderwa	Input	Available in Basic double-width mode only. Swaps the MSByte and LSByte of the 16/20-bit data at the output of the word aligner.	Channel
rx_enabyteord	Input	Available in modes with byte ordering block enabled. Triggers the byte ordering block to perform byte alignment.	Channel
debug_rx_phase_comp_fifo_error	Output	Indicates receiver phase compensation FIFO overrun or underrun situation.	Channel

Table 2–1. Stratix II GX ALT2GXB Ports (Part 4 of 7)

Port Name	Input/Output	Description	Scope
Receiver Physical Media Attachment (PMA)			
rx_pll_locked	Output	Receiver PLL locked signal. Indicates if the receiver PLL is phase locked to the CRU reference clock.	Channel
rx_analogreset	Input	Receiver analog reset. Resets all analog circuits in the receiver PMA.	Channel
rx_freqlocked	Output	CRU mode indicator port. Indicates if the CRU is locked to data mode or locked to the reference clock mode. 0 – Receiver CRU is in lock-to-reference clock mode 1 – Receiver CRU is in lock-to-data mode	Channel
rx_signaldetect	Output	Signal detect port. In PIPE mode, indicates if a signal that meets the specified range is present at the input of the receiver buffer. In all other modes, rx_signaldetect is forced high and must not be used as an indication of a valid signal at receiver input.	Channel
rx_serialpbken	Input	Serial loopback control port. 0 – normal data path, no serial loopback 1 – serial loopback	Channel
rx_locktodata	Input	Lock-to-data control for the CRU. Use with rx_locktorefclk.	Channel
rx_locktorefclk	Input	Lock-to-reference lock mode for the CRU. Use with rx_locktodata. rx_locktodata/rx_locktorefclk 0/0 – CRU is in automatic mode 0/1 – CRU is in lock-to-reference clock 1/0 – CRU is in lock-to-data mode 1/1 – CRU is in lock-to-data mode	Channel
rx_cruclk	Input	Receiver PLL/CRU reference clock.	Channel
Transmitter PCS			
tx_datain	Input	Transmitter parallel data input. The bus width is the channel width multiplied by the number of channels in the instance.	Channel
tx_clkout	Output	PLD logic array clock from the transceiver to the PLD. In a single-channel mode, there is one tx_clkout per channel.	Channel

Table 2–1. Stratix II GX ALT2GXB Ports (Part 5 of 7)

Port Name	Input/Output	Description	Scope
tx_coreclk	Input	Optional write clock port for the transmitter phase compensation FIFO. If not selected, Quartus II software automatically selects tx_clkout as the write clock for transmitter phase compensation FIFO. If selected, you must drive this port with a clock that is frequency locked to tx_clkout.	Channel
tx_detectrxloopback	Input	PIPE receiver detect / loopback pin. Depending on the power-down state the signal either activates receiver detect or loopback.	Channel
tx_forceelecidle	Input	PIPE Electrical Idle mode.	Channel
tx_forcedispcompliance	Input	PIPE forced negative disparity port for transmission of the compliance pattern. The pattern requires starting at a negative disparity. Assertion of this port at the first byte ensures that the first byte has a negative disparity. This port must be deasserted after the first byte.	Channel
powerdn	Input	PIPE power mode port. This port sets the power mode of the associated PCI Express channel. The power modes are as follows: 2'b00: P0 – Normal operation 2'b01: P0s – Low recover time latency, power saving state 2'b10: P1 – Longer recovery time (64 us max) latency, lower power state 2'b11: P2 – Lowest power state	Channel
tx_digitalreset	Input	Reset port for the transmitter PCS block. This port resets all the digital logic in the transmit channel. The minimum pulse width is two parallel clock cycles.	Channel
tx_ctrlenable	Input	Transmitter control code indicator port. Indicates whether the data at the tx_datain port is a control or data word. This port is used with the 8B/10B encoder.	Channel
tx_forcedisp	Input	Available in Basic mode with 8B/10B encoding enabled. Forces positive or negative disparity on the current symbol depending on the tx_dispval signal level.	Channel
tx_dispval	Input	Available in Basic mode with 8B/10B encoding enabled. A high forces negative starting running disparity on the current symbol and a LOW forces positive starting running disparity on the current symbol, provided tx_forcedisp signal is asserted.	Channel

Table 2–1. Stratix II GX ALT2GXB Ports (Part 6 of 7)

Port Name	Input/Output	Description	Scope
tx_invpolarity	Input	Available in all modes except (OIF) CEI PHY. Inverts the polarity of the data to be transmitted at the transmitter PCS-PMA interface (input to the serializer).	Channel
debug_tx_phase_comp_fifo_error	Output	Indicates transmitter phase compensation FIFO overrun or underrun situation.	Channel
Transmitter PMA			
fixedclk	Input	2.5-125 MHz clock for Adaptive Equalization (AEQ) feature. 125-MHz clock for receiver detect functionality in PCI Express (PIPE) mode.	Channel
Central Control Unit (CCU)			
rx_channelaligned	Output	10-Gigabit Attachment Unit Interface (XAUI) deskew FIFO aligned flag. This signal goes high after the channel aligner acquires channel alignment per the IEEE 802.3ae specification.	Transceiver block
coreclkout	Output	×4 mode output. This is the clock output from the central clock generation block. In ×8 mode, the central clock generator block from the lower transceiver generates this clock. For use with XAUI, PCI Express, ×4, and ×8 modes.	Transceiver block
reconfig_clk	Input	Input reference clock for the dynamic reconfiguration controller. The frequency range of this clock is 2.5 MHz to 50 MHz. The assigned clock uses global resources by default. This same clock should be connected to ALT2GXB.	
reconfig_togxb	Input	From reconfiguration controller for dynamic reconfiguration.	Transceiver block
reconfig_fromgxb	Output	To reconfiguration controller.	Transceiver block
aeq_togxb	Input	From reconfiguration controller for Adaptive Equalization.	Transceiver block
aeq_fromgxb	Output	To reconfiguration controller for Adaptive Equalization.	Transceiver block
CMU PMA			
gxb_powerdown	Input	Transceiver block reset and power down. This resets and powers down all circuits in the transceiver block. This does not affect the REFCLK buffers and reference clock lines.	Transceiver block

Table 2–1. Stratix II GX ALT2GXB Ports (Part 7 of 7)			
Port Name	Input/Output	Description	Scope
pll_locked	Output	PLL locked indicator for the transmitter PLLs.	Transceiver block
pll_inclk	Input	Reference clocks for the transmitter PLLs.	Transceiver block
Calibration Block			
cal_blk_clk	Input	Calibration clock for the transceiver termination blocks. This clock supports frequencies from 10 MHz to 125 MHz.	Device
cal_blk_powerdown (active_low)	Input	Power-down signal for the calibration block. Assertion of this signal may interrupt data transmission and reception. Use this signal to recalibrate the termination resistors if temperature and/or voltage changes warrant it.	Device
External Signals			
tx_dataout	Output	Transmitter serial output port.	Channel
rx_datain	Input	Receiver serial input port.	Channel
rrefb (1)	Output	Reference resistor port. This port is always used and must be tied to a 2K-Ω resistor to ground. This port is highly sensitive to noise. There must be no noise coupled to this port.	Device
refclk (1)	Input	Dedicated reference clock inputs (two per transceiver block) for the transceiver. The buffer structure is similar to the receiver buffer, but the termination is not calibrated.	Transceiver block
gxb_enable	Input	Dedicated transceiver block enable pin. If instantiated, this port must be tied to the pll_ena input pin. A high level on this signal enables the transceiver block; a low level disables it.	Transceiver block

Note to Table 2–1:

(1) These are dedicated pins for the transceiver and do not appear in the MegaWizard® Plug-In Manager.



This chapter uses “transceiver block number” and “transceiver bank number” interchangeably. [Table 2-2](#) maps transceiver block number to the Stratix II GX transceiver bank number.

Table 2-2. Transceiver Block Number to Transceiver Bank Number Mapping

Transceiver Block Number	Transceiver Bank Number
0	13
1	14
2	15
3	16
4	17

Transmitter Modules

This section describes the Stratix II GX transceiver’s transmitter path. This section describes the following modules:

- Clock multiplier unit (CMU)
- Transmitter phase compensation FIFO buffer
- Byte deserializer
- 8B/10B encoder
- Serializer
- Transmitter buffer

Clock Multiplier Unit

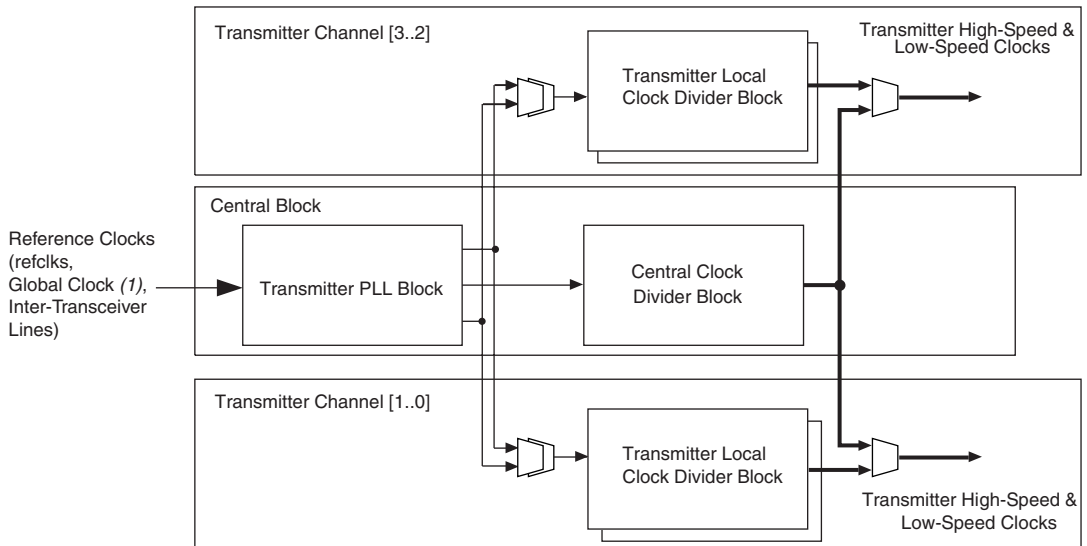
The CMU in Stratix II GX devices takes the reference clock from either the PLD or the dedicated reference clock inputs (`refclk0` and `refclk1`) and synthesizes the clocks that are used for the transmitter logic, serializer, receiver PLL reference clock, and PLD clocks.

Each transceiver block has its own CMU block that is further divided into three CMU sub blocks:

- Transmitter PLL block
- Central clock divider block
- Transmitter local clock divider block

The transmitter PLL block and central clock divider blocks are located in the central block of the transceiver block. A transmitter local clock divider block is located in each transmitter of the transceiver block. Each transceiver block has a dedicated CMU block and two dedicated reference clock inputs that feed the CMU (refer to [Figure 2-2](#)).

Figure 2–2. Clock Multiplier Unit Block Diagram



Note to Figure 2–2:

- (1) The Global Clock line must be driven by an input pin.

The Quartus® II software simplifies the CMU settings. It sets most of the settings automatically for protocol modes; for example, PLL multiplication factors. You need provide only the data rate in the ALT2GXB MegaWizard Plug-In Manager and then select the input clock frequency.

Dedicated Reference Clock Pin Specifications

Table 2–3 shows the I/O standards allowed for the reference clock pins.

Table 2–3. Reference Clock Specifications (Part 1 of 2)			
Protocol	I/O Standard	Coupling	Termination
Basic, XAUI, GIGE, SONET/SDH, (OIF) CEI PHY, Serial RapidIO, SDI, CPRI	1.2-V PCML, 1.5-V PCML, 3.3-V PCML, Differential LVPECL, LVDS	AC	On-chip

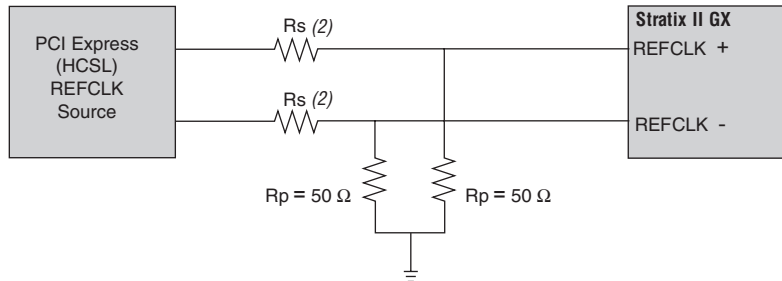
Table 2–3. Reference Clock Specifications (Part 2 of 2)

Protocol	I/O Standard	Coupling	Termination
PIPE	1.2-V PCML, 1.5-V PCML, 3.3-V PCML, Differential LVPECL, LVDS	AC	On-chip
	HCSL (1)	DC	External (2)

Notes to Table 2–3:

- (1) In PIPE mode, you have the option of selecting the HCSL standard for the reference clock if compliance to PCI Express is required. The Quartus II software automatically selects DC coupling with external termination for the REFCLK signal if configured as HCSL.
- (2) Refer to Figure 2–3 for an example termination scheme.

Figure 2–3 shows an example termination scheme for a reference clock signal when configured as HCSL.

Figure 2–3. DC Coupling and External Termination Scheme for PCI Express Reference Clock Notes (1), (2)**Notes to Figure 2–3:**

- (1) No biasing is required if the reference clock signals are generated from a clock source that conforms to the PCI Express specification.
- (2) Select resistor values as recommended by the PCI Express clock source vendor.

Transmitter PLL Block

The transmitter PLL block contains two transmitter PLLs (transmitter PLL0 and transmitter PLL1) per transceiver block, as shown in Figure 2–4. The transmitter PLL block multiplies the reference clock to the frequency required to support the serial data rate. Transmitter PLL0 and transmitter PLL1 can support data rates from 600 Mbps to 6.375 Gbps. Each PLL has a dedicated locked signal (p1l1_locked) that is fed to the PLD logic array to indicate when the PLLs are locked to the reference clock.

You can use transmitter PLL0 and transmitter PLL1 individually (one PLL active at a time) to provide a base high-speed clock to the entire transceiver block, or simultaneously to provide support for the different data rates within the transceiver block that does not have a common base reference clock frequency. For example, one PLL can support a 1.25 Gbps data rate with a 125 MHz reference clock and the other PLL can support a 2.488 Gbps data rate with a 62.2 MHz reference clock.

You can use up to two reference clocks for the transmitter PLLs in a single transceiver block at any given time. The reference clocks can come from the following:

- Dedicated reference clock pins of the associated transceiver blocks (two total per transceiver block)
- PLD clock network (one per transceiver block, must be connected directly from an input clock pin and cannot be driven by user logic or enhanced PLL)
- Inter-transceiver lines (up to five total, one from each transceiver block)



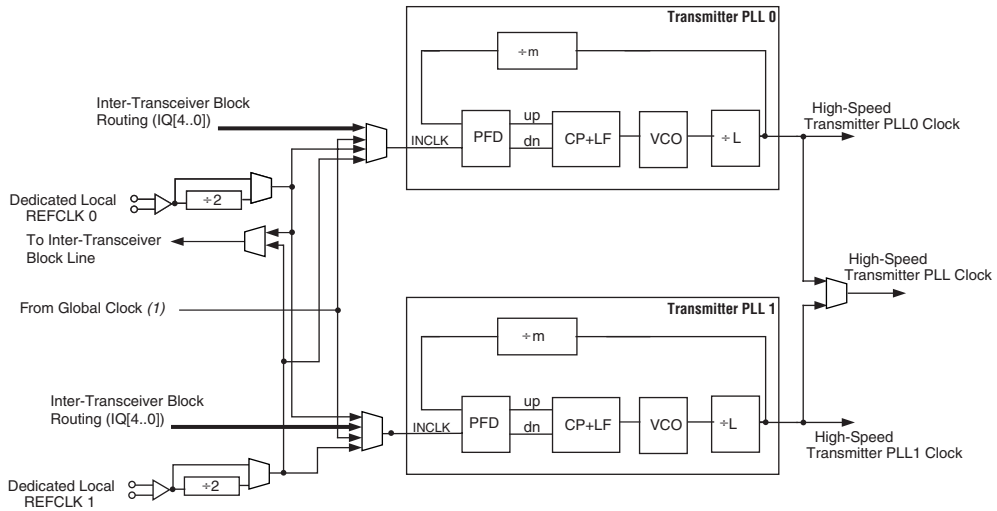
If you assign an I/O or a non-REFCLK clock pin to provide clock ONLY for the `p11_inclk/rx_cruc1k` ports of the transceiver, the Quartus II software requires the following setting for the clock source in the assignment editor for successful compilation:

Assignment name: Stratix II GX REFCLK and termination setting

Value: Use as regular I/O.

Figure 2–4 shows the transmitter PLL block.

Figure 2–4. Transmitter PLL Block



Note to Figure 2–4:

- (1) The Global Clock line must be driven by an input pin.

Transmitter PLLs

There are two transmitter PLLs in each CMU (transmitter PLL0 and transmitter PLL1). Transmitter PLL0 and transmitter PLL1 receive the reference clock from one of five inter-transceiver lines (refer to “[Inter-Transceiver Line Routing](#)” on page 2–19 for more information), a global PLD clock driven from a clock input pin, or from the dedicated reference clock REFCLK0 or REFCLK1 (both reference clock pins can drive either transmitter PLL0 or transmitter PLL1). You can divide the reference clocks from the REFCLK pins by two to support higher reference clock frequencies.

Transmitter PLL0 and transmitter PLL1 have half-rate VCOs that operate at half the rate of the serial data stream. The range of these VCOs are from 500 MHz to 3.1875 GHz to support a native data rate of 1 Gbps to 6.375 GHz. Lower data rates (600 Mbps to 1 Gbps) are supported via additional clock dividers (refer to “[Clock Synthesis](#)” on page 2–16 for more information).

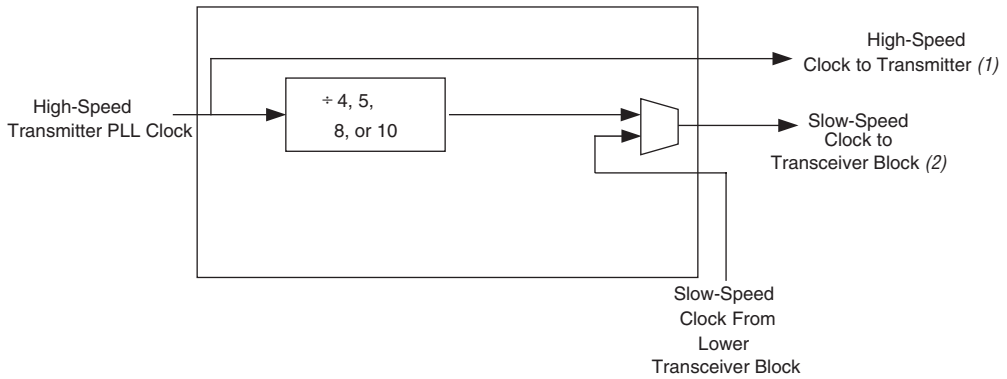
The PLL contains two multiplier blocks in the PLL feedback loop to multiply the reference clock to support the required data rate. The Quartus II software automatically selects the values for all the dividers. You must input a data rate and select the input clock frequency.

The PLL output feeds the central clock divider block through the high-speed transmitter PLL clock multiplexer or feeds the transmitter local clock divider block in each transmitter channel through the high-speed transmitter PLL clocks.

Central Clock Divider Block

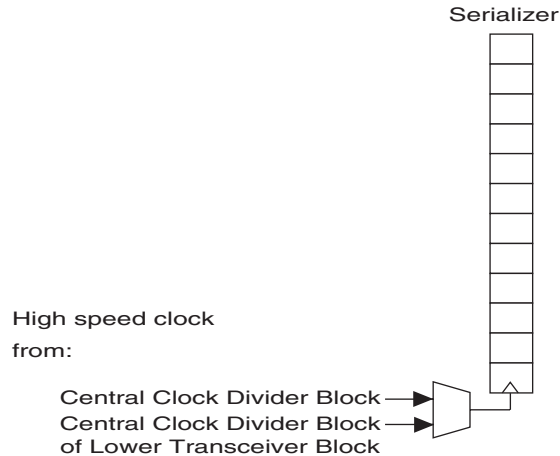
The central clock divider block is located in the central block of the transceiver block (refer to [Figure 2-2](#)). This block provides the high-speed clock for the serializer and the low-speed clock for the transceiver's PCS logic within the transceiver block in a four-lane mode. In Physical Interface for PCI Express (PIPE) $\times 8$ mode, the central clock divider block also provides the high-speed clock and low-speed clock for the adjacent upper transceiver block and provides the high- and low-speed clocks to the associated transceiver block. The PLLs, central clock divider block, and the transmitter local clock dividers are powered down in the adjacent upper transceiver block in an eight-lane configuration.

[Figure 2-5](#) shows the central clock divider block. The /4, /5, /8, and /10 block generates the slow-speed clock based on the serialization factor. In an eight-lane configuration in PIPE mode, the slow-speed clock is multiplexed from the lower transceiver block. The high-speed clock goes directly into each channel's serializer through a clock multiplexer.

Figure 2-5. Central Clock Divider Block**Notes to Figure 2-5:**

- (1) Feeds the transmitter within the transceiver and the above adjacent transceiver block.
- (2) Feeds the PCS logic.

Figure 2-6 shows the clock selection for the serializer.

Figure 2-6. Serializer High-Speed Clock Connection

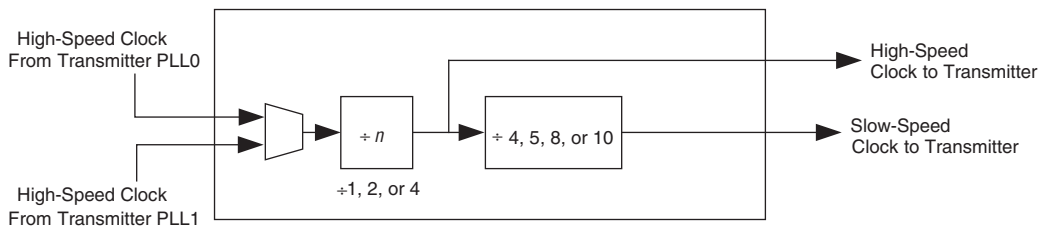
The central clock divider block feeds all the channels in the transceiver block and, in PIPE $\times 8$ mode, it also feeds the adjacent upper transceiver block. This ensures that the serializer in each channel outputs the same bit number at the same time and minimizes the channel-to-channel skew.

Transmitter Local Clock Divider Block

The TX local clock divider blocks are located in each transmitter channels of the transceiver block. The purpose of this block is to provide the high-speed clock for the serializer and the low-speed clock for the transmitter data path and the PLD for all the transmitters within the transceiver block. This allows for each of the transmitter channels to run at different rates. The $/n$ divider offers a $/1$, $/2$, and $/4$ factors to provide capability to reduce base frequency of the driving PLL to a half or a quarter rate. This allows each transmitter channel to run at a $/1$, $/2$, or $/4$ of the original data rate.

Figure 2–7 shows the transmitter local clock divider block.

Figure 2–7. Transmitter Local Clock Divider Block



Each transmitter local clock divider block is operated independently, so there is no guarantee that each channel sends out the same bit at the same time.

Clock Synthesis

Each PLL in a transceiver block receives a reference clock and generates a high-speed clock that is forwarded to the clock generator blocks. There are two types of clock generators:

- the transmitter local clock divider block
- the central clock divider block

The transmitter local clock divider block resides in the transmit channel and synthesizes the high-speed serial clock (used by the serializer) and slow-speed clock (used by the transmitter's PCS logic). The central clock divider block resides in the transceiver block outside the transmit or receive channels. This block synthesizes the high-speed serial clock (used by the serializer) and slow-speed clock (used by the transceiver block PCS logic—transmitter and receiver (if the rate matcher is used). The PLD

clock is also supplied by the central clock divider block and goes through the divide-by-two block (located in the central block of the transceiver block) if the byte serializer/deserializer is used.

The PLLs in the transceiver have half rate VCOs that run at half the rate of the data stream. When in the individual channel mode, the slow-speed clocks for the transmitter logic and the serializer need only be / 4 or a / 5 divider to support a $\times 8$ and $\times 10$ serialization factor. The $\times 16$ and $\times 20$ serialization factor is supported by the / 8 and / 10 clock divider. [Table 2–4](#) shows the divider settings for achieving the available serialization factor.

Serialization Factor	Divider Setting
$\times 8$	/ 4
$\times 10$	/ 5
$\times 16$	/ 8
$\times 20$	/ 10

In the four-lane mode, the central clock divider block supplies all the necessary clocks for the entire transceiver block.

The reference clock ranges from 50 MHz to 622.08 MHz. The phase frequency detector (PFD) has a minimum frequency limit of 50 MHz and a maximum frequency limit of 325 MHz.

The `refclk` pre-divider (/2) is available if you use the dedicated `refclk` pins for the input reference clock. The `refclk` pre-divider is required if **one** of the following conditions is satisfied:

1. If the input clock frequency is greater than 325 MHz.
2. For functional modes with a data rate less than 3.125 Gbps (the data rate is specified in the **what is the data rate?** option in the **General** tab of the ALT2GXB MegaWizard):
 - If the input clock frequency is greater than or equal to 100 MHz AND
 - If the ratio of data rate to input clock frequency is 4, 5, or 25

3. For functional modes with an data rate greater than 3.125 Gbps:
 - If the input clock frequency is greater than or equal to 100 MHz AND
 - If the ratio of data rate to input clock frequency is 8, 10, or 25

Table 2–5 shows the `refclk` pre-divider and the available PLL multiplication factors.

Transmitter PLL Reference Clock Source	Reference Clock Pre-Divider	/ M (2)	/ L
Inter-transceiver routing	1, 2 (1)	1, 4, 5, 8, 10, 16, 20, 25	1, 2, 4
Dedicated local reference clock	1, 2	1,4,5,8,10,16,20, 25	1, 2, 4

Notes to Table 2–5:

- (1) The / 2 is achieved by using the pre-divider on the driving REFCLK pin.
- (2) The M, L counters are automatically selected by the Quartus II software based on the selected data rate and protocol and the reference clock frequency.

Table 2–6 shows the multiplication values for Basic mode.

Protocol Functional Mode	Reference Clock Pre-Divider	/ M
Basic single width	1, 2	4, 5, 8, 10, 16, 20, 25
Basic double width (1 Gbps to 3.125 Gbps)	1, 2	4, 5, 8, 10, 16, 20, 25
Basic double width (> 3.125 Gbps to 6.375 Gbps)	1, 2	8, 10, 16, 20, 25

Transmitter PLL Bandwidth Setting

The Stratix II GX transmitter PLLs in the transceiver offer a programmable bandwidth setting. The bandwidth of a PLL is the measure of its ability to track the input clock and jitter. It is determined by the –3dB frequency of the closed-loop gain of the PLL.

There are three bandwidth settings: high, medium, and low. The high bandwidth setting filters out internal noise from the VCO because it tracks the input clock above the frequency of the internal VCO noise. With the low bandwidth setting, if the noise on the input reference clock

is greater than the internal noise of the VCO, the PLL filters out the noise above the -3dB frequency of the closed-loop gain of the PLL. The medium bandwidth setting is a compromise between the high and low settings.

The -3dB frequencies for these settings can vary because of the non-linear nature and frequency dependencies of the circuit.

Inter-Transceiver Line Routing

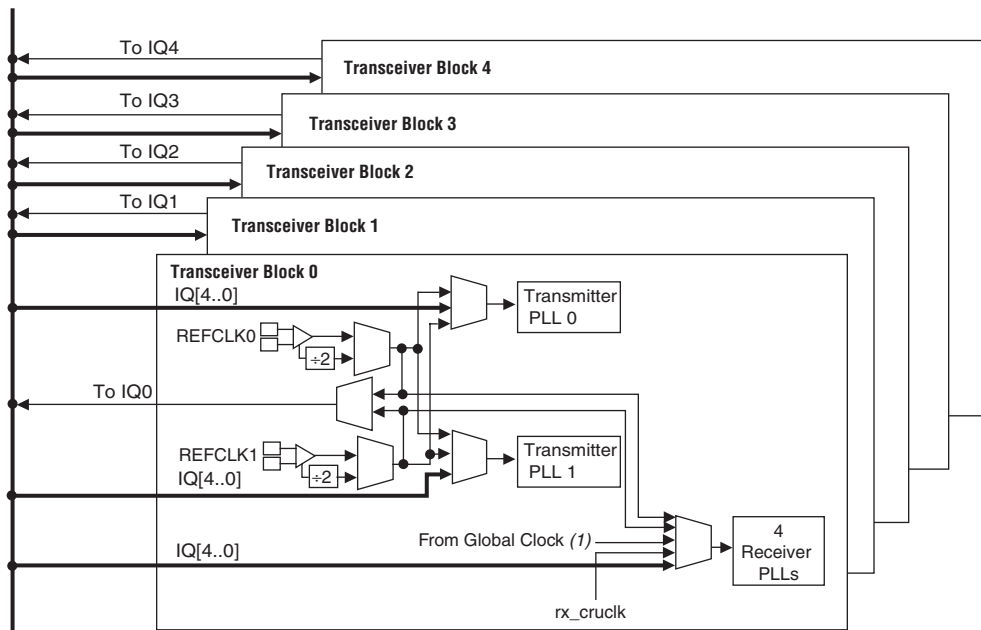
The inter-transceiver lines allow the REFCLK of one transceiver block to drive other transceiver blocks. There are a maximum of five inter-transceiver clock routing lines available in each device in the Stratix II GX family. Each transceiver block can drive one inter-transceiver line from either one of its associated REFCLK pins. The inter-transceiver lines can drive any or all of the transmitter PLLs and receiver PLLs in the device. The inter-transceiver lines offer flexibility when multiple channels in separate transceiver blocks share a common reference clock frequency. The inter-transceiver lines only distribute the reference clock and cannot be used to bond channels because each PLL and clock dividers operate independently. If you select an input reference clock frequency such that it requires the use of the REFCLK pre-divider, you cannot use the clock on REFCLK pins to drive PLD logic.

The inter-transceiver lines also drive the reference clock from the REFCLK pins into the PLD, which reduces the need to drive multiple clocks of the same frequency into the device.

The Quartus II software automatically uses the appropriate inter-transceiver line if the transceiver block is being clocked by the dedicated reference clock (REFCLK) pin of another transceiver block.

Figure 2-8 shows the inter-transceiver line interface to the transceivers in the gigabit transceiver blocks and to the PLD. The connections of transceiver block 0 are shown. The other connections are the same, with the exception of the inter-transceiver line number that the transceiver block drives.

Figure 2–8. Inter-Transceiver Lines of a Five Transceiver Block Device (2SGX130G)



Note to Figure 2–8:

(1) The Global Clock line must be driven by an input pin.

Transceiver Clock Distribution

This section describes single-lane, four-lane, and eight-lane configurations for the high-speed and low-speed transceiver clocks. All protocol support falls in the single-lane configuration except for the four-lane and eight-lane PIPE mode and XAUI. The four-lane PIPE mode uses the four-lane configuration. The eight-lane PIPE mode uses the eight-lane configuration.

Single Lane

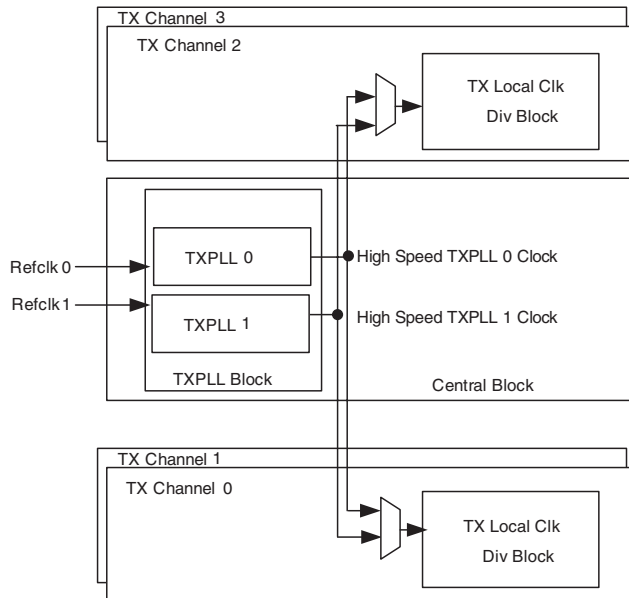
In a single-lane configuration, the PLLs in the central block supply the high speed clock and the clock generation blocks in each transmitter channel divides down that clock to the frequency needed to support its particular data rate. In this configuration, two separate clocks can be supplied through the central block to provide support for two separate base frequencies. The transmitter clock generation blocks can divide those down to create additional frequencies for specific data rate

requirements. Each of the four transmitter channels can operate at a different data rate with the use of the individual transmitter local clock dividers and both Transmitter PLL0 and Transmitter PLL1.



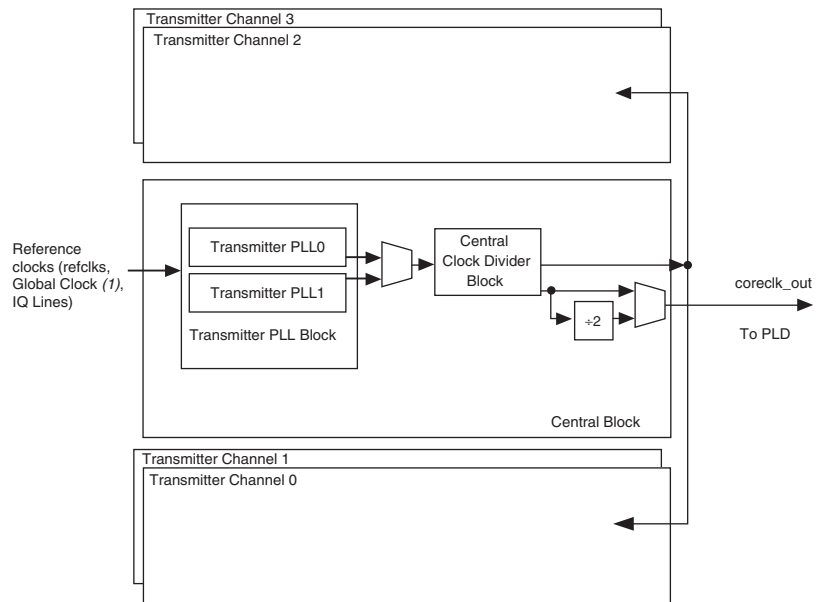
If you instantiate four channels and are not in PIPE $\times 4$, XAUI, or Basic single-width mode with $\times 4$ clocking, the Quartus II software automatically chooses the single-lane configuration.

Figure 2–9. Clock Distribution for Individual Channel Configuration



Four-Lane Mode

In a four-lane configuration (Figure 2–10), the central block generates the parallel and serial clocks that feed the transmitter channels within the transceiver. All channels in a transceiver must operate at the same data rate. This configuration is only supported in PIPE $\times 4$, XAUI and Basic single-width mode with $\times 4$ clocking.

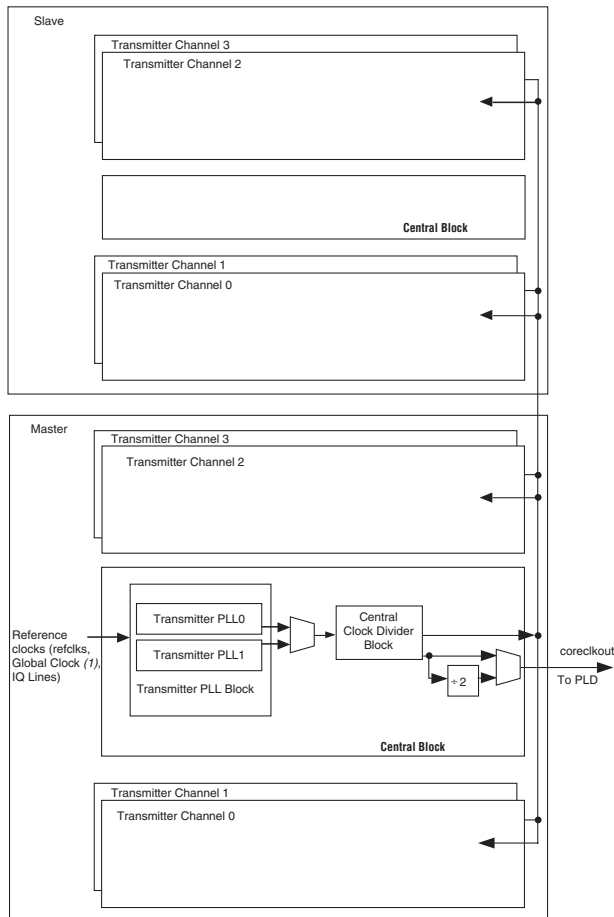
Figure 2–10. Clock Distribution for a Four-Lane Configuration**Note to Figure 2–10:**

(1) The Global Clock line must be driven by an input pin.

Eight-Lane Mode

The eight-lane mode (refer to Figure 2–11) is reserved for PIPE $\times 8$ use only. The central block of the lower transceiver supplies the parallel and serial clocks for all eight transmitter channels. The clock distribution uses a dedicated eight-lane clocking routing that offers low skew for transmitter channel-to-channel skew specification. The high- and low-speed clocks are forwarded using this dedicated eight-lane clocking tree. The central block of the upper transceiver block and all the transmitter clock generation blocks are unused and are powered down in this mode. The clock to the PLD (`coreclkout`) is generated by the central clock generation block of the master transceiver block (the lower transceiver block).

Figure 2–11. Clock Distribution for Eight-Lane Mode

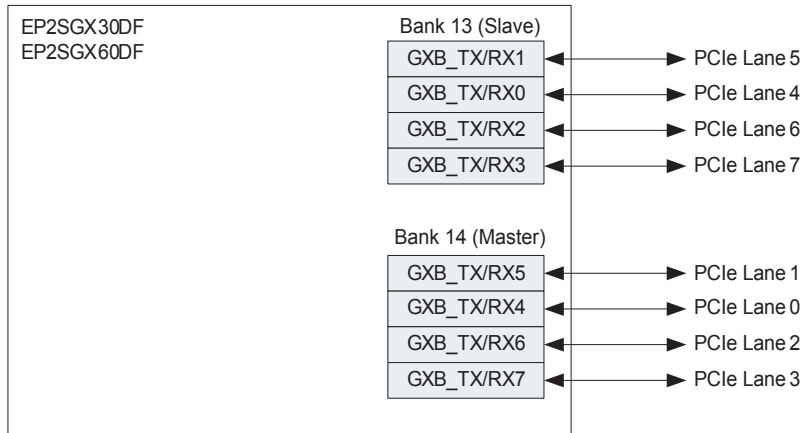
**Note to Figure 2–11:**

- (1) The Global Clock line must be driven by an input pin.

Only designated lower transceiver blocks can be used as a master (transceiver blocks 1 and 3), and designated upper transceiver blocks (transceiver blocks 0 and 2) can be used as a slave as long as they are coupled to the lower master transceiver block. The Quartus II software automatically utilizes the correct transceiver blocks in a $\times 8$ mode if you do not assign placement. If you do not place the master and slave transceiver blocks accordingly (through pin assignments), a no fit error occurs.

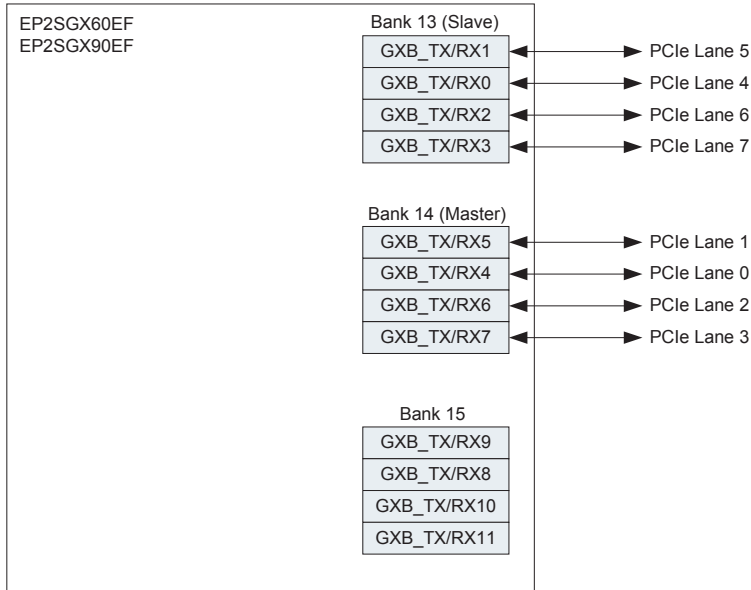
Single transceiver block devices (EP2SGX30C and EP2SGX60C) cannot be used for PCI-E x8 mode. [Figure 2-12](#) shows how double transceiver block devices (EP2SGX30D and EP2SGX60D) are configured for PCI-E x8 mode.

Figure 2-12. Two Transceiver Block Device With One x8 PCI-E Link



The three transceiver block devices (EP2SGX60E and EP2SGX90E) support only one PCI-E $\times 8$ link. Figure 2-13 shows the PCI-E $\times 8$ configuration.

Figure 2-13. Three Transceiver Block Device With One $\times 8$ PCI-E Link *Note (1)*

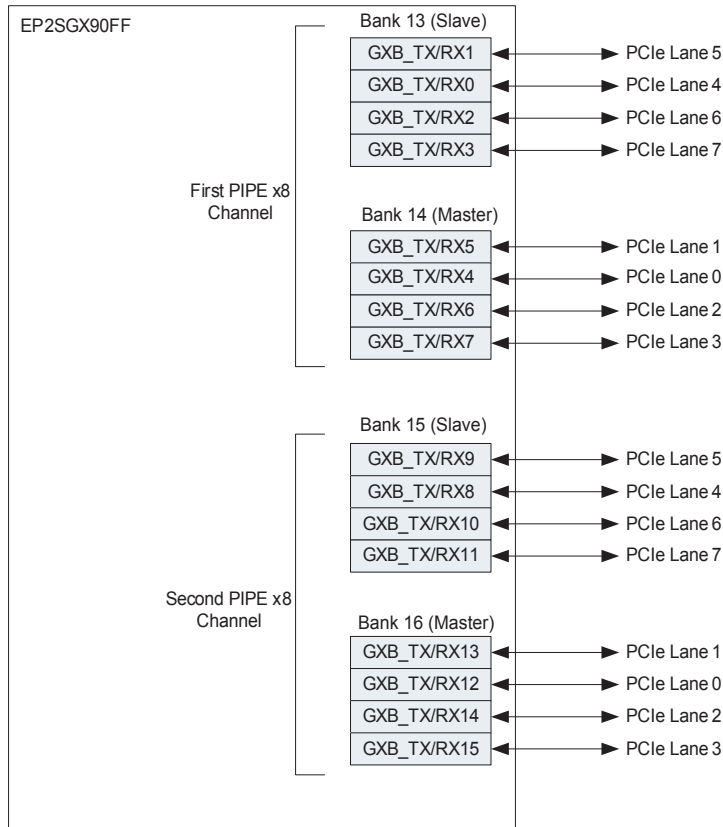


Note to Figure 2-13:

- (1) Transceiver Bank 15 can be active and used to support other protocols.

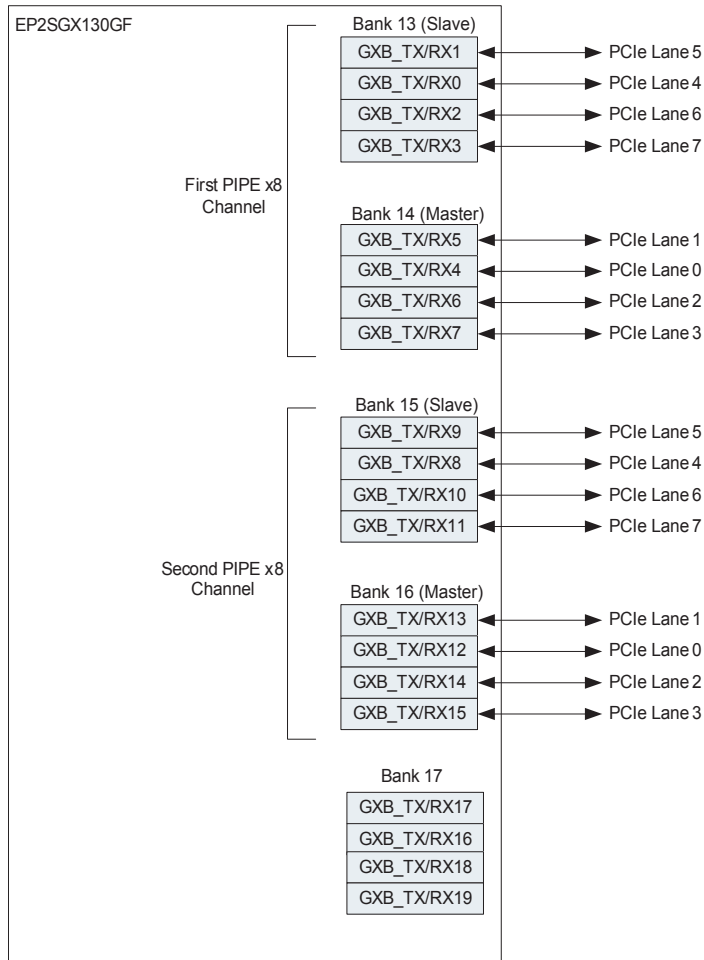
A four transceiver block device (EP2SGX90F) supports up to two PCI-E $\times 8$ links (refer to Figure 2-14). The transceiver block pairs are blocks 0 and 1 and blocks 2 and 3. If only one PCI-E $\times 8$ link is used, the other transceiver blocks can be active and be used to support other protocols.

Figure 2-14. Four Transceiver Block Device With Two $\times 8$ PCI-E Links



A five transceiver device (EP2SGX130G) supports up to two PCI-E $\times 8$ links (refer to [Figure 2-15](#)). The transceiver block pairs are the same as in the four transceiver device—blocks 0 and 1 and blocks 2 and 3. Block 4 is not used for PCI-E $\times 8$ mode. If any transceiver blocks are not used in the PCI-E $\times 8$ mode, they can be used to support any other protocol.

Figure 2-15. Five Transceiver Block Device With Two $\times 8$ PCI-E Links *Note (1)*



Note to [Figure 2-15](#):

(1) Transceiver Bank 17 can be active and used to support other protocols.

Channel Clock Distribution

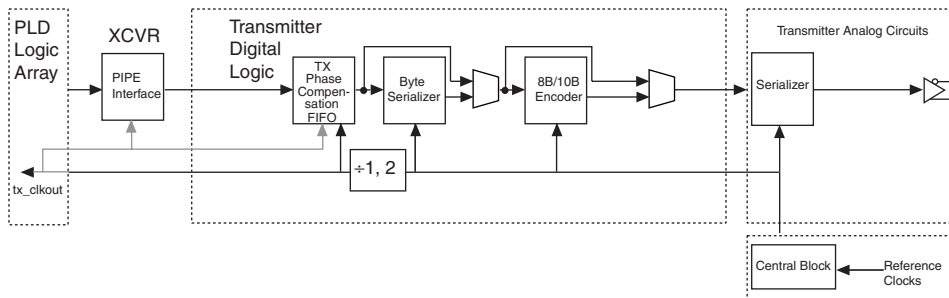
This section describes clocking within each channel for:

- Individual channels in Basic (without $\times 4$ clocking enabled), SONET/SDH, PIPE $\times 1$, GIGE, (OIF) CEI PHY Interface (with low-jitter option disabled), Serial RapidIO, SDI, and CPRI modes
- Bonded channels in XAUI, PIPE $\times 4$, PIPE $\times 8$, Basic (with $\times 4$ clocking enabled), and (OIF) CEI PHY Interface (with low-jitter option enabled) modes

Individual Channels Clocking

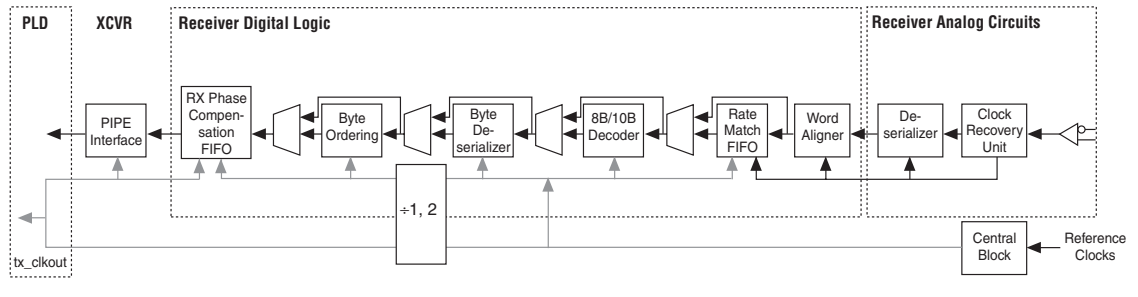
In individual channel modes, the transmitter logic is clocked by the slow-speed clock from the clock divider block. The transmitter phase compensation FIFO buffer and the PIPE interface (in PIPE mode) are clocked by the `tx_clkout` clock of the channel that is fed back to the transmitter channel from the PLD logic. [Figure 2–16](#) shows the clock routing for the transmitter channel.

Figure 2–16. Individual Channel Transmitter Logic Clocking

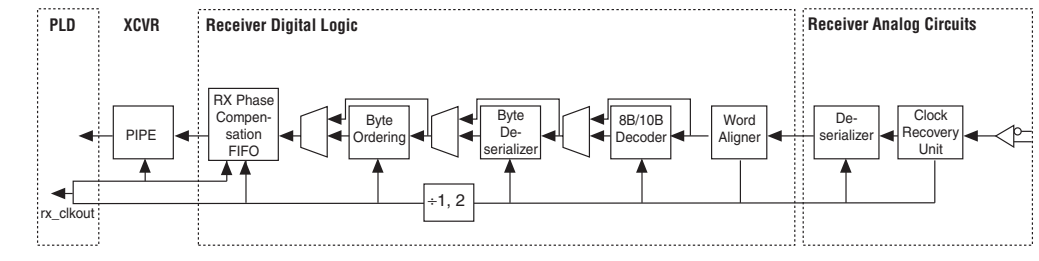


The receiver logic clocking has two clocking methods: one when rate matching is used and the other when rate matching is not used.

If rate matching is used (PIPE, GIGE, and Basic modes), the receiver logic from the serializer to the rate matcher is clocked by the recovered clock from its associated channel. The rest of the logic is clocked by the slow clock from the clock divider block of its associated channel. The read side of the phase compensation FIFO buffer and the PIPE interface (for PIPE mode) is clocked by the `tx_clkout` fed back through the PLD logic. [Figure 2–17](#) shows the clocking of the receiver logic with the rate matcher.

Figure 2-17. Individual Channel Receiver Logic Clocking With Rate Matching

If rate matching is not used (Basic, SONET/SDH, CPRI, (OIF) CEI PHY Interface, SDI, Serial RapidIO modes), then the receiver logic is clocked by the recovered clock of its associated channel (Figure 2-18). The receiver phase compensation FIFO buffer's read port is clocked by the recovered clock that is fed back from the PLD logic array as `rx_clkout`.

Figure 2-18. Individual Channel Receiver Logic Clocking Without Rate Matching

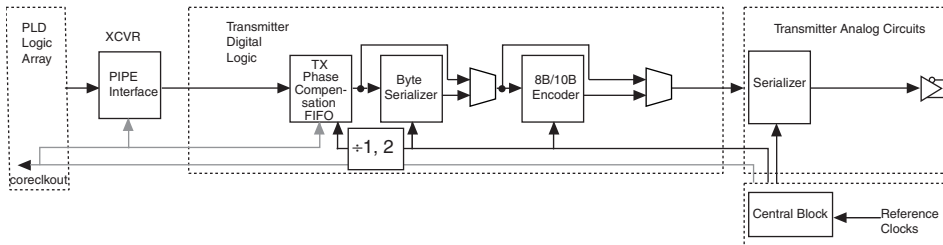
Transmitter Clocking (Bonded Channels)

The clocking in bonded channel modes (Figure 2-19) is different from that of the individual channel. All the transmitters are synchronized to the same transmitter PLL and clock divider from the central block. In $\times 4$ bonded channel modes, the central clock divider of the transceiver block clocks all 4 channels. In PIPE $\times 8$ bonded channel mode, the central clock divider of the master transceiver block clocks all 8 channels.

The transmitter logic up to the read port of the transmitter phase compensation FIFO buffer is clocked by the slow-speed clock from the central block. The PIPE interface and the write port of the transmitter phase compensation FIFO buffer is clocked by the `coreclkout` signal routed from the PLD. In the PIPE $\times 8$ slave transceiver, the central block of the associated transceiver is not active and the transmitter logic to the read port of the transmitter phase compensation FIFO buffer is clocked by

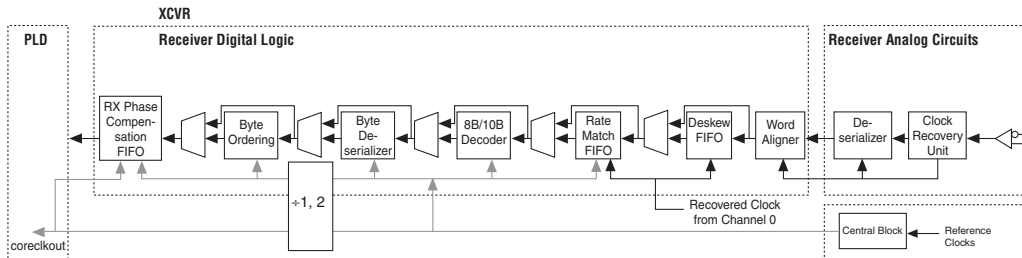
the slow-speed clock from the master transceiver. The PIPE interface and the write port of the transmitter phase compensation FIFO buffer is clocked by the `coreclkout` signal of the master transceiver. The slave transceiver does not output a `coreclkout` signal because the CMU is powered down.

Figure 2–19. Transmitter Channel Clocking in Transceiver Mode



For the receiver logic, in XAUI mode (Figure 2–20), the local recovered clock feeds the logic up to the write clock of the deskew FIFO buffer. The recovered clock from channel [0] feeds the read clock of the deskew FIFO buffer and the write port of the rate matcher. The slow clock from the central block feeds the rest of the logic up to the write port of the phase compensation FIFO buffer. The `coreclkout` signal routed through the PLD from the central block feeds the read side of the phase compensation FIFO buffer.

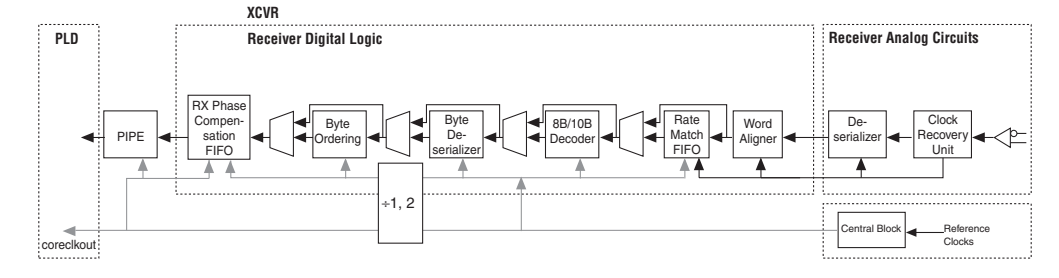
Figure 2–20. Receiver Channel Clocking in XAUI Mode



In the PIPE $\times 4$ and PIPE $\times 8$ modes (Figure 2–21), the local recovered clock feeds the logic up to the write port of the rate matcher FIFO buffer. The slow clock from the central block feeds the rest of the logic up to the write port of the phase compensation FIFO buffer. The `coreclkout` signal routed through the PLD from the central block feeds the read side of the phase compensation FIFO buffer. In PIPE $\times 8$, the slave transceiver takes

the clocks from the central block of the master transceiver. The individual channel clock distribution of the slave transceiver is the same as the master transceiver.

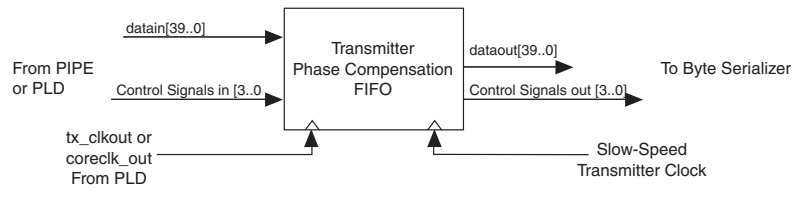
Figure 2–21. Receiver Channel PIPE $\times 4$ and $\times 8$ Modes



Transmitter Phase Compensation FIFO Buffer

The transmitter phase compensation FIFO buffer (Figure 2–22) is located at the device's logic array interface in the transmitter block and compensates for phase difference between the transmitter clock and the clock from the PLD. The transmitter phase compensation FIFO buffer operates in two modes: low latency and high latency. In the low latency mode, the FIFO buffer is four words deep. The Quartus II software chooses low latency mode automatically for every mode except the PCI-Express PIPE mode (which automatically uses high latency mode). In high latency mode, the FIFO buffer is eight words deep.

Figure 2–22. Transmitter Phase Compensation FIFO Buffer



The phase compensation FIFO buffer's read port is clocked by the transmitter PLL clock. The write clock is fed by `tx_clkout` of the associated channel in a single-channel configuration. The FIFO buffer's write clock is clocked by `coreclkout` in the four- or eight-channel configuration.

The transmitter phase compensation FIFO buffer is always used and cannot be bypassed. The input to the transmitter phase compensation FIFO buffer is the data from the PLD logic array or the PIPE interface in PIPE mode.

Transmitter Phase Compensation FIFO Error Flag

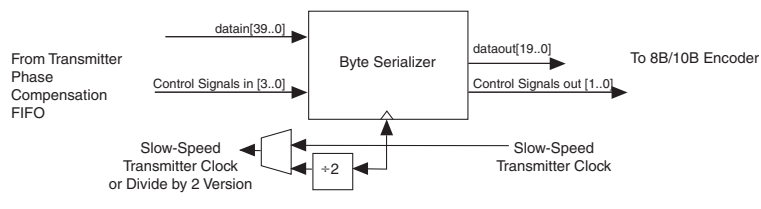
The write port of the transmitter phase compensation FIFO can be clocked by either transmitter PLL output clock (`tx_clkout` or `coreclkout`) or a PLD clock. The read port is always clocked by the transmitter PLL output clock. In all configurations, the write clock and the read clock must have 0 PPM difference to avoid overrun/underflow of the phase compensation FIFO.

An optional `debug_tx_phase_comp_fifo_error` port is available in all modes to indicate transmitter phase compensation FIFO overrun/underflow condition. The `debug_tx_phase_comp_fifo_error` is asserted high when the phase compensation FIFO gets either full or empty. This feature is useful to verify the phase compensation FIFO overrun/underflow condition as a probable cause of link errors.

Byte Serializer

The byte serializer (Figure 2–23) converts the two- or four-byte interface into a one- or two-byte-wide data path for the transceiver from the PLD interface. The PLD interface has a limit of 250 MHz, so the byte serializer is required to stripe the parallel data into the single- or double-wide transceiver data path. At 6.375 Gbps, the transceiver logic has a double-byte-wide data path that runs at 318.75 MHz in a $\times 20$ serializer factor, which is above the maximum PLD interface speed. By using the byte serializer, the PLD interface width doubles to 40-bits (36 bits when using the 8B/10B encoder) and the interface speed drops to 159.375 MHz.

Figure 2–23. Byte Serializer



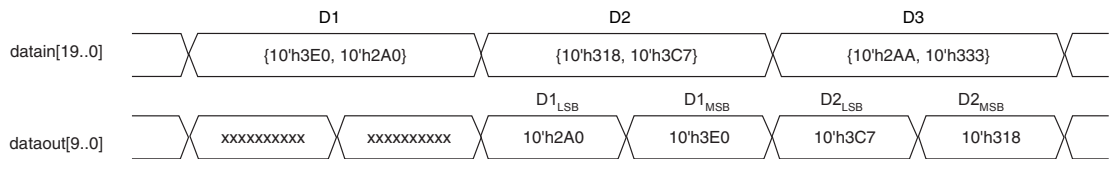
The byte serializer takes in a 40-, 32-, 20-, or 16-bit-wide input from the phase compensation FIFO buffer and serializes it to 20, 16, 10, or 8 bits, respectively (refer to [Table 2-7](#)). At the same time, the clock frequency is doubled.

Input Data Width (Bits)	Output Data Width After Byte Serialization (Bits)
40	20
32	16
20	10
16	8

After serialization, the byte serializer transmits the least significant byte to the most significant byte. Always use the transmitter digital reset to reset the byte serializer FIFO pointers whenever the transmitter PLL loses lock. Refer to [“Reset Control and Power Down”](#) on page 2-214 for further details on the reset sequence.

[Figure 2-24](#) shows byte serializer input and output signals when serializing a 20-bit input to 10 bits. The `tx_datain` signal is the input from the FPGA’s logic array that has already passed through the transmitter phase compensation FIFO buffer.

Figure 2-24. Transmitter Byte Serializer



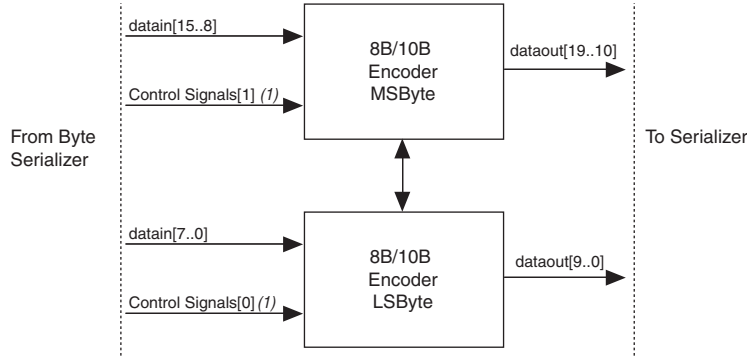
In [Figure 2-24](#), the LSB is transmitted before the MSB in the transmitter byte serializer. For the input of `D1`, the output is `D1LSB` and then `D1MSB`.

8B/10B Encoder

The 8B/10B encoder (refer to [Figure 2-25](#)) is part of the Stratix II GX transceiver digital blocks and lies between the byte serializer and the serializer. The 8B/10B encoder operates in two modes: single-width and double-width and can be bypassed if the 8B/10B encoder is not used. In single-width mode, the 8B/10B encoder generates a 10-bit code group

from the 8-bit data and 1-bit control identifier. In double-width mode, there are two 8B/10B encoders that are cascaded together to generate two 10-bit code groups from two 8-bit data and their respective control identifiers. The 8B/10B encoder conforms to the IEEE 802.3 1998 edition standards.

Figure 2–25. 8B/10B Encoder



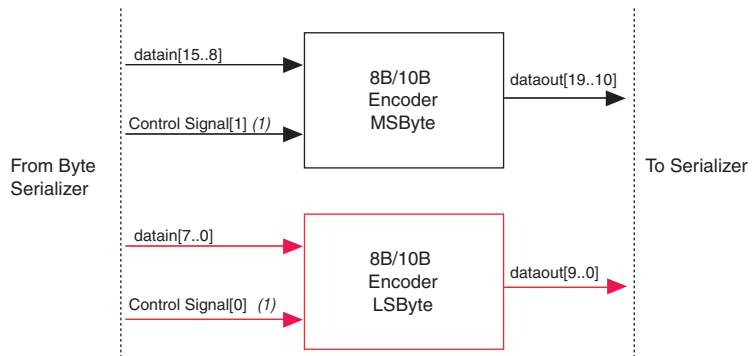
Note to Figure 2–25:

(1) The control signal is tx_ctrlenable.

Single-Width Mode

The 8B/10B encoder data path active in single-width mode is highlighted in Figure 2–26.

Figure 2–26. 8B/10B Encoder, Single-Width Mode



Note to Figure 2–26:

(1) The control signal is tx_ctrlenable.

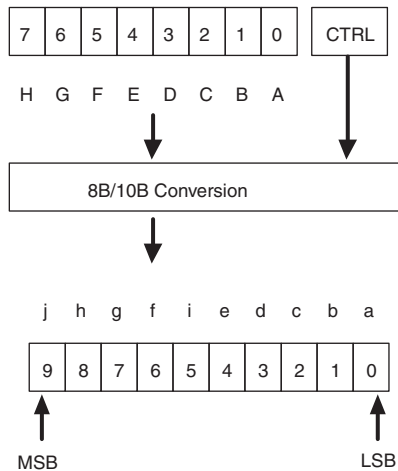
10-Bit Encoding

In single-width mode, the 8B/10B encoder translates the 8-bit data or 8-bit control character (as qualified by the control identifier) to a 10-bit code group with proper disparity. Figure 2–27 shows the conversion format. The LSB is transmitted first.



For additional information regarding the 8B/10B code itself, refer to the *Specifications & Additional Information* chapter in volume 2 of the *Stratix II GX Handbook*.

Figure 2–27. 8B/10B Conversion Format



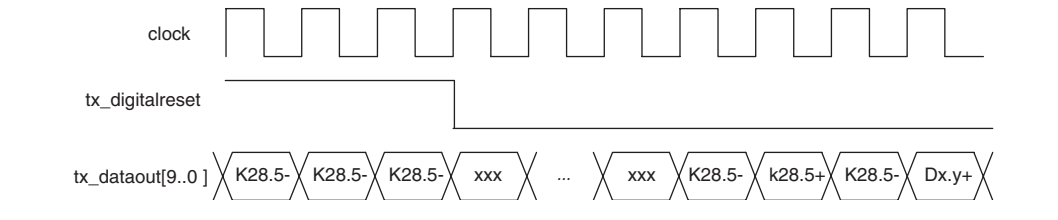
Reset Condition

The `tx_digitalreset` signal resets the 8B/10B encoder. During reset, the running disparity registers and the data registers are cleared. The 8B/10B encoder outputs a K28.5 pattern from the RD- column continuously until `tx_digitalreset` goes low. The input data and `tx_ctrlnable` signals are ignored during the reset state. Once out of reset, the 8B/10B encoder starts with a negative disparity (RD-) and transmits three K28.5 code groups for synchronizing before it starts encoding and transmitting the data on the `tx_datain` port.

While the `tx_digitalreset` signal is asserted, the 8B/10B decoder receives errors in the form of an invalid code error, synchronization error, control detect, and/or disparity error.

Figure 2–28 shows the 8B/10B encoder’s reset behavior. When in reset (`tx_digitalreset` is high), a K28.5- (K28.5 10-bit code group from the RD- column) is sent continuously until `tx_digitalreset` is low. The transmitter channel pipelining causes some “don’t cares (10’hxxx)” until the first of three K28.5 is sent. User data follows the third K28.5.

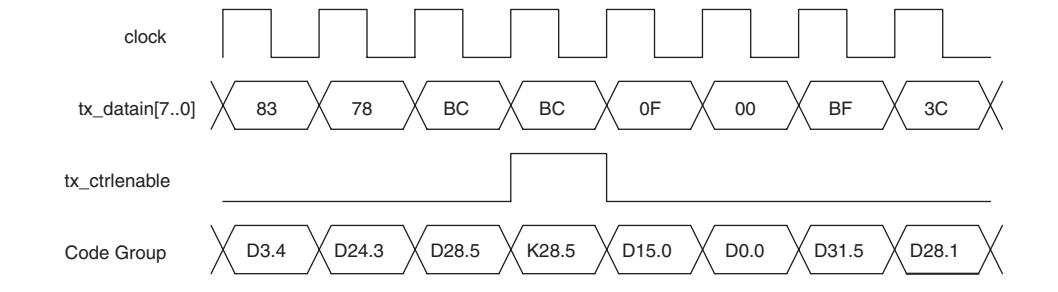
Figure 2–28. Transmitter Output During Reset Conditions



Control Code Encoding

The `tx_ctrlnable` port identifies if the 8-bit data at the `tx_datain` port is to be encoded as a control word (Kx.y). If this port is not used, there is no way to send control words out. When `tx_ctrlnable` is low, the byte at `tx_datain` port of the transceiver is encoded as data (Dx.y). When `tx_ctrlnable` is high, the data at the `tx_datain` port is encoded as a Kx.y code group. The waveform in Figure 2–29 shows that the 2nd 0xBC is encoded as a control word (K28.5). The rest of the `tx_datain` bytes are encoded as data (Dx.y).

Figure 2–29. Control Word Identification Waveform



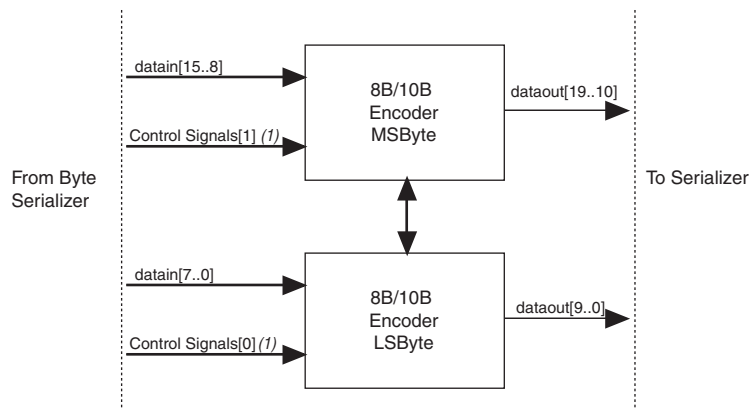
The 8B/10B encoder does not check to see if the code word entered is one of the 12 valid codes. If you enter an invalid control code, the resultant 10-bit code group may be encoded as an invalid code (does not map to a valid Dx.y or Kx.y code), or unintended valid Dx.y code, depending on the value entered.

It is possible for an 8B/10B decoder to decode an invalid control word encoded into a valid Dx.y code without asserting any code error flags. For example, depending on the current running disparity, the invalid code K24.1 ($tx_datain = 8'h38 + tx_ctrl = 1'b1$) can be encoded to $10'b0110001100$ (0x18C), which is equivalent to a D24.6+ (8'hD8 from the RD+ column). Altera recommends that you do not send invalid control words.

Double-Width Mode

In double-width mode, the 8B/10B encoder operates in a cascaded mode. The least significant byte is transmitted prior to the most significant byte. Figure 2–30 shows the active 8B/10B encoder blocks in double-width mode.

Figure 2–30. Active 8B/10B Encoder Blocks in Double-Width Mode



Note to Figure 2–30:

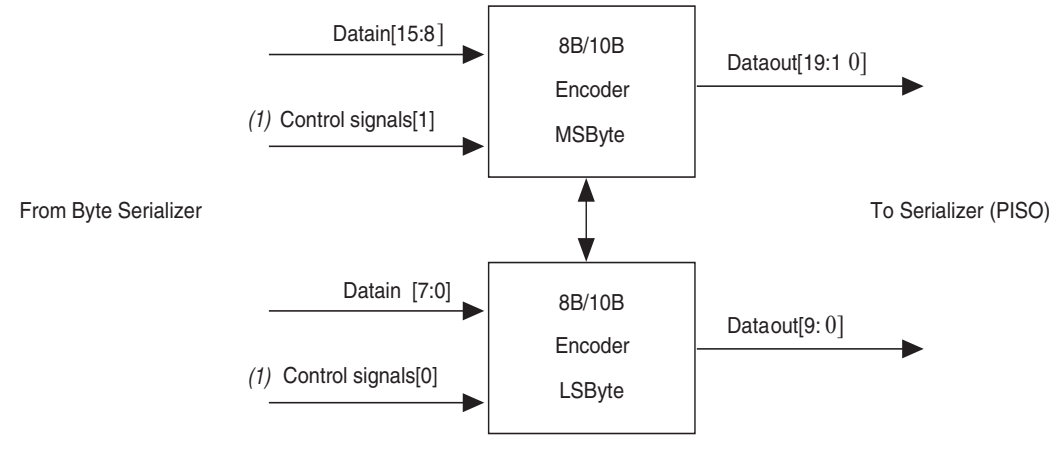
(1) The control signal is `tx_ctrlenable`.

20-Bit Encoding

In double-width mode, the cascaded 8B/10B encoders generate two 10-bit code groups from two 8-bit data and their respective control identifiers. The 8B/10B encoder forwards the current running disparity value from the LSByte encoder to the MSByte encoder to calculate the disparity of the symbol going into the MSByte encoder. The MSByte encoder's ending running disparity is then fed back to the LSByte encoder on the next clock cycle.

Refer to [Figure 2–31](#) for the conversion format. The LSB is transmitted first. [Figure 2–31](#) shows the 20-bit encoding.

Figure 2–31. 8B/10B Conversion Format

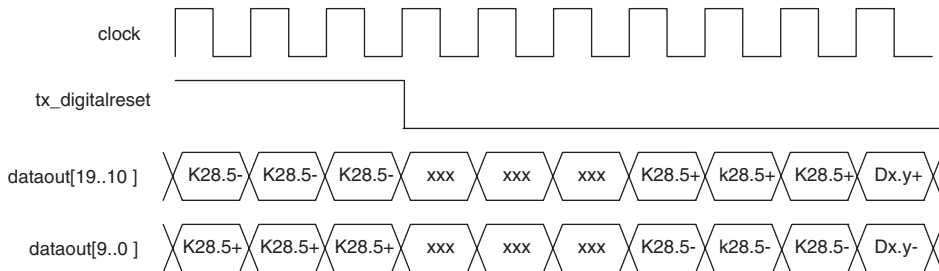


Reset Condition

The `tx_digitalreset` signal resets the 8B/10B encoder. During reset, the running disparity registers and the data registers are cleared. Also, the 8B/10B encoder outputs a K28.5 pattern with proper disparity continuously until `tx_digitalreset` goes low. The `tx_datain` and `tx_ctrlenable` ports are ignored during the reset state. Once out of reset, the 8B/10B encoder starts the LSByte with a negative disparity (RD-) bias and the MSByte with a positive disparity (RD+) and transmits six K28.5 code groups (three on the LSByte and three on the MSByte encoder) for synchronizing before it starts encoding and transmitting the data on `tx_datain`.

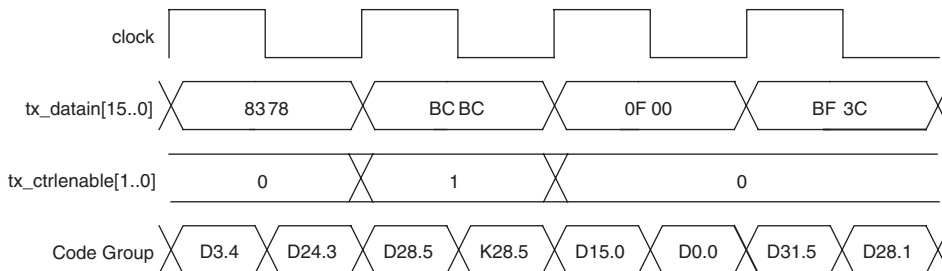
If the reset signal for the 8B/10B encoder is asserted, the 8B/10B decoder receiving the data may receive an invalid code error, synchronization error, control detect, and/or disparity error while `tx_digitalreset` is high.

[Figure 2–32](#) shows the reset behavior of the 8B/10B encoder. When in reset (`tx_digitalreset` is high) a K28.5- code group is sent continuously until `tx_digitalreset` is low. Transmitter channel pipelining causes some “don’t cares” (0’hxxx) until the first K28.5 is sent ([Figure 2–32](#) shows six don’t cares, but the number can vary). Both LSByte and MSByte transmit three K28.5 code groups each before the data at the `tx_datain` port is encoded and sent out.

Figure 2–32. Transmitter Output During Reset Conditions

Control Code Encoding

The `tx_ctrlenable` port identifies which 8-bit data is encoded as a control word. If this port is not used, control words cannot be sent. In double-width mode, the `tx_ctrlenable` port has two bits. The lower bit is associated with the LSB byte and the upper bit is associated with the MSByte. When `tx_ctrlenable` is low, the byte at the `tx_datain` port of the transceiver is encoded as data (Dx.y), otherwise, it is encoded as a control code (Kx.y). [Figure 2–33](#) shows that the lower byte of the second byte of `tx_datain` (8'hBC) is encoded as a control code as identified by a high on the lower `tx_ctrlenable` bit's second clock cycle.

Figure 2–33. Control Word Identification Waveform

The 8B/10B encoder does not check to see if the code word entered is one of the 12 valid codes. If an invalid control code is entered, the resultant 10-bit code may be encoded as an invalid code (does not map to a valid Dx.y or Kx.y code), or unintended valid Dx.y code, depending on the value entered.

It is possible for an 8B/10B decoder to decode an invalid control word encoded into a valid Dx.y code without asserting any code error flags. For example, depending on the current running disparity, the invalid code

K24.1 ($tx_datain = 8'h38 + tx_ctrl = 1'b1$) can be encoded to 10'b0110001100 (0x18C), which is equivalent to a D24.6+ (8'hD8 from the RD+ column). Altera recommends that you do not send invalid control words.

Transmitter Force Disparity

Upon power on or reset, the 8B/10B encoder has a negative disparity and will choose the 10-bit code from the RD- column. The Transmitter Force Disparity feature allows altering the running disparity via the `tx_forcedisp` and `tx_dispv` ports.

Two optional ports namely `tx_forcedisp` and `tx_dispv` ports are available in 8B/10B enabled Basic single-width and Basic double-width modes. In double-width mode, both `tx_forcedisp` and `tx_dispv` signals have two bits. The low bit is associated to the LSByte and the high bit is associated to the MSByte. A high value on the `tx_forcedisp` bit will change the disparity value of the data to the value indicated by the associated `tx_dispv` bit. If the `tx_forcedisp` bit is low, then `tx_dispv` is ignored and the current running disparity is not altered. Forcing disparity can either maintain the current running disparity calculations if the forced disparity value (on the `tx_dispv` bit) happens to match the current running disparity, or flip the current running disparity calculations if it does not. If the forced disparity flips the current running disparity, the down stream 8B/10B decoder may detect a disparity error which should be tolerated by the down stream device.

Figure 2–34 shows the current running disparity being altered in Basic single-width mode by forcing a positive disparity on 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 ending running disparity (RD+) K28.5 and a negative ending running disparity (RD-) K28.5 as governed by the 8B/10B encoder 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. Since the `tx_forcedisp` is high at time $n+4$, and `tx_dispv` is also high, the K28.5 at time $n+4$ is encoded as a positive disparity code group. As the `tx_forcedisp` is low at $n+5$ the K28.5 will take the current running disparity of $n+4$ and encode the K28.5 in time $n+5$ with a negative disparity. If the `tx_forcedisp` were driven high at time $n+5$, that K28.5 would also be encoded with positive disparity.

Figure 2–34. Transmitter Force Disparity Feature in Single-Width Mode

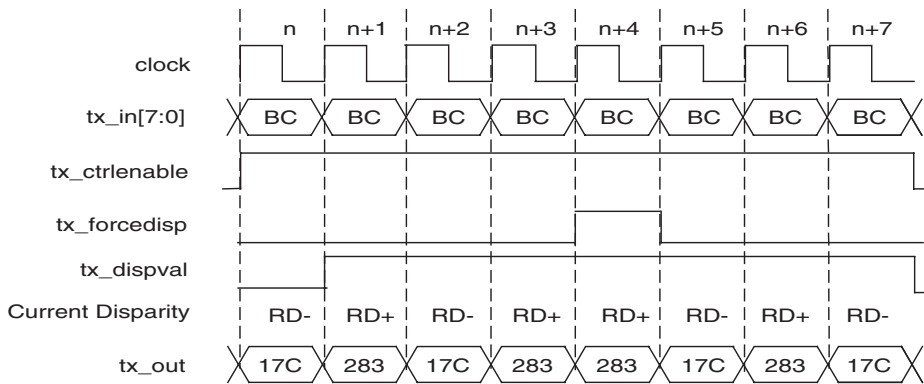
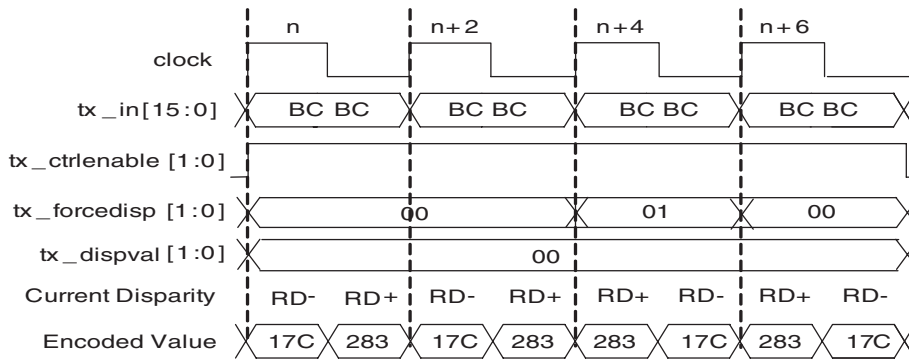


Figure 2–35 shows the current running disparity being altered in Basic double-width mode by forcing a positive disparity on a negative disparity K28.5. In this example, a series of K28.5 are continuously being sent. The stream alternates between a positive ending running disparity (RD+) K28.5 and a negative ending running disparity (RD-) K28.5 as governed by the 8B/10B encoder to maintain a neutral overall disparity. The current running disparity at the end of time n+2 indicates that the K28.5 at the low byte position in time n+4 should be encoded with a positive disparity. Since the `tx_forcedisp` is high at time n+4, the signal level of `tx_dispsval` is used to convert the lower byte K28.5 to be encoded as a negative disparity code word. As the upper bit of `tx_forcedisp` is low at n+4 the high byte K28.5 will take the current running disparity dictated by the low byte and encode the upper byte K28.5 with a positive disparity. If the upper bit of `tx_forcedisp` were driven high in time n+4, the upper byte K28.5 in time n+4 will be encoded with negative disparity.

Figure 2–35. Transmitter Force Disparity Feature in Double-Width Mode



Transmitter Polarity Inversion

The positive and negative signals of a serial differential link might accidentally be swapped during board layout. Solutions like a board re-spin or major updates to the PLD logic can prove expensive. The transmitter polarity inversion feature is provided to correct this situation.

An optional `tx_invpolarity` port is available in all single-width and double-width modes except (OIF) CEI PHY to dynamically enable the transmitter polarity inversion feature. In single-width modes, a high value on the `tx_invpolarity` port inverts the polarity of every bit of the 8-bit or 10-bit input data word to the serializer in the transmitter data path. In double-width modes, a high value on the `tx_invpolarity` port inverts the polarity of every bit of the 16-bit or 20-bit input data word to the serializer in the transmitter data path. Since inverting the polarity of each bit has the same effect as swapping the positive and negative signals of the differential link, correct data is seen by the receiver. The `tx_invpolarity` is a dynamic signal and may cause initial disparity errors at the receiver of an 8B/10B encoded link. The downstream system must be able to tolerate these disparity errors.

Figure 2–36 illustrates the transmitter polarity inversion feature in a single-width 10-bit wide data path configuration.

Figure 2–36. Transmitter Polarity Inversion in Single-Width Mode

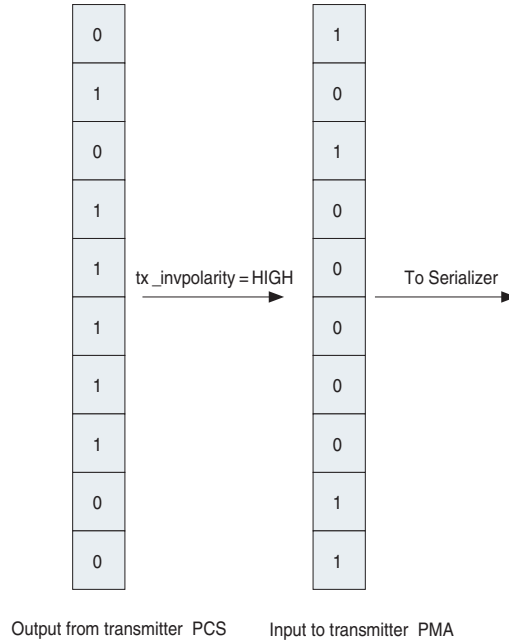
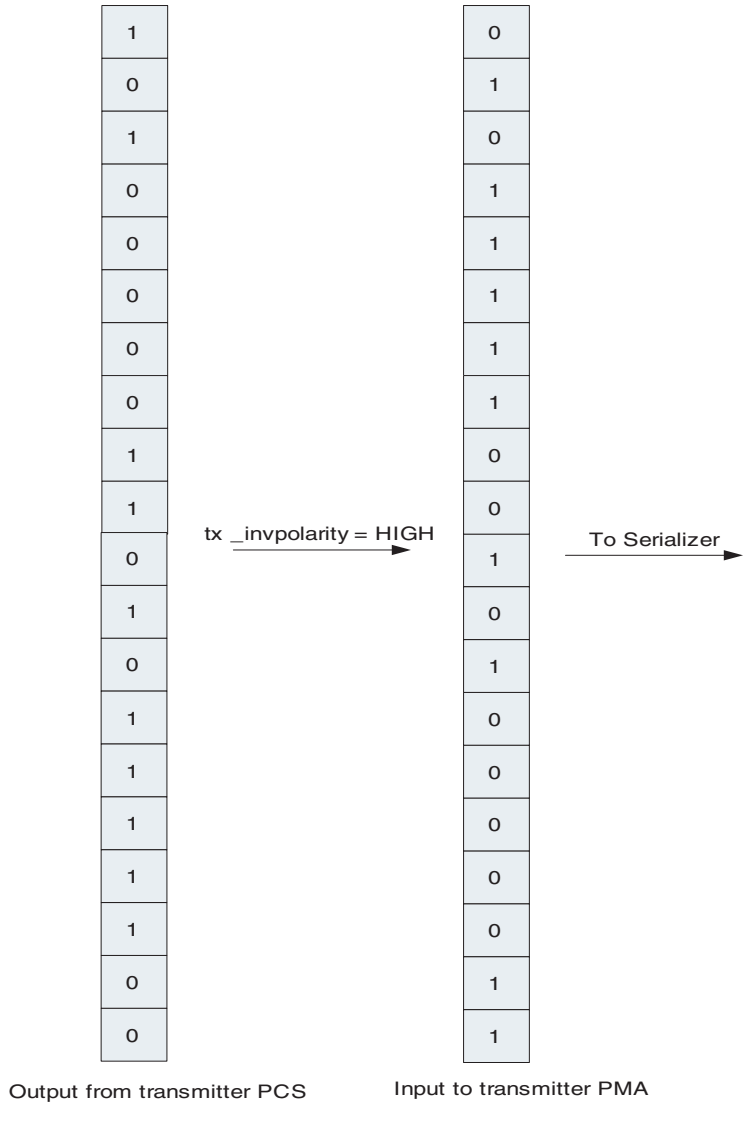


Figure 2–37 illustrates the transmitter polarity inversion feature in double-width 20-bit wide data path configuration.

Figure 2–37. Transmitter Polarity Inversion in Double-Width Mode



Transmitter Bit Reversal

By default, the Stratix II GX transmitted bit order is LSBit to MSBit. In single-width mode, the least significant bit of the 8/10-bit data word is transmitted first and the most significant bit is transmitted last. In double-width mode, the least significant bit of the 16/20-bit data word is transmitted first and the most significant bit is transmitted last. The Transmitter Bit Reversal feature allows reversing the transmitted bit order as MSBit to LSBit.

If the Transmitter Bit Reversal feature is enabled in Basic single-width mode, the 8-bit D[7:0] or 10-bit D[9:0] data at the input of the serializer gets rewired to D[0:7] or D[0:9], respectively. If the Transmitter Bit Reversal feature is enabled in Basic double-width mode, the 16-bit D[15:0] or 20-bit D[19:0] data at the input of the serializer gets rewired to D[0:15] or D[0:19], respectively. Flipping the parallel data using this feature and transmitting LSBit to MSBit effectively provides MSBit to LSBit transmission.

Figure 2–38 illustrates the transmitter bit reversal feature in Basic single-width 10-bit wide data path configuration.

Figure 2–38. Transmitter Bit Reversal in Single-Width Mode

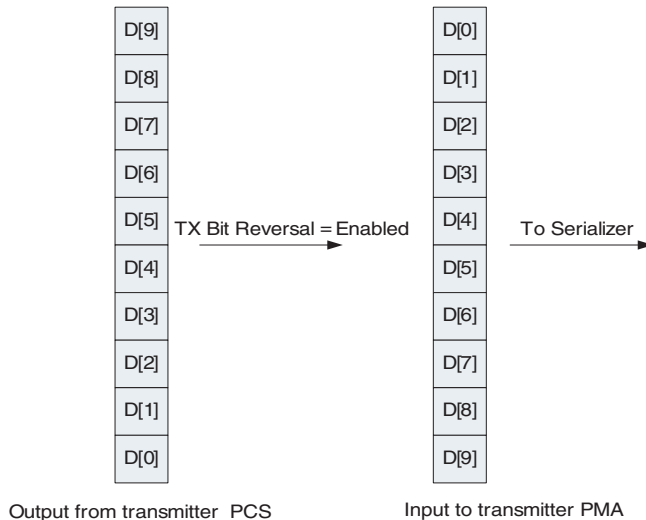
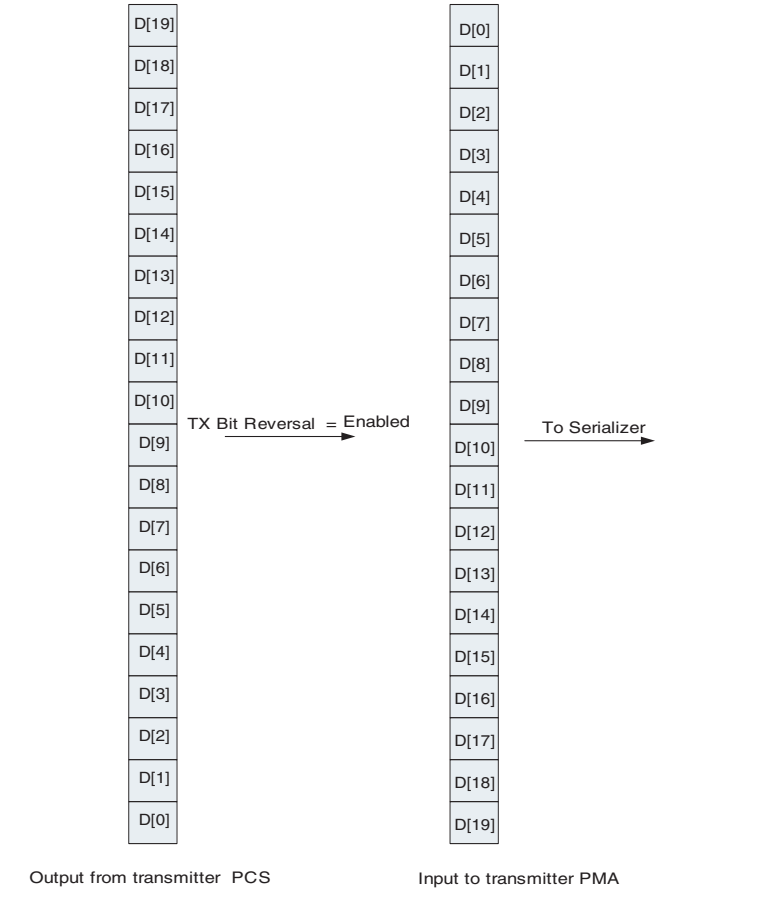


Figure 2–39 illustrates the transmitter bit reversal feature in Basic double-width 20-bit wide data path configuration.

Figure 2–39. Transmitter Bit Reversal in Double-Width Mode



Serializer

The serializer block converts parallel data to serial data at the transmitter output buffer. The serializer block supports 8-bit (Figure 2–40), 10-bit, 16-bit, and 20-bit words. The 8-bit and 10-bit operations are for use in the single-width mode and support the data rate range from 600 Mbps to 3.125 Gbps. The 16-bit and 20-bit operations are for the double-width mode and support the data rate range from 1 Gbps to 6.375 Gbps.

The serializer block drives the serial data to the output buffer as shown in the figure below. The serializer block drives the serial bit-stream at a data rate range of 600 Mbps to 6.375 Gbps. The serializer block natively transmits the LSB of the word first.

Figure 2–40. Serializer Block In 8-bit Mode

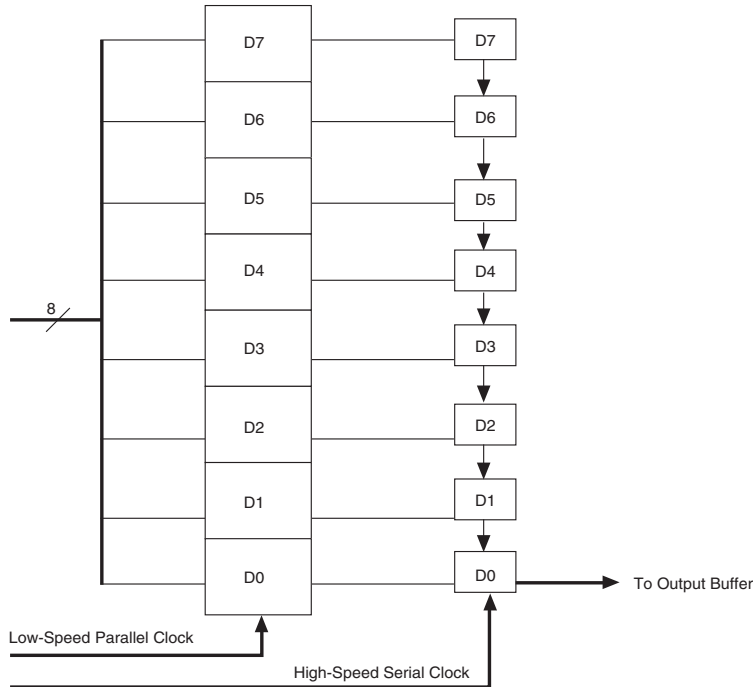
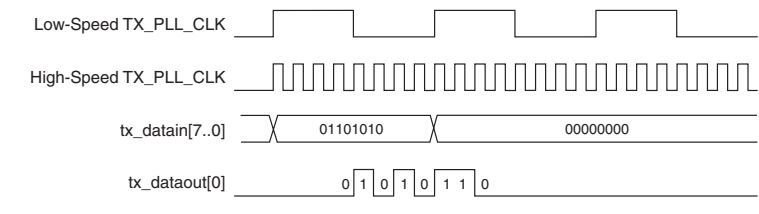


Figure 2–41 shows the serial bit order of the serializer block output. In this example a constant 8'h6A (01101010) value is serialized, and the serial data is transmitted from LSB to MSB.

Figure 2–41. Serializer Bit Order



In the individual channel mode, the serializer block supplies the PLD and transceiver parallel clock. The serializer takes the $\div 8$ or $\div 10$ parallel clock from the clock divider block and distributes it to the transmitter's PCS logic in the associated transmitter channel. In single-width mode, the clock is unaltered. In double-width mode, the serializer block creates a $\div 16$ or $\div 20$ clock from the clock provided by the clock divider block, depending on the serialization factor.

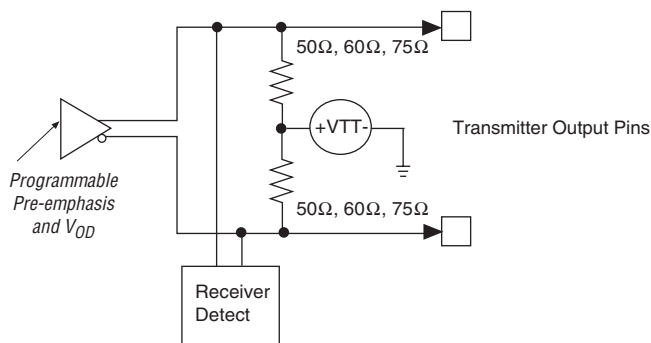


In Quartus II software version 7.1 and later, basic single width allows 8-bit serialization (disable 8B/10B).

Transmitter Buffer

The Stratix II GX transmitter buffers support 1.2-V and 1.5-V pseudo current mode logic (PCML) up to 6.375 Gbps and can drive 40 inches of FR4 trace across two connectors. The transmitter buffer (refer to [Figure 2-42](#)) has additional circuitry to improve signal integrity—programmable output voltage, programmable three-tap pre-emphasis circuit, and internal termination circuitry—and the capability to detect the presence of a downstream receiver.

Figure 2-42. Transmitter Buffer



Programmable Voltage Output Differential (V_{OD})

Stratix II GX device's allow you to customize the differential output voltage (V_{OD}) to handle different trace lengths, various backplanes, and receiver requirements (refer to [Figure 2-43](#)). You select the V_{OD} from a range between 200 and 1,400 mV, as shown in [Table 2-8](#).

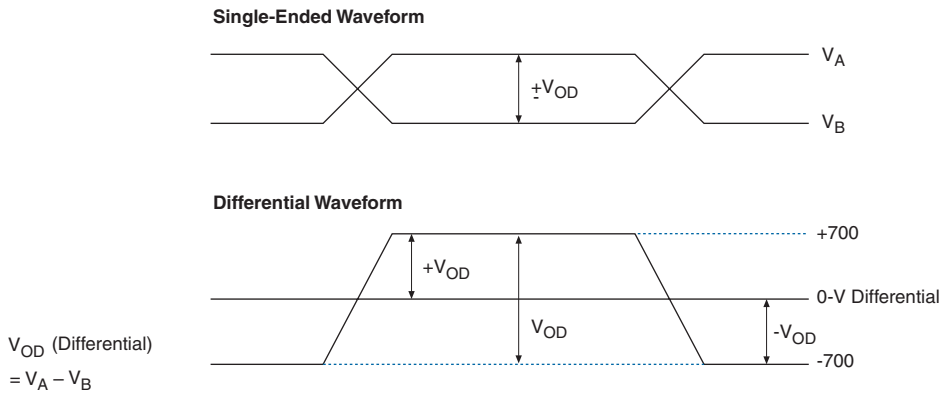
Figure 2–43. V_{OD} (Differential) Signal Level

Table 2–8 shows the differential output voltage V_{OD} setting per supply voltage for each of the on-chip transmitter programmable termination values.

Table 2–8. Programmable V_{OD}					
V_{OD} Differential Peak to Peak					
1.2-V VCC			1.5-V VCC		
100-Ω (mV)	120-Ω (mV)	150-Ω (mV)	100-Ω (mV)	120-Ω (mV)	150-Ω (mV)
—	192	240	200	240	300
320	384	480	400	480	600
480	576	720	600	720	900
640	768	960	800	960	1,200
800	960	—	1,000	1,200	—
960	—	—	1,200	—	—
—	—	—	1,400	—	—

You set the V_{OD} values in the MegaWizard.

The transmitter buffer is powered by either a 1.2-V or a 1.5-V power supply. You choose the transmitter buffer power (V_{CCH}) of 1.2 V or 1.5 V through the ALT2GXB MegaWizard Plug-In Manager (the **What is the transmit buffer power (V_{CCH})?** option). The transmitter buffer power supply in Stratix II GX devices is transceiver based. The 1.2-V power supply supports the 1.2-V PCML standard.

You specify the V_{OD} settings either through the MegaWizard or dynamically using the dynamic reconfiguration controller. Refer to [“Introduction” on page 2-1](#) for more information

Programmable Pre-Emphasis

The programmable pre-emphasis module in each transmit buffer boosts the high frequencies in the transmit data signal, which may be attenuated in the transmission media. Using pre-emphasis can maximize the data eye opening at the far-end receiver.

The transmission line’s transfer function can be represented in the frequency domain as a low-pass filter. Any frequency components below the -3dB frequency pass through with minimal losses. Frequency components greater than the -3dB frequency are attenuated. This variation in frequency response yields data-dependent jitter and other ISI effects. By applying pre-emphasis, the high frequency components are boosted, that is, pre-emphasized. Pre-emphasis equalizes the frequency response at the receiver so the difference between the low-frequency and high-frequency components are reduced, which minimizes the ISI effects from the transmission medium.

The pre-emphasis requirements increase as data rates through legacy backplanes increase. The Stratix II GX transmitter buffer employs a pre-emphasis circuit with up to 477% of pre-emphasis to correct for losses in the transmission medium.

You set pre-emphasis settings through a slider menu in the ALT2GXB MegaWizard Plug-In Manager. Specify the pre-emphasis settings (pre-emphasis control pre-tap, pre-emphasis 1st post tap, and pre-emphasis 2nd post tap) through the MegaWizard or dynamically using the dynamic reconfiguration controller. To enable the dynamic reconfiguration controller, you must first enable the option for dynamic reconfiguration in the ALT2GXB MegaWizard. After you enable that option, you must configure the settings in the ALT2GXB_RECONFIG MegaWizard Plug-In Manager (Stratix II GX device family) in the Quartus II software. Refer to [“Introduction” on page 2-1](#) for more information.

Programmable Transmitter Termination

The Stratix II GX transmitter buffer includes programmable on-chip differential termination of $100\ \Omega$, $120\ \Omega$, or $150\ \Omega$. The resistance is adjusted by the on-chip calibration circuit in the calibration block (refer to [“Calibration Blocks” on page 2-229](#) for more information), which compensates for temperature, voltage, and process changes. The Stratix II GX transmitter buffers in the transceiver are current mode

drivers, so the resultant V_{OD} is a function of the transmitter termination value. Refer to “[Programmable Voltage Output Differential \(VOD\)](#)” on [page 2–48](#) for more information regarding resultant V_{OD} values. You can disable the on-chip termination to use external termination. If you select external termination, the transmitter common mode is also tri-stated.

You select transmitter termination in the Quartus II software (in the Assignments menu choose Assignment Organizer, then Options for Individual Nodes Only, and then Stratix II GX Termination Value). By default, the value is 100 Ω . If you plan to use 100- Ω termination, the Quartus II software automatically sets it during the compilation.

You set the transmitter termination setting through a pull-down menu in the ALT2GX MegaWizard

Programmable Common Mode

You can set the common mode in Stratix II GX devices to 600 mV or 700 mV. Use the 600-mV setting with the 1.2-V PCML standard. Use the 700-mV setting for the 1.5-V PCML standard. [Table 2–9](#) shows the available common mode settings for 1.2-V/1.5-V V_{CCH} .

Common Mode Settings	1.2-V PCML Standard	1.5-V PCML Standard	Data Rates (Mbps)
600 mV	✓	✓	600 to 3125
600 mV	—	✓	3126 to 6375
700 mV	—	✓	600 to 3125

Note to [Table 2–9](#):

- (1) The PIPE protocol only allows 1.2-V PCML with 600 mV common mode. It does not allow a 1.5-V PCML and 700 mV combination.

PCI Express Receiver Detect

The Stratix II GX transmitter buffer has a built-in receiver detection circuit for use in the PIPE mode. This circuit detects if there is a receiver downstream by sending out a pulse on the common mode of the transmitter and monitoring the reflection. This mode requires the transmitter buffer to be tri-stated (in Electrical Idle mode) and the use of on-chip termination and a 125 MHz `fixedclk` signal.

This feature is only available in the PIPE mode and you enable it by setting the `tx_forceelectidle` and `tx_detectrxloopback` ports to 1'b1. You must set the `powerdn` port to 2'b10 to place the transmitter in the PCI-Express P1 power-down state. The results of the receiver detect is encoded on the `pipestatus` port.

PCI Express Electrical Idle

The Stratix II GX transmitter buffer supports PCI Express Electrical Idle (or individual transmitter tri-state). This feature is only active in the PIPE mode. The `tx_forceelectidle` port puts the transmitter buffer in Electrical Idle mode. This port is available in all PCI Express power-down modes and has a specific use in each mode. [Table 2-10](#) shows the usage in each power mode.

Power Mode	Usage
P0	<code>tx_forceelectidle</code> must be asserted. If this signal is deasserted, it indicates that there is valid data.
P1	<code>tx_forceelectidle</code> must be asserted.
P2	When deasserted, the beacon signal must be transmitted. Refer to "PCI Express (PIPE) Mode" on page 2-150.

Receiver Modules

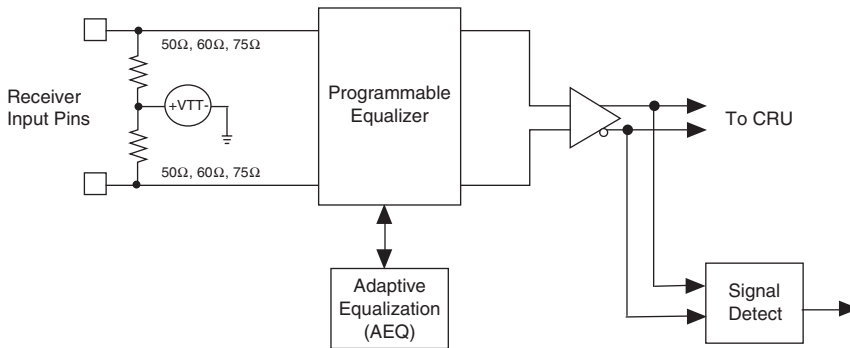
This section describes the Stratix II GX transceiver's receiver path. This section describes the following modules:

- Receiver buffer
- Receiver PLL
- Clock recovery unit
- Deserializer
- Word aligner
- Channel aligner (deskew)
- Rate matcher
- 8B/10B decoder
- Byte deserializer
- Byte Ordering
- Receiver phase compensation FIFO buffer

Receiver Buffer

The Stratix II GX receiver buffers support 1.2-V, 1.5-V, 3.3-V PCML (pseudo-current mode logic), differential LVPECL and LVDS I/O standards. The receiver buffers support data rates from 600 Mbps to 6.375 Gbps and are capable of compensating up to 40 inches of FR4 trace across two connectors. The receiver buffer (Figure 2–44) has additional circuitry to improve signal integrity, including a programmable equalization circuit and internal termination circuitry. Through a signal detect circuit, the receiver buffers can also detect if a signal of predefined amplitude exists at the receiver.

Figure 2–44. Receiver Buffer



Programmable Receiver Termination

The Stratix II GX receiver buffer has an optional programmable on-chip differential termination of 100 Ω, 120 Ω or 150 Ω. You can set the receiver termination resistance setting using one of the two ways.

- Set the receiver termination resistance option in the MegaWizard if on-chip termination is used. The settings allowed are 100 Ω, 120 Ω and 150 Ω. If the design requires external receive termination, check the **Use external Receiver termination** option.
- You make the differential termination assignment per pin in the Quartus II software (in the Assignments menu, choose Assignment Organizer, then Options for Individual Nodes Only, and then Stratix II GX GXB Termination Value).



Verify and set the receiver termination settings before compilation.

Signal Threshold Detection Circuit

The signal detect feature supported only in PIPE mode. The signal detect/loss threshold detector senses if the specified voltage level exists at the receiver buffer. This detector has a hysteresis response, that filters out any high-frequency ringing caused by inter-symbol interference or high-frequency losses in the transmission medium. The `rx_signaldetect` signal indicates if a signal conforms to the signal detection settings. A high level indicates that the signal conforms to the settings, a low level indicates that the signal does not conform to the settings.

The signal detect levels are to be determined by characterization. The signal detect levels may vary because of changing data patterns.

The signal/detect loss threshold detector also switches the receiver PLL/CRU from lock to reference mode to lock to data mode. The lock to reference and lock to data modes dictate whether the VCO of the clock recovery unit (CRU) is trained by the reference clock or by the data stream. Refer to [“Lock-to-Reference and Lock-to-Data Modes” on page 2–64](#) for more information regarding this process.

You can bypass the signal/detect loss threshold detection circuit by choosing the Forced Signal Detect option in the MegaWizard. This is useful in lossy environments where the voltage thresholds might not meet the lowest voltage threshold setting. Forcing this signal high enables the receiver PLL to switch from VCO training based on the reference clock to the incoming data without detecting a valid voltage threshold.

Receiver Common Mode

Stratix II GX transceivers support the receiver buffer common mode voltage of 0.85 V and 1.2 V.

For AC-coupled links, Altera recommends selecting 0.85 V as the receiver buffer common mode voltage. For DC-coupled links, refer to [“DC Coupling” on page 2–55](#) for recommendations on selecting the receiver common mode voltage.

Programmable Equalization

The Stratix II GX device offers an equalization circuit in each gigabit transceiver block receiver channel to increase noise margins and help reduce the effects of high-frequency losses. The programmable equalizer compensates for the high-frequency losses that distort the signal and reduces the noise margin of the transmission medium by equalizing the frequency response. There are 16 equalizer control settings allowed for a Stratix II GX device (including a setting with no equalization). In addition

to equalization, Stratix II GX devices offer an equalizer DC gain option. There are three legal settings for DC gain. You specify the equalizer settings (Equalization Settings and DC Gain) either through the MegaWizard or dynamically using the dynamic reconfiguration controller. To enable the dynamic reconfiguration controller, you first enable the option for dynamic configuration in the ALT2GXB MegaWizard Plug-In Manager. After you enable that option, you must configure the settings in the ALT2GXB_RECONFIG MegaWizard Plug-In Manager (Stratix II GX device family) in the Quartus II software. Refer to [“Introduction” on page 2-1](#) for more information.

The transmission line's transfer function can be represented in the frequency domain as a low-pass filter. Any frequency components below the -3 -dB frequency pass through with minimal losses. Frequency components that are greater than the -3 -dB frequency are attenuated. This variation in frequency response yields data-dependant jitter and other ISI effects. By applying equalization, the low frequency components are attenuated. This equalizes the frequency response such that the delta between the low frequency and high frequency components are reduced, which in return minimizes the ISI effects from the transmission medium.



Stratix II GX receiver also offers the Adaptive Equalization (AEQ) feature. If you enable this feature, the equalizer circuit automatically selects and adjusts appropriate equalization levels, depending on the changing link characteristics. You can enable this feature through the dynamic reconfiguration controller. For more information, refer to the [Stratix II GX Dynamic Reconfiguration](#) chapter in volume 2 of the *Stratix II GX Device Handbook*.

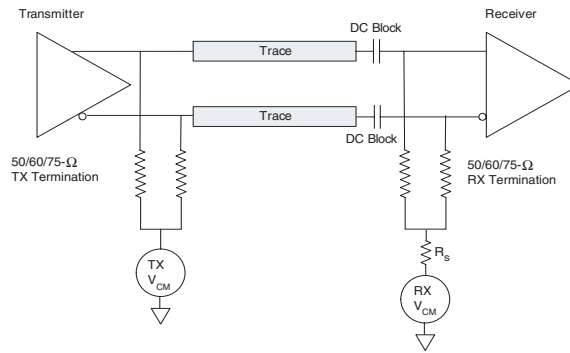
DC Coupling

A high-speed serial link can either be AC-coupled or DC-coupled, depending on the serial protocol being implemented. While most of the serial protocols require links to support AC-coupling, protocols like Common Electrical I/O (CEI) optionally allow DC coupling.

In an AC coupled link, the DC blocking capacitor blocks the transmitter DC common mode voltage ($TX V_{CM}$). The Stratix II GX receiver buffer allows a common mode voltage ($RX V_{CM}$) setting of 0.85 V and 1.2 V. The on-chip receiver termination and bias circuitry automatically restores the selected common mode voltage. If you select external termination, you must ensure that the link is terminated properly to the selected common mode voltage.

Figure 2–45 shows an AC coupled link.

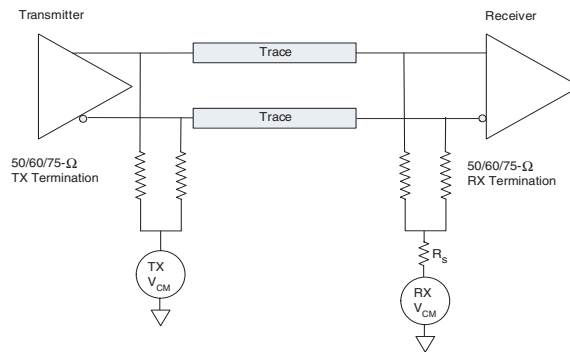
Figure 2–45. AC Coupled Link



In a DC coupled link, the transmitter DC common mode voltage is seen unblocked at the receiver buffer. The trace common mode voltage depends on the transmitter common mode voltage and the receiver common mode voltage. The external or on-chip receiver termination and bias circuitry must ensure compatibility between the transmitter and the receiver common mode voltage.

Figure 2–46 shows a DC coupled link.

Figure 2–46. DC Coupled Link



The following protocols mandate AC coupled links:

- PCI Express (PIPE)
- Gigabit Ethernet
- Serial RapidIO
- CPRI
- XAUI
- SDI

You may choose to DC couple the high-speed link for the following functional modes only:

- Basic Single and Double Width
- SONET/SDH
- CEI

The following sections discuss DC coupling requirements for a high-speed link with a Stratix II GX device used as the transmitter, receiver, or both. Specifically, the following link configurations are discussed:

- Stratix II GX Transmitter (PCML) to Stratix II GX Receiver (PCML)
- Stratix II GX Transmitter (PCML) to Stratix GX Receiver (PCML)
- Stratix GX Transmitter (PCML) to Stratix II GX Receiver (PCML)
- LVDS Transmitter to Stratix II GX Receiver (PCML)

Stratix II GX Transmitter (PCML) to Stratix II GX Receiver (PCML)

Figure 2-47 shows a typical Stratix II GX to Stratix II GX DC coupled link.

Figure 2-47. Stratix II GX to Stratix II GX DC Coupling

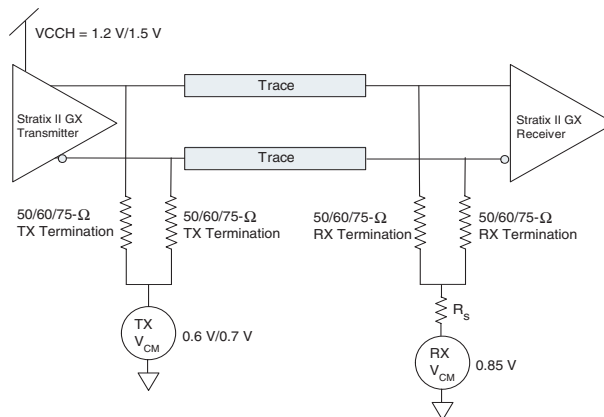


Table 2–11 shows the allowed transmitter and receiver settings in a Stratix II GX to Stratix II GX DC coupled link.

Transmitter (Stratix II GX) Settings				Receiver (Stratix II GX) Settings		
Data Rate	VCCH (1)	TX V_{CM} (1)	Differential Termination	Data Rate	RX V_{CM}	Differential Termination
600-6375 Mbps	1.2 V/1.5 V	0.6 V/0.7 V	100/120/150- Ω	600-6375 Mbps	0.85 V	100/120/150- Ω

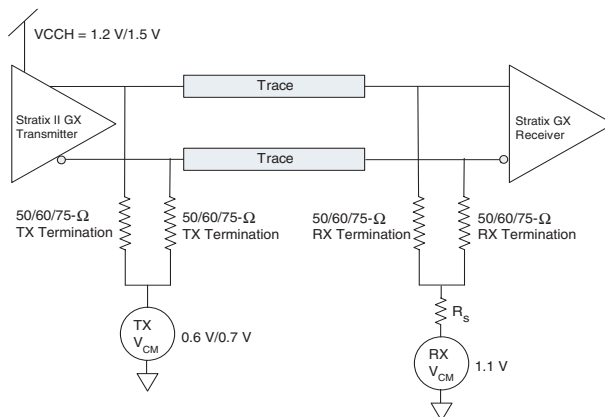
Note to Table 2–10:

- (1) VCCH = 1.2 V with TX V_{cm} = 0.6 V can support data rates from 600 Mbps to 3125 Mbps. VCCH = 1.5 V with TX V_{cm} = 0.7 V can support data rates from 600 Mbps to 3125 Mbps. VCCH = 1.5 V with TX V_{cm} = 0.6 V can support data rates from 600 Mbps to 6375 Mbps.

Stratix II GX Transmitter (PCML) to Stratix GX Receiver (PCML)

Figure 2–48 shows a typical Stratix II GX to Stratix GX DC coupled link.

Figure 2–48. Stratix II GX to Stratix GX DC Coupling Note (1)



Note to Figure 2–48:

- (1) When DC coupling to a Stratix GX receiver, you must select the **Enable Stratix GX to Stratix GX DC coupling** option for the Stratix GX receiver.

Table 2–12 shows the allowed transmitter and receiver settings in a Stratix II GX to Stratix GX DC coupled link.

Transmitter (Stratix II GX) Settings				Receiver (Stratix GX) Settings		
Data Rate	VCCH (1)	TX V_{CM} (1)	Differential Termination	Data Rate	RX V_{CM}	Differential Termination
600-3187.5 Mbps	1.5 V (1.5 V PCML)	0.6 V/0.7 V	100/120/150- Ω	600-3187.5 Mbps	1.1 V	100/120/150- Ω

Note to Table 2–12:

- (1) VCCH = 1.5 V with TX V_{cm} = 0.7 V can support data rates from 600 Mbps to 3125 Mbps. VCCH = 1.5 V with TX V_{cm} = 0.6 V can support data rates from 600 Mbps to 3187.5 Mbps.

Stratix GX Transmitter (PCML) to Stratix II GX Receiver (PCML)

Figure 2–49 shows a typical Stratix GX to Stratix II GX DC coupled link.

Figure 2–49. Stratix GX to Stratix II GX DC Coupling

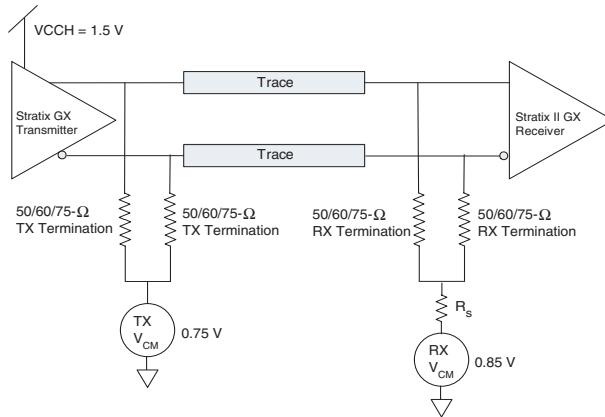


Table 2–13 shows the allowed transmitter and receiver settings in a Stratix GX to Stratix II GX DC coupled link.

Transmitter (Stratix GX) Settings				Receiver (Stratix II GX) Settings			
Data Rate	VCCH	TX V_{CM}	Differential Termination	Data Rate	I/O Standard	RX V_{CM}	Differential Termination
600-3187.5 Mbps	1.5 V	0.75 V	100/120/150- Ω	600-3187.5 Mbps	1.5 V PCML	0.85 V	100/120/150- Ω

LVDS Transmitter to Stratix II GX Receiver (PCML)

Figure 2–50 shows a typical LVDS transmitter to Stratix II GX receiver DC coupled link.

Figure 2–50. LVDS to Stratix II GX DC Coupling

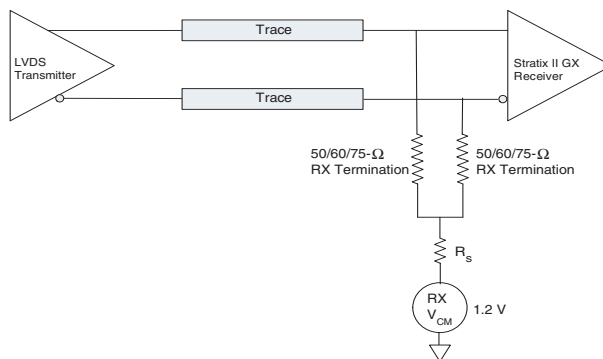


Table 2–14 shows the allowed transmitter and receiver settings in a LVDS to Stratix II GX DC coupled link.

Receiver (Stratix II GX) Settings		
RX V_{CM}	Differential Termination	R_s
1.2 V	100- Ω	88- $\Omega \pm 10\%$

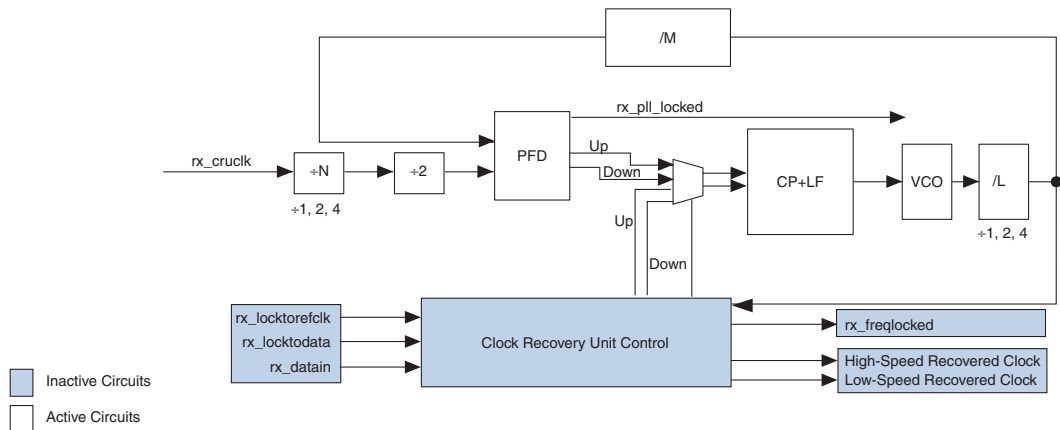
Note to Table 2–14:

- (1) When DC coupling an LVDS transmitter to the Stratix II GX receiver, use $RX V_{cm} = 1.2$ V and series resistance value $R_s = 88$ - Ω to verify compliance to the LVDS specification.

Receiver PLL

Each transceiver block contains four receiver PLLs, which receive the reference clock to train the VCO used by the CRU to match the phase and frequency of the reference clock. Figure 2-51 shows the block diagram for the lock-to-reference portion as the receiver PLL is active. Table 2-15 lists some of the PLL specifications. Table 2-16 lists the available $/M$ and $/L$ values within the receiver PLL.

Figure 2-51. Receiver PLL Block Diagram



Note to Figure 2-51:

- (1) Values of $/M$ and $/L$ counters are specified in Table 2-16. The Quartus II software selects these values automatically based on the data rate and the selected reference clock frequencies.



This section focuses on the receiver PLL in lock-to-reference mode only (the receiver is not active in lock-to-data mode). The lock-to-data mode is discussed in the section “Clock Recovery Unit” on page 2-63. For information on the operation between the lock-to-reference and lock-to-data modes, refer to “Lock-to-Reference and Lock-to-Data Modes” on page 2-64.

The receiver PLL has an optional lock indicator, `rx_pll_locked`, which indicates when the receiver PLL is phase and frequency locked to the reference clock. The `rx_pll_locked` is an active high signal. A high signal indicates that the PLL is phase and frequency-locked to a reference clock, a low signal indicates that the PLL is not locked to the reference clock. If the CRU is locked to the incoming data, the `rx_pll_locked` port may toggle (assert and deassert) because the phase and/or frequency differences between the recovered clock and the reference clock might be large enough to trigger a loss of lock. This is an expected behavior because the receiver PLL is inactive in the lock-to-data mode.

and the `rx_pll_locked` signal is ignored when the CRU is in lock-to-data mode. Table 2-15 lists the specifications for the clock recovery unit. Table 2-16 lists the available /M and /L values within the receiver PLL.

Table 2-15. Clock Recovery Unit Specifications

Parameter	Specifications
Input reference frequency range	50 MHz to 622.08 MHz
Data rate support	600 Mbps to 6.375 Gbps

Table 2-16. Available /M and /L Values Within the Receiver PLL

RX PLL Reference Clock Source	/M	/L
<code>rx_crucclk</code>	1, 4, 5, 8, 10, 16, 20, 25	1, 2, 4

Clock Synthesis

The maximum input frequency of the receiver PLL's phase frequency detector (PFD) is 325 MHz. To achieve a reference clock frequency above this limitation, the divide by 2 pre-divider on the dedicated local `REFCLK` path is automatically enabled by the Quartus II software. This divides the reference clock frequency by a factor of 2, and the /M PLL multiplier multiplies this pre-divided clock to yield the configured data rate. For example, in a situation with a data rate of 2,488 Mbps and a reference clock of 622 MHz, the reference clock must be assigned to the `REFCLK` port where the 622-MHz reference clock can be divided by 2, yielding a 311-MHz clock at the PFD. The VCO runs at half the data rate, so the selected multiplication factor should yield a 1244 MHz high-speed clock. The Quartus II software automatically selects a multiplication factor of 4 in this case to generate a 1244 MHz clock from the pre-divided 311 MHz clock.

If the /2 pre-divider is used, the reference clock must be fed by a dedicated reference clock input (`REFCLK`) pin. Otherwise the Quartus II compiler gives a fitter error.

The pre-divider and the multiplication factors are automatically set by the Quartus II software. The MegaWizard takes the data rate input and provides a list of the available reference clock frequencies that fall within the supported multiplication factors that you can select.

PPM Frequency Threshold Detector

The PPM frequency threshold detector senses whether the incoming reference clock to the clock recovery unit (CRU) and the PLL VCO of the CRU are within a prescribed PPM tolerance range. Valid parameters are 62.5, 100, 125, 200, 250, 300, 500, or 1,000 PPM. The default parameter, if no assignments are made, is 1,000 PPM. The output of the PPM frequency threshold detector is one of the variables that assert the `rx_freqlocked` signal. Refer to [“Automatic Lock Mode” on page 2-64](#) for more details regarding the `rx_freqlocked` signal.

Receiver Bandwidth Type

The Stratix II GX receiver PLL in the CRU offers a programmable bandwidth setting. The PLL bandwidth is the measure of its ability to track the input data and jitter. The bandwidth is determined by the -3dB frequency of the closed-loop gain of the PLL.

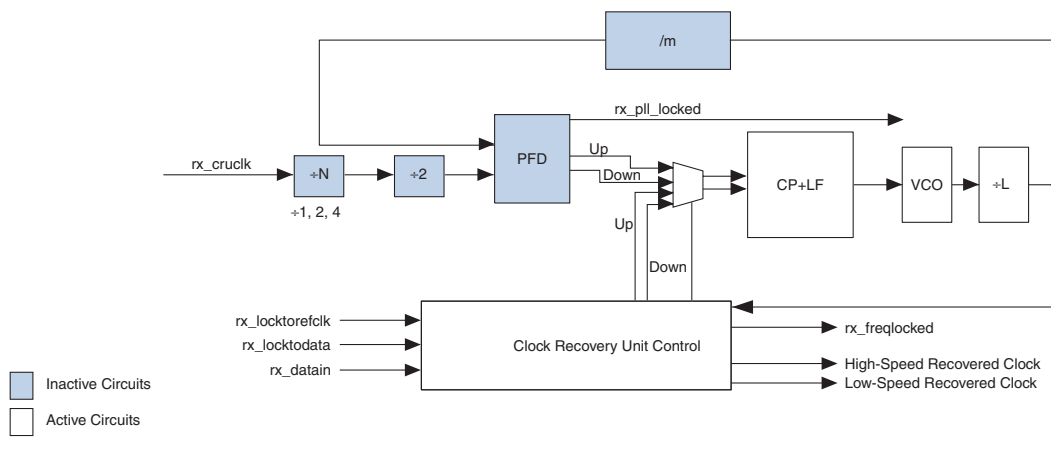
A higher bandwidth setting helps reject noise from the VCO and power supplies. A low bandwidth setting filters out more high-frequency data input jitter.

Valid receiver bandwidth settings are low, medium, or high. The -3dB frequencies for these settings vary because of the non-linear nature and data dependencies of the circuit. You can vary the bandwidth to adjust and customize the performance on specific systems.

Clock Recovery Unit

The CRU (refer to [Figure 2-52](#)) in each Stratix II GX transceiver channel recovers the clock from the serial data stream on `rx_datain`. You can set the CRU to automatically or manually lock to the data phase and frequency to match the bit transition to eliminate any clock-to-data skew or to keep the receiver PLL locked to the reference clock (lock-to-data or lock-to-reference mode). The CRU generates two clocks: a high-speed clock that feeds the deserializer and a slow-speed clock that feeds the rest of the receiver’s digital logic.

Figure 2–52. Clock Recovery Unit



Lock-to-Reference and Lock-to-Data Modes

The lock-to-reference and lock-to-data modes describe the receiver PLL and the CRU. The receiver PLL is active in the lock-to-reference mode and the CRU is active in the lock-to-data mode. The switch between the two modes can be done automatically or manually.

Automatic Lock Mode

The CRU, by default, initially locks to the CRU reference clock (lock-to-reference mode) until switching over to the incoming data (lock-to-data mode). The switch to lock-to-data mode is indicated by the assertion of the `rx_freqlocked` signal. The `rx_freqlocked` signal only indicates the current mode of CRU (lock-to-data or lock-to-reference). After switching into lock-to-data mode (assertion of the `rx_freqlocked` signal), the CRU unit needs time to acquire phase lock to the incoming data stream.

For automatic transition from the lock-to-reference mode to the lock-to-data mode, the following conditions must be met:

- The serial data at the receiver input buffer is within the prescribed voltage signal loss threshold.
- The CRU PLL is within the prescribed PPM frequency threshold setting (62.5, 100, 125, 200, 250, 300, 500, or 1,000 PPM) of the CRU reference clock.
- The reference clock and CRU PLL output are phase matched (phases are within approximately 0.08 UI).

When the receiver PLL and CRU is in lock-to-reference mode, the PPM detector, phase detector, and signal-detect circuits monitor the relationship of the reference clock to the receiver PLL and VCO output. If the frequency difference is within the prescribed PPM setting, the amplitude is within the prescribed limits, and the phase is within 0.08 UI, then the CRU switches to the lock-to-data mode.

In lock-to-data mode, the PLL uses a phase detector to keep the recovered clock aligned properly with the data. If the PLL does not stay locked to data because of problems such as frequency drift or severe amplitude attenuation, the receiver PLL locks back to the reference clock of the CRU to train the VCO. When the device is in lock-to-data mode (`rx_freqlocked` is asserted), the CRU is trying to align with incoming data and there is no phase relationship with the reference clock.

The `rx_freqlocked` signal indicates which mode the CRU is in (either lock-to-data or lock-to-reference mode). In lock-to-data mode, the `rx_freqlocked` signal is high. In lock-to-reference mode, the `rx_freqlocked` signal is low.

In lock-to-data mode, the `rx_freqlocked` signal is asserted and the `rx_pll_locked` signal loses its significance because the `rx_pll_locked` signal indicates that the CRU has locked to the reference clock. When the CRU is in lock-to-data mode, the phase of the VCO may differ from the reference clock, which may deassert the `rx_pll_locked` signal. You should ignore the `rx_pll_locked` signal when the `rx_freqlocked` signal is high.

In automatic lock mode, there are two conditions that force the CRU to fall out of lock-to-data mode.

- The serial data at the receiver input buffer is not within the prescribed voltage signal loss threshold. This condition is ignored if the Force signal detection option in the MegaWizard is enabled.
- The CRU PLL is not within the prescribed PPM frequency threshold setting (62.5, 100, 125, 200, 250, 300, 500, or 1,000 PPM) of the CRU reference clock.

If the CRU falls out of lock-to-data mode, the `rx_freqlocked` signal is deasserted. You can also deassert the `rx_freqlocked` signal by asserting either `rx_analogreset` (powers down the receiver) or `gxb_powerdown` (powers down all four channels of the transceiver block).

Manual Lock Options

Two optional input pins (`rx_locktorefclk` and `rx_locktodata`) allow you to control whether the CRU PLL automatically or manually switches between lock-to-reference mode and lock-to-data modes. This enables you to bypass the default automatic switchover circuitry if either `rx_locktorefclk` or `rx_locktodata` is instantiated.

When the `rx_locktorefclk` signal is asserted, it forces the CRU PLL to lock to the reference clock (`rx_cruclk`). Asserting `rx_locktodata` forces the CRU PLL to lock to data. This occurs whether the CRU is ready or not. When both signals are asserted, the `rx_locktodata` signal takes precedence over the `rx_locktorefclk` signal.

The signal loss threshold detector, PPM threshold frequency detector, and phase relationship detector reaction times may be too long for some applications. You can manually control the CRU to reduce CRU lock times using the `rx_locktorefclk` and `rx_locktodata` ports. Using the manual mode may reduce the time it takes for the CRU to switch from lock-to-reference mode to lock-to-data mode. You can assert the `rx_locktorefclk` to initially train the CRU. The `rx_locktodata` signal should be asserted after training the CRU.

When the `rx_locktorefclk` signal is asserted, the `rx_freqlocked` signal does not have any significance because it is low, indicating that the CRU is in lock-to-reference mode. If lock-to-data mode is asserted, the `rx_freqlocked` signal is always asserted, indicating that the CRU is in lock-to-data mode. When both signals are asserted, lock-to-data mode takes precedence. If both signals are deasserted, the CRU switchover is in automatic mode. [Table 2–17](#) shows a summary of the control signals.

Table 2–17. CRU User Control Lock Signals

<code>rx_locktorefclk</code>	<code>rx_locktodata</code>	CRU Mode
1	0	Lock-to-reference clock
x	1	Lock to data
0	0	Automatic

Deserializer

The deserializer block converts incoming high-speed serial data streams to 8-, 10-, 16-, or 20-bit-wide parallel data synchronized to the recovered clock of the CRU. Use the 8- and 10-bit operations, which support a data rate from 600 Mbps to 3.125 Gbps, in the single-width mode. Use the 16- and 20-bit operations, which support a data rate from 1 Gbps to 6.375 Gbps, for the double-width mode.

The deserializer block drives the parallel data to the pattern detector and word aligner, as shown in Figure 2-53. The deserializer block output bus data rate is the input data rate divided by the width of the output data bus. For example, for a 10-bit bus and a serial input data rate of 2.5 Gbps, the parallel data rate is $2.5 \div 10$ or 250 MHz. The first bit into the deserializer block is the LSB of the data bus out.

Figure 2-53. Deserializer Block in 8-Bit Mode

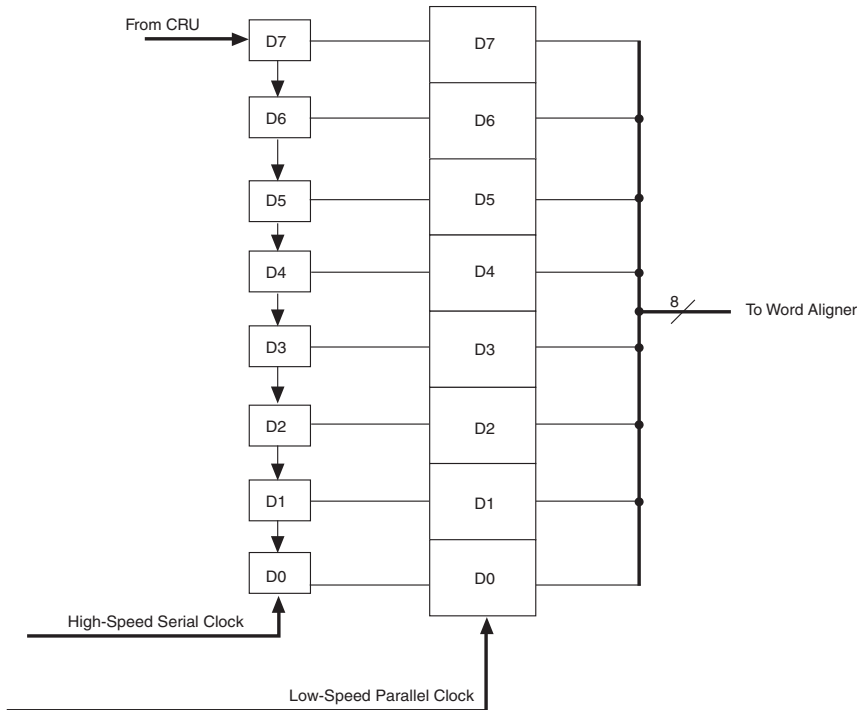
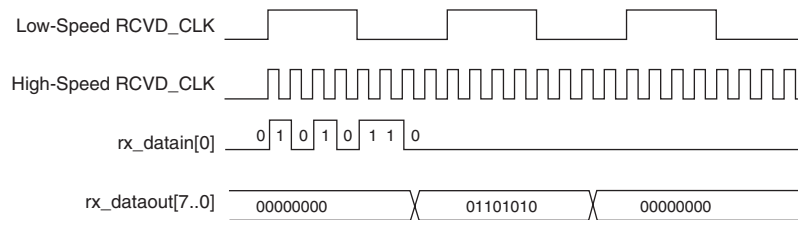


Figure 2-54 shows the serial bit order of the deserializer block input and the parallel data out of the deserializer block. Figure 2-54 shows a serial stream (01101010) deserialized into a value 8'h6A (01101010). The serial data is received LSB to MSB.



In Quartus II software version 7.1 and later, basic single width allows 8-bit deserializer (disable 8B/10B).

Figure 2–54. Deserializer Bit Order

Generic Receiver Polarity Inversion

The positive and negative signals of a serial differential link are often erroneously swapped during board layout. Solutions like a board re-spin or major updates to the PLD logic can prove expensive. The receiver polarity inversion feature is provided to correct this situation.

An optional `rx_invpolarity` port is available in all single-width and double-width modes except (OIF) CEI PHY to dynamically enable the receiver polarity inversion feature. In single-width modes, a high value on the `rx_invpolarity` port inverts the polarity of every bit of the 8-bit or 10-bit input data word to the word aligner in the receiver data path. In double-width modes, a high value on the `rx_invpolarity` port inverts the polarity of every bit of the 16-bit or 20-bit input data word to the word aligner in the receiver data path. Since inverting the polarity of each bit has the same effect as swapping the positive and negative signals of the differential link, correct data is seen by the receiver. The `rx_invpolarity` is a dynamic signal and may cause initial disparity errors in an 8B/10B encoded link. The downstream system must be able to tolerate these disparity errors.

The generic receiver polarity inversion feature is different from the PIPE 8B/10B polarity inversion feature. The generic receiver polarity inversion feature inverts the polarity of the data bits at the input of the word aligner. The PIPE 8B/10B polarity inversion feature inverts the polarity of the data bits at the input of the 8B/10B decoder and is available only in PIPE mode. Enabling the generic receiver polarity inversion and the PIPE 8B/10B polarity inversion simultaneously is not allowed in PIPE mode.

Figure 2–55 illustrates the receiver polarity inversion feature in single-width 10-bit wide data path configuration.

Figure 2–55. Receiver Polarity Inversion in Single-Width Mode

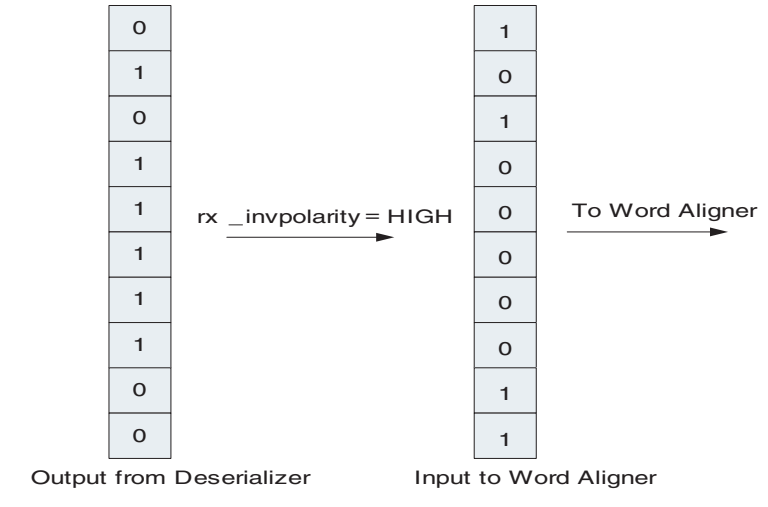
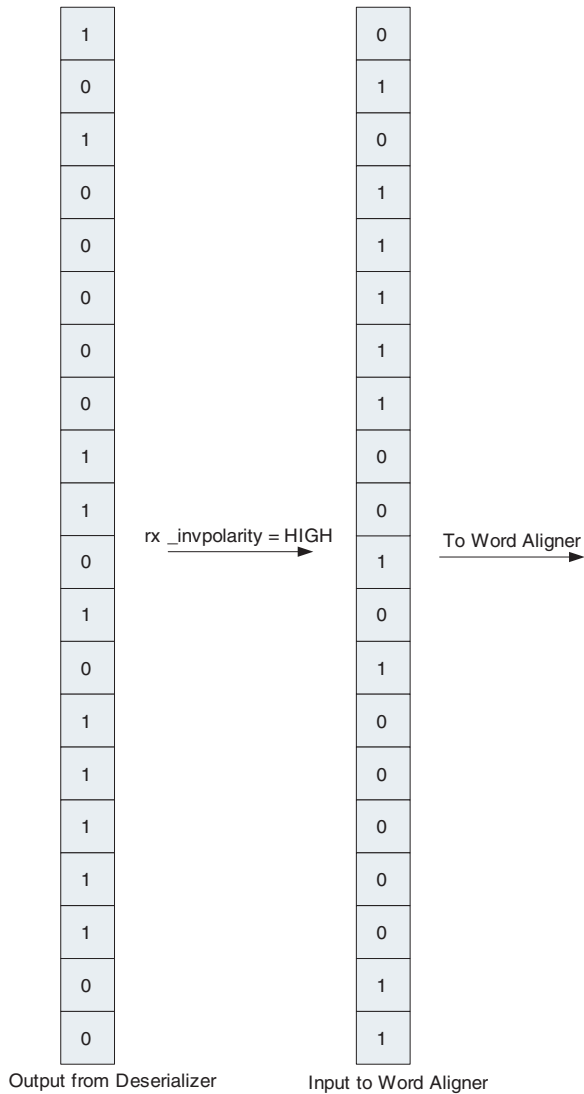


Figure 2–56 illustrates the receiver polarity inversion feature in double-width 20-bit wide data path configuration.

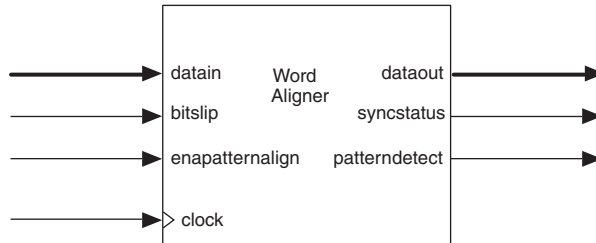
Figure 2–56. Receiver Polarity Inversion in Double-Width Mode



Word Aligner

The word aligner (refer to [Figure 2-57](#)) is part of the Stratix II GX transceiver digital blocks and is located in the receiver path between the deserializer and the de-skew FIFO buffer. The word aligner restores the byte boundary of the upstream transmitter based on a programmable alignment pattern that appears in the serial data stream.

Figure 2-57. Word Aligner



The word aligner block consists of four main sub-blocks:

- Aligner block
- Pattern detect block
- Manual bit-slip block
- Run-length checker

There are three modes in which the word aligner works: single-width mode, double-width mode, and automatic synchronization state machine mode. The following sections explain each of the blocks in each mode of operation. The word aligner cannot be bypassed and must be used. However, you can use the `rx_enapatternalign` port to set the word alignment to not align to the pattern.

Single-Width Mode

In single-width mode, there are three blocks active in the word aligner:

- Pattern detector
- Manual word aligner
- Automatic synchronization state machine

The pattern detector detects if the pattern exists in the current word boundary. The manual alignment identifies the alignment pattern across the byte boundaries and aligns to the correct byte boundary. The synchronization state machine detects the number of alignment patterns and good code groups for synchronization and goes out of

synchronization if code group errors (bad code groups) are detected. Figure 2–58 and Table 2–18 show the modes available in the single-width mode and the supported alignment modes.

Figure 2–58. Word Aligner Components in Single-Width Mode

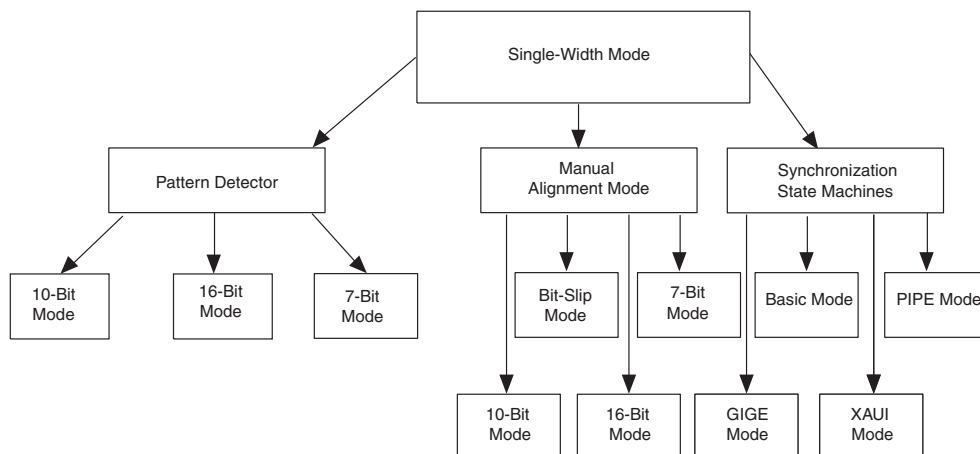


Table 2–18. Word Alignment Modes

Word Alignment Mode	Effective Mode	Control Signals	Status Signals
Synchronization state machine	PCI Express, XAUI, GIGE, Serial RapidIO, CPRI, or Basic	Automatically controlled to adhere to the specified standard or by user entered parameter	rx_syncstatus rx_patterndetect
Manual 7- and 10-bit alignment mode	Alignment to detected pattern when allowed by the rx_enapatternalign signal	rx_enapatternalign	rx_syncstatus rx_patterndetect
Manual 16-bit alignment mode	Alignment to detected pattern when allowed by the rx_enapatternalign signal	rx_enapatternalign	rx_syncstatus rx_patterndetect
Manual bit-slipping alignment mode	Manual bit slip controlled by the PLD logic array	rx_bitslip	rx_patterndetect

Pattern Detector Module

The pattern detector matches a pre-defined alignment pattern to the current byte boundary. When the pattern detector locates the alignment pattern, the optional `rx_patterndetect` signal is asserted for the duration of one clock cycle to signify that the alignment pattern exists in the current word boundary. The pattern detector module only indicates that the signal exists and does not modify the word boundary. Modification of the word boundary is discussed in the sections “Manual Alignment Modes” on page 2–75 and “Synchronization State Machine Mode” on page 2–81.

In the MegaWizard, you can program a 7-bit, 10-bit, or 16-bit pattern for the pattern detector to recognize. The pattern used for pattern matching is automatically derived from the word alignment pattern in the MegaWizard. For the 7-bit and 10-bit patterns, the actual alignment pattern specified in the MegaWizard and its complement are checked. For the 16-bit alignment pattern, only the actual pattern is checked. Table 2–19 shows the supported alignment patterns.

Pattern Detect Mode	Supported Protocols	Pattern Checked
7 bit	Basic, CPRI	Actual and complement
10 bit	Basic, XAUI, GIGE, CPRI, Serial RapidIO, and PIPE	Actual and complement
Two consecutive 8-bit characters	SONET/SDH	Actual



In 8B/10B encoded data, actual and complement pattern indicates positive and negative disparities.

Each bit in the `rx_patterndetect` and `rx_syncstatus` signal indicates the status of the group of 8 or 10 bits in the `rx_dataout` port. For the `rx_dataout` width of 8/10, 16/20, and 32/40 bits, the `rx_patterndetect` and `rx_syncstatus` widths are 1, 2, and 4 bits, respectively. An example of corresponding signals for `rx_dataout` (widths of 16 and 32) is shown in Table 2–20.

Data Width	Signal	Corresponding Signal
16	<code>rx_patterndetect [1]</code>	<code>rx_dataout [15:8]</code>
	<code>rx_patterndetect [0]</code>	<code>rx_dataout [7:0]</code>

Table 2–20. Corresponding Signals for rx_dataout (Part 2 of 2)

Data Width	Signal	Corresponding Signal
32	rx_patterndetect [3]	rx_dataout [31:24]
	rx_patterndetect [2]	rx_dataout [23:16]
	rx_patterndetect [1]	rx_dataout [15:8]
	rx_patterndetect [0]	rx_dataout [7:0]

7-Bit Pattern Mode

In the 7-bit pattern detection mode (use this mode with 8B/10B code), the pattern detector matches the seven LSBs of the 10-bit alignment pattern, which you specified in your ALT2GXB custom megafunction variation, in the current word boundary. Both positive and negative disparities are also checked in this mode.

The 7-bit pattern mode can mask out the three MSBs of the data, which allows the pattern detector to recognize multiple alignment patterns. For example, in the 8B/10B encoded data, a /K28.5/ (b'0011111010), /K28.1/ (b'0011111001), and /K28.7/ (b'0011111000) share seven common LSBs. Masking the three MSBs allows the pattern detector to resolve all three alignment patterns and indicate them on the rx_patterndetect port.

In 7-bit pattern mode, the word aligner still aligns to a 10-bit word boundary. The specified 7-bit pattern forms the least significant seven bits of the 10-bit word.

10-Bit Pattern Mode

In the 10-bit pattern detection mode (use this mode with 8B/10B code), the module matches the 10-bit alignment pattern you specified in your ALT2GXB custom megafunction variation with the data and its complement in the current word boundary. Both positive and negative disparities are checked by the pattern checker in this mode. For example, if you specify a /K28.5/ (b'0011111010) pattern as the comma, rx_patterndetect is asserted if b'0011111010 or b'1100000101 is detected in the incoming data.

16-bit Pattern Mode

You specify the 16-bit alignment pattern in the MegaWizard and it is oriented with the MSB first and the LSB last. A1 represents the least significant byte, which consists of bits [7 . . 0]. A2 represents the most significant byte, which consists of bits [15 . . 8]. Therefore, the alignment pattern is specified as [A2,A1] in the MegaWizard. Only the actual alignment pattern specified in the MegaWizard is detected in this mode.

The pattern detector is defaulted to the A1A2 mode. Only the positive disparity is detected in this mode.

Manual Alignment Modes

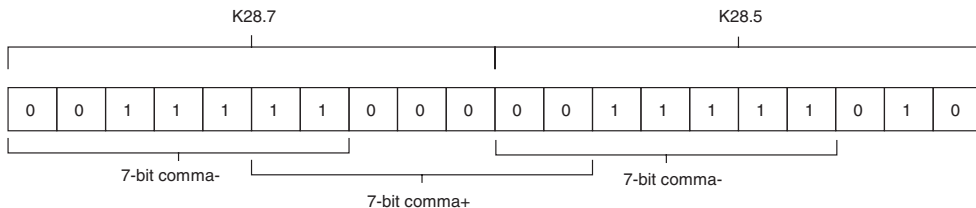
The word aligner has three manual alignment modes (7-, 10-, and 16-bits) when the transceiver data path is in single-width mode. The 10- and 7-bit alignment modes are used with 8B/10B code and the 16-bit alignment mode is for scrambled or non-scrambled data.

7-bit Alignment Mode

In the 7-bit alignment mode (use the 8B/10B encoded data with this mode), the module looks for the 7-bit alignment pattern you specified in the MegaWizard Plug-In Manager in the incoming data stream. The 7-bit alignment mode is useful because it can mask out the three most significant bits of the data, which allows the word aligner to align to multiple alignment patterns. For example, in the 8B/10B encoded data, a /K28.5/ (b'0011111010), /K28.1/ (b'0011111001), and /K28.7/ (b'0011111000) share seven common LSBs. Masking the three MSBs allows the word aligner to resolve all three alignment patterns synchronized to it. The word aligner places the boundary of the 7-bit pattern in the LSByte position with bit positions [0 . . 7]. The true and complement of the patterns is checked.

Use the `rx_enapatternalign` port to enable the 7-bit manual word alignment mode. When the `rx_enapatternalign` signal is high, the word aligner detects the specified alignment patterns and realigns the byte boundary if needed. The `rx_syncstatus` port is asserted for one parallel clock cycle to signify that the word boundary was detected across the current word boundary and has synchronized to the new boundary, if a rising edge was detected previously on the `rx_enapatternalign` port. You must differentiate if the acquired byte boundary is correct, because the 7-bit pattern can appear between word boundaries. For example, in the standard 7-bit alignment pattern -7'b1111100, if a K28.7 is followed by a K28.5, the 7-bit alignment pattern appears on K28.7, between K28.7 and K28.5, and also again in K28.5 (refer to [Figure 2–59](#)).

Figure 2–59. Cross Boundary 7-Bit Comma When /K28.7 is Followed by /K28.5



Manual 10-Bit Alignment Mode

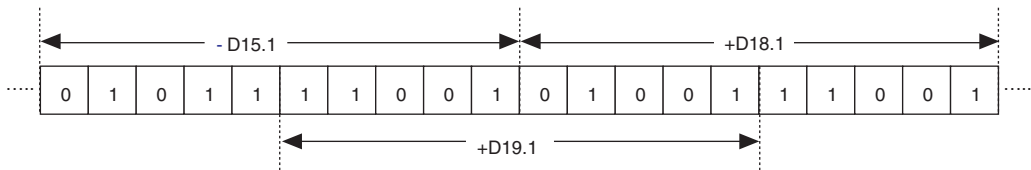
You can configure the word aligner to align to a 10-bit word boundary. The internal word alignment circuitry shifts to the correct word boundary if the alignment pattern specified in the pattern detector is detected in the data stream.

The `rx_enapatternalign` port enables the word alignment in the manual 10-bit alignment mode. When the `rx_enapatternalign` signal is high, the word aligner detects the specified alignment pattern and reasserts the byte boundary if necessary. The `rx_syncstatus` port is asserted for one parallel clock cycle to signify that the word boundary has been detected across the word boundary and has synchronized to the new boundary.

The `rx_enapatternalign` signal is held high if the alignment pattern is known to be unique and does not appear across the byte boundaries of other data. For example, if an 8B/10B encoding scheme guarantees that the `/K28.5/` code group is a unique pattern in the data stream, the `rx_enapatternalign` port is held at a constant high.

If the alignment pattern can exist between word boundaries, the `rx_enapatternalign` port must be controlled by the user logic in the PLD to avoid false word alignment. For example, assume that 8B/10B is used and a `/+D19.1/` (`b'110010 1001`) character is specified as the alignment pattern. In this case, a false word boundary is detected if a `/-D15.1/` (`b'010111 1001`) is followed by a `/+D18.1/` (`b'010011 1001`). Refer to [Figure 2-60](#).

Figure 2-60. False Word Boundary Alignment if Alignment Pattern Exists Across Word Boundaries, Single Width



In this example, the `rx_enapatternalign` signal is deasserted after the word aligner locates the initial word alignment to prevent false word boundary alignment. When the `rx_enapatternalign` signal is deasserted, the current word boundary is locked even if the alignment pattern is detected across different boundaries. In this case, the `rx_syncstatus` acts as a re-synchronization signal to signify that the

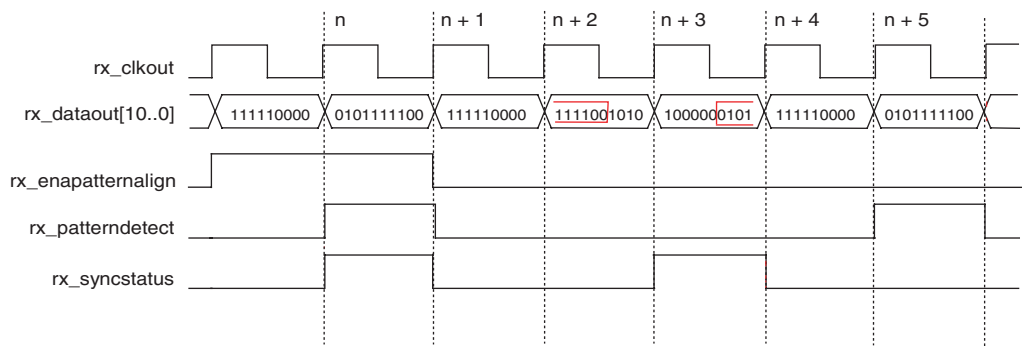
alignment pattern was detected, but the boundary is different than the current boundary. You must monitor this signal and reassert the `rx_enapatternalign` signal if realignment is desired.

Figure 2–61 shows an example of how the word aligner signals interact in 10-bit alignment mode. In this example, a `/K28.5/ (10'b0011111010)` is specified as the alignment pattern. The `rx_enapatternalign` signal is held high at time n , so alignment occurs whenever an alignment pattern exists in the pattern. The `rx_patterndetect` signal is asserted for one clock cycle to signify that the pattern exists on the re-aligned boundary. The `rx_syncstatus` signal is also asserted for one clock cycle to signify that the boundary has been synchronized. At time $n + 1$, the `rx_enapatternalign` signal is deasserted to instruct the word aligner to lock the current word boundary.

The alignment pattern is detected at time $n + 2$, but it exists on a different boundary than the current locked boundary. The bit orientation of the Stratix II GX device is LSB to MSB, so the alignment pattern exists across time $n + 2$ and $n + 3$ (refer to Figure 2–61). In this condition the `rx_patterndetect` remains low because the alignment pattern does not exist on the current word boundary, but the `rx_syncstatus` signal is asserted for one clock cycle to signify a resynchronization condition. This means that the alignment pattern has been detected across another word boundary.

The user logic design in the PLD must decide whether or not to assert the `rx_enapatternalign` to re-initiate the word alignment process. At time $n + 5$ the `rx_patterndetect` signal is asserted for one clock cycle to signify that the alignment pattern has been detected on the current word boundary.

Figure 2–61. Word Aligner Symbol Interaction in 10-Bit Manual Alignment Mode



Manual SONET/SDH Alignment Mode (Two Consecutive 8-bit Characters (A1A2) or Four Consecutive 8-bit Characters (A1A1A2A2))

The word aligner can be configured to align to a 16-bit word boundary in SONET/SDH protocol mode. In the SONET/SDH protocol mode, the word aligner either aligns to two consecutive 8-bit characters (A1A2) or four consecutive 8-bit characters (A1A1A2A2). The `rx_a1a2size` signal can be used to differentiate between the two and four consecutive modes. The word aligner aligns to the A1A2 pattern when the `rx_a1a2size` signal is held low "0," or to the A1A1A2A2 when `rx_a1a2size` is high "1." The `rx_a1a2sizeout` port sends the state of the `rx_a1a2size` signal as seen by the word aligner back to the PLD logic array.

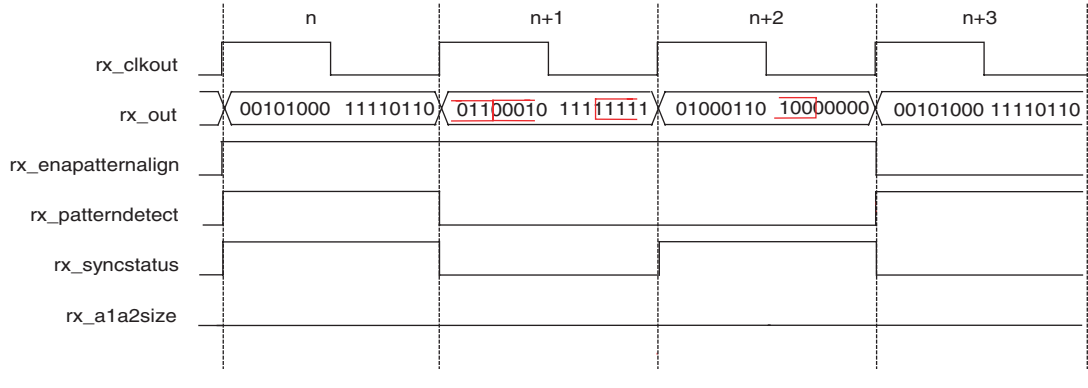
Word alignment is enabled or re-enabled by the `rx_enapatternalign` signal, but the behavior is different than that described for the 7-bit or 10-bit manual mode in single-width configuration. In the 7/10-bit mode the byte boundary can be dynamically changed if the `rx_enapatternalign` signal is held high. However, in the SONET/SDH mode the byte boundary is locked after the first alignment pattern is detected and aligned after the rising edge of the `rx_enapatternalign` signal. If the byte boundary changes, the `rx_enapatternalign` signal must be deasserted and reasserted to re-enable the alignment circuit. This feature is valuable in SONET/SDH because the data is scrambled and not encoded. The alignment pattern can potentially exist across byte boundaries and can trigger a false realignment. In SONET/SDH the byte boundary should be aligned and locked at the beginning of a SONET/SDH frame since the A1A2 alignment pattern resides in the framing section at the beginning of the transport overhead.

Initially, the word aligner locks onto the first alignment pattern detected. In this scenario the `rx_patterndetect` signal is asserted for one clock cycle to signify that the alignment pattern has been aligned. The `rx_syncstatus` signal is asserted for a clock cycle to signify that the word boundary has been synchronized. After the word boundary has been locked, regardless of whether the `rx_enapatternalign` signal is held high or low, the `rx_syncstatus` signal asserts for one clock cycle whenever the alignment pattern is detected across a different byte boundary. The `rx_syncstatus` signal operates in this resynchronization state until a rising edge is detected on `rx_enapatternalign`.

Figure 2–62 shows an example of how the word aligner signals interact in SONET/SDH alignment mode for an A1A2 pattern. For this example, a SONET/SDH A1A2 framing pattern uses 16'hF628 (16'b1111011000101000) with the reverse bit ordering. This option reverses

the bit order so that the data can be transmitted and received MSB to LSB. If the bit reversal option in the MegaWizard Plug-In Manager is not used, the transceiver transmits and receives LSB to MSB.

Figure 2–62. Word Aligner Symbol Interaction in SONET/SDH A1A2 Manual Alignment Mode



The `rx_a1a2size` signal is held low, which sets the SONET/SDH alignment mode to A1A2. The `rx_enapatternalign` signal is toggled high at time n , so the aligner locks to the boundary of the next present alignment pattern. The A1A2 alignment pattern appears on the `rx_dataout` port during this period. The alignment pattern exists, so the `rx_patterndetect` and `rx_syncstatus` signals are asserted for one clock cycle to signify that the A1A2 alignment pattern has been detected and the word boundary has been locked. The A1A2 alignment pattern appears again across word boundaries in during periods $n+1$, and $n+2$. The `rx_enapatternalign` signal is held high, but the word aligner does not re-align the byte boundary as it would in 10-bit manual alignment mode. Instead the `rx_syncstatus` signal is asserted for one clock cycle to signify a re-synchronization condition. You must deassert and reassert the `rx_enapatternalign` signal to re-trigger the word aligner and align on the next alignment pattern. The next transition occurs at time $n+3$, where `rx_enapatternalign` is deasserted and the A1A2 pattern is present on the `rx_dataout` port. The word aligner then asserts the `rx_patterndetect` signal for one clock cycle to flag the detection of the alignment pattern on the current word boundary.

Manual 16-bit Alignment Mode

You enable the 16-bit alignment mode in the single-width mode. This mode aligns to the 16-bit alignment pattern you specified in the MegaWizard.

The byte boundary is locked after the first alignment pattern is detected and after the rising edge of the `rx_enapatternalign` port. If the byte boundary changes, the `rx_enapatternalign` port is deasserted and reasserted to enable the alignment circuit to search for and align to the next available alignment pattern.

On the rising edge of the `rx_enapatternalign` port, the word aligner locks onto the first alignment pattern detected. In this scenario, `rx_patterndetect` is asserted to signify that the alignment pattern has been aligned. The `rx_syncstatus` signal is also asserted for one clock cycle to signify that the word boundary has been synchronized. After the word boundary is locked, whether or not `rx_enapatternalign` is held high or low, the `rx_syncstatus` signal asserts for one clock cycle whenever the alignment pattern is detected across a different byte boundary. The `rx_syncstatus` signal operates in this resynchronization state until a rising edge is detected on the `rx_enapatternalign` port.

Manual Bit-Slip Alignment Mode

You can also achieve word alignment by enabling the manual bit-slip option in the MegaWizard. With this option enabled, the transceiver shifts the word boundary MSB to LSB one bit every parallel clock cycle. The transceiver shifts the word boundary every time the bit-slipping circuitry detects a rising edge of the `rx_bitslip` signal. At each rising edge of the `rx_bitslip` signal, the word boundary slips one bit. The bit that arrives at the receiver first is skipped. When the word boundary matches the alignment pattern you specified in the MegaWizard, the `rx_patterndetect` signal is asserted for one clock cycle. You must implement the logic in the PLD logic array to control the bit-slip circuitry.

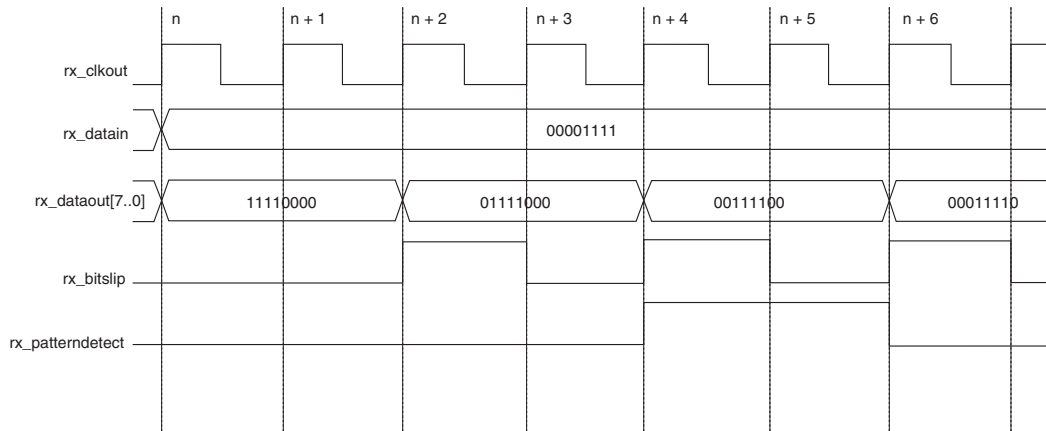
The bit slipper is useful if the alignment pattern changes dynamically when the Stratix II GX device is in user mode. You can implement the controller in the logic array, so you can build a custom controller to dynamically change the alignment pattern without needing to reprogram the Stratix II GX device.

Figure 2–63 shows an example of how the word aligner signals interact in the manual bit-slip alignment mode. For this example, `8'b00111100` is specified as the alignment pattern and an `8'b11110000` value is held at the `rx_datain` port.

Every rising edge on the `rx_bitslip` port causes the `rx_dataout` data to shift one bit from the MSB to the LSB by default. This is shown at time $n + 2$ where the `8'b11110000` data is shifted to a value of `8'b01111000`. At this state the `rx_patterndetect` signal is held low because the specified alignment pattern does not exist in the current word boundary.

The `rx_bitslip` is disabled at time $n + 3$ and re-enabled at time $n + 4$. The output of the `rx_dataout` now matches the specified alignment pattern, thus the `rx_patterndetect` signal is asserted for one clock cycle. At time $n + 5$, the `rx_patterndetect` signal is still asserted because the alignment pattern still exists in the current word boundary. Finally, at time $n + 6$ the `rx_dataout` boundary is shifted again and the `rx_patterndetect` signal is deasserted to signify that the word boundary does not contain the alignment pattern.

Figure 2–63. Word Aligner Symbol Interaction in Manual Bit-Slip Mode

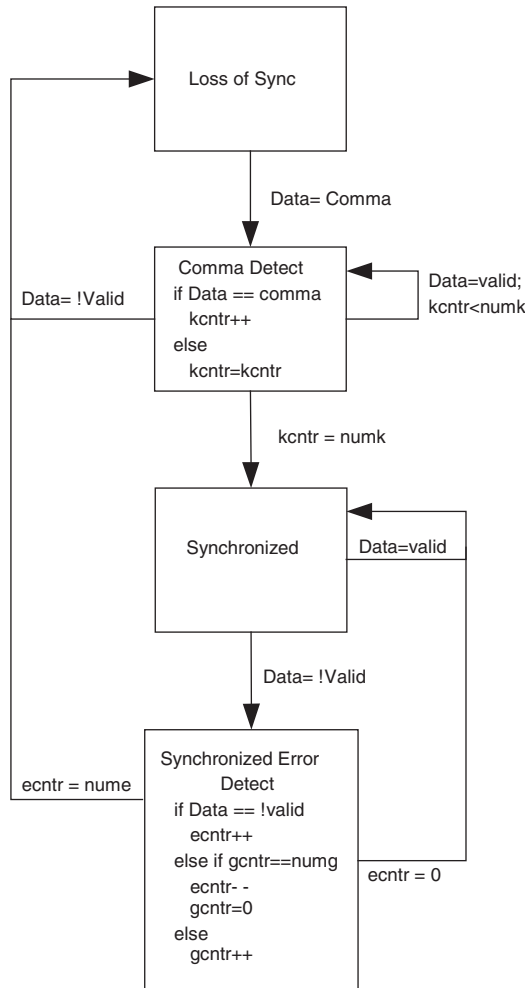


Synchronization State Machine Mode

In single-width mode, you can choose to have the link synchronization handled by a state machine. Unlike the manual alignment mode where there is no built-in hysteresis to go into or fall out of synchronization, the synchronization state machine offers automatic detection of a valid number of alignment patterns and synchronization and detection of code group errors for automatically falling out of synchronization. The synchronization state machine is available in the Basic (single-width mode only), XAUI, GIGE, and PIPE modes. For the XAUI, GIGE, and PIPE modes, the number of alignment patterns, consecutive code groups, and bad code groups are fixed. You must use the 8B/10B code for the synchronization state machine. In XAUI, GIGE, and PIPE modes, the 8B/10B encoder/decoder is embedded in the transceiver data path. In Basic single-width mode, you can configure the MegaWizard to either use or bypass the 8B/10B encoder/decoder in the transceiver. If the synchronization state machine is enabled and the 8B/10B encoder/decoder is bypassed, the 8B/10B encoder/decoder logic must be implemented outside the transceiver as a requirement for using the synchronization state machine.

In Basic mode, you can configure the state machine to suit a variety of standard and custom protocols. In the MegaWizard, you can program the number of alignment patterns to acquire link synchronization. You can program the number of bad code groups to fall out of synchronization. You can program the number of good code groups to negate a bad code group. You enter these values in the MegaWizard. The `rx_syncstatus` port indicates the link status. A high level indicates link synchronization is achieved, a low level indicates that synchronization has not yet been achieved or that there were enough code group errors to fall out of synchronization. [Figure 2-64](#) shows a flowchart of the synchronization state machine.

Figure 2-64. Word Aligner Synchronization State Machine Flow Chart



The maximum value for the number of valid alignment patterns and good code groups is 256. The maximum value of invalid or bad code groups to fall out of synchronization is 8. For example, if 3 is set for the number of good code groups, then when 3 consecutive good code groups are detected after a bad code group, the effect of the bad code group on synchronization is negated. This does not negate the bad code group that actually triggers the loss of synchronization. To negate a loss of synchronization, the protocol-defined number of alignment patterns must be received.

When either XAUI or GIGE mode is used, the synchronization and word alignment is handled automatically by a built-in state machine that adheres to either the IEEE 802.3ae or IEEE 802.3 synchronization specifications, respectively. If you specify either standard, the alignment pattern is automatically defaulted to /K28.5/ (b'0011111010).

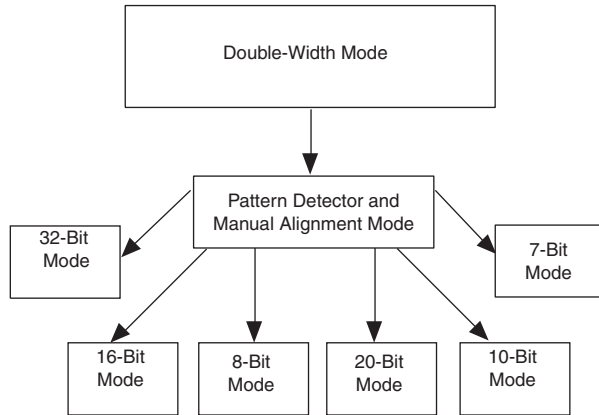
When you specify the XAUI protocol, code-group synchronization is achieved upon the reception of four /K28.5/ commas. Each comma can be followed by any number of valid code groups. Invalid code groups are not allowed during the synchronization stage. When code-group synchronization is achieved the optional `rx_syncstatus` signal is asserted. Refer to clause 47-48 of the IEEE P802.3ae standard or “XAUI Mode” on page 2-167 for more information regarding the operation of the synchronization phase.

If you specify the GIGE protocol, code-group synchronization is achieved upon the reception of three consecutive ordered sets. An ordered set starts with the /K28.5/ comma and can be followed by an odd number of valid data code groups. Invalid code groups are not allowed during the reception of three ordered-sets. When code-group synchronization is achieved the optional `rx_syncstatus` signal is asserted.

In PIPE mode, lane synchronization is achieved when the word aligner sees 4 good /K28.5/ commas and 16 good code groups. This is accomplished through the reception of 4 good PCI Express training sequences (TS1 or TS2). The PCI-Express fast training sequence (FTS) can also be used to achieve lane or link synchronization, but requires at least five of these training sequences. The `rx_syncstatus` signal is asserted when synchronization is achieved and is deasserted when the word aligner receives 23 code group errors.

Double-Width Mode

In the double-width mode, there are two blocks active in the word aligner: the pattern detector and manual alignment mode. The pattern detector detects if the pattern exists in the current word boundary. The manual alignment identifies the alignment pattern across the byte boundaries and aligns to the correct byte boundary. There are no synchronization state machines available for the double-width mode.

Figure 2–65. Word Aligner Components, Double-Width Mode**Pattern Detector Module**

The pattern detector matches a pre-defined comma to the current word boundary. If the pattern detector locates the correct alignment pattern, the optional `rx_patterndetect` signal is asserted for the duration of one clock cycle to signify that the alignment pattern exists in the current word boundary. The pattern detector module only indicates that the signal exists and does not modify the word boundary.

You can program and then specify in the MegaWizard a 7-bit, 8-bit, 10-bit, 16-bit, 20-bit, or 32-bit, pattern for the pattern detector to recognize in double-width mode. For the 7-bit, 10-bit, and 20-bit patterns, the actual and complement of the alignment patterns are checked and used with 8B/10B coding. For the 8-bit, 16-bit, and 32-bit alignment patterns, only the actual patterns are checked, not the complements. These patterns are used for scrambled coding or non-encoded data.

7-Bit Pattern Mode

In the 7-bit pattern detection mode, the pattern detector matches the seven least significant bits of the 10-bit alignment pattern in the LSByte as specified in the MegaWizard in the current word boundary. The pattern detector checks both positive and negative disparities in this mode.

The 7-bit pattern mode can mask out the three most significant bits of the data, which allows the pattern detector to recognize multiple alignment patterns. For example, in the 8B/10B encoded data, a /K28.5/ (b'0011111010), /K28.1/ (b'0011111001), and /K28.7/ (b'0011111000) share seven common LSBs, so masking the three MSBs allows the pattern

detector to resolve all three alignment patterns and indicate it on the `rx_patterndetect` port. If the alignment pattern appears on the MSByte, no actions are taken by the pattern detect module.

8-Bit Pattern Mode

You can enable a single 8-bit character (A1) detection in the MegaWizard. In this mode, the pattern detector detects a single 8-bit alignment pattern character in the LSByte of the data path. If the alignment pattern appears in the MSByte, no action is taken by the pattern detector.

10-Bit Pattern Mode

In the 10-bit pattern detection mode, the pattern detector matches the 10-bit alignment pattern you specified in the MegaWizard to the LSByte data and its complement in the current word boundary. The pattern detector checks both positive and negative disparities in this mode. For example, if you specified a `/K28.5/ (b'0011111010)` pattern as the alignment pattern, the `rx_patterndetect` signal is asserted if `b'0011111010` or `b'1100000101` is detected in the incoming data. If the alignment pattern appears on the MSByte, no action is taken by the pattern detector.

16-Bit Pattern Mode

You enable the two consecutive 8-bit characters (A1A2).

You specify the 16-bit alignment pattern, which has the bit orientation of `[MSB . . LSB]`, in the MegaWizard. A1 represents the least significant byte, which consists of bits `[7 . . 0]`. A2 represents the most significant byte, which consists of bits `[15 . . 8]`. Therefore, the alignment pattern is specified as `[A2,A1]` in the MegaWizard. Only the actual alignment pattern you specified in the MegaWizard is detected in this mode.

20-Bit Pattern Mode

In the 20-bit pattern detection mode, the pattern detector matches the 20-bit comma (K1K2) you specified in the MegaWizard to the incoming data stream. The pattern detector checks the true and complement of the pattern. For example, if you specify a `/K28.5/ and /K28.0/ (10'b0011111010, 10'b0011110100)` pattern as the alignment pattern, the `rx_patterndetect` signal is asserted if `10'b0011111010` or `10'b1100000101` and `10'b0011110100` or `10'b1100001011` are detected in the incoming data. You do not need to enter the correct disparity in the MegaWizard, because the true and complement of each code group is checked. In this mode, only `Kx.y` codes are used as the true and complement to represent the same code group (but different disparity). The `Dx.y` code group does not necessarily use its true and complement to represent the same code group.

32-Bit Pattern Mode

You enable the four consecutive 8-bit character (A1A2A3A4 or A1A1A2A2) detection in the MegaWizard. You specify the 32-bit alignment pattern in the MegaWizard and it is oriented with the MSB first and the LSB last. A1 represents the least significant byte, which consists of bits [7 . . 0]. A4 represents the most significant byte, which consists of bits [31 . . 24]. Therefore, the alignment pattern is specified as [A4,A3,A2,A1] in the MegaWizard. Only the actual alignment pattern you specified in the MegaWizard is detected in this mode.

Manual Alignment Modes

The word aligner has six manual alignment modes (7-, 8-, 10-, 16-, 20- and 32-bits) when the transceiver data path is in double-width mode. The 7-, 10-, and 20-bit alignment modes are used with 8B/10B encoded data. Both the actual and complement of the alignment pattern are checked for these modes. The 8-, 16-, and 32-bit alignment modes are for scrambled or non-scrambled data. Only the actual alignment pattern is checked for these modes.

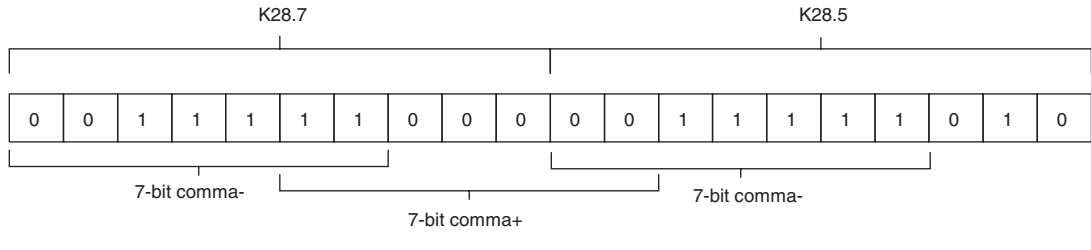
Manual 7-bit Alignment Mode

In the 7-bit alignment mode (use the 8B/10B encoded data with this mode), the module looks for the 7-bit alignment pattern you specified in the MegaWizard Plug-In Manager in the incoming data stream. The 7-bit alignment mode is useful because it can mask out the three most significant bits of the data, which allows the word aligner to align to multiple alignment patterns. For example, in the 8B/10B encoded data, a /K28.5/ (b'0011111010), /K28.1/ (b'0011111001), and /K28.7/ (b'0011111000) share seven common LSBs. Masking the three MSBs allows the word aligner to resolve all three alignment patterns synchronized to it. The word aligner places the boundary of the 7-bit pattern in the LSByte position with bit positions [0 . . 7]. The true and complement of the patterns is checked.

In 7-bit manual word alignment mode, the word aligner looks for the 7-bit alignment pattern after detecting a rising edge on the `rx_enapatternalign` signal. On finding the alignment pattern, the word aligner locks the word boundary and asserts the `rx_syncstatus` signal. The `rx_syncstatus` signal remains high until it sees another rising edge on the `rx_enapatternalign`. After detecting a rising edge on the `rx_enapatternalign` signal, the word aligner starts looking for the 7-bit word alignment pattern again and asserts the `rx_syncstatus` signal once it finds the 7-bit alignment pattern. You must differentiate if the acquired byte boundary is correct, because the 7-bit pattern can appear between word boundaries. For example, in the standard 7-bit

alignment pattern $-7'b1111100$, if a K28.7 is followed by a K28.5, the 7-bit alignment pattern appears on K28.7, between K28.7 and K28.5, and also again in K28.5 (refer to [Figure 2–66](#)).

Figure 2–66. Cross Boundary 7-Bit Comma When /K28.7 is Followed by /K28.5



Manual 8-bit Alignment Mode

You can enable the 8-bit alignment mode in the double-width mode. This mode aligns to the 8-bit alignment pattern you specified in the MegaWizard.

The byte boundary is locked after the first alignment pattern is detected and after the rising edge of the `rx_enapatternalign` signal. The detected pattern is placed in the LSByte of the 16-bit word. If the byte boundary changes, the `rx_enapatternalign` port must be deasserted and reasserted to enable the alignment circuit to search for and align to the next available alignment pattern. On the rising edge of the `rx_enapatternalign` signal, the word aligner locks onto the first alignment pattern detected and places the detected pattern in the data stream on the LSByte position. In this scenario, `rx_patterndetect` is asserted to signify that the alignment pattern has been aligned. The `rx_syncstatus` signal is also asserted to signify that the word boundary has been synchronized.

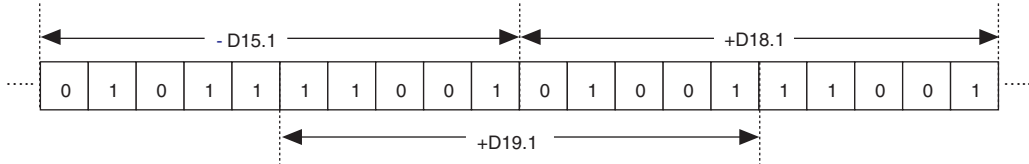
Manual 10-Bit Alignment Mode

You can configure the word aligner to align to a 10-bit word boundary. The internal word alignment circuitry shifts to the correct word boundary if the alignment pattern you specified in the pattern detector is detected in the data stream. The word aligner then puts the alignment pattern in the LSByte of the data path.

In 10-bit manual word alignment mode, the word aligner looks for the 10-bit alignment pattern after detecting a rising edge on the `rx_enapatternalign` signal. On finding the alignment pattern, the word aligner locks the word boundary and asserts the `rx_syncstatus` signal. The `rx_syncstatus` signal remains high until it sees another rising edge on the `rx_enapatternalign`. After detecting a rising edge

on the `rx_enapatternalign` signal, the word aligner starts looking for the 10-bit word alignment pattern again and asserts the `rx_syncstatus` signal once it finds the 10-bit alignment pattern. The `rx_enapatternalign` port only operates in an edge-sensitive fashion in double-width mode. You must deassert the `rx_enapatternalign` signal and assert it again for re-alignment. Altera recommends using the `/K28.5/` code group as one of the control codes for this alignment pattern. For example, assume that 8B/10B coding is used and a `+/D19.1/` (b'110010 1001) character is specified as the alignment pattern. In that case, a false word boundary is detected if a `-D15.1/` (b'010111 1001) is followed by a `+/D18.1/` (b'010011 1001). Refer to [Figure 2–67](#).

Figure 2–67. False Word Boundary Alignment if Alignment Pattern Exists Across Word Boundaries, Double Width

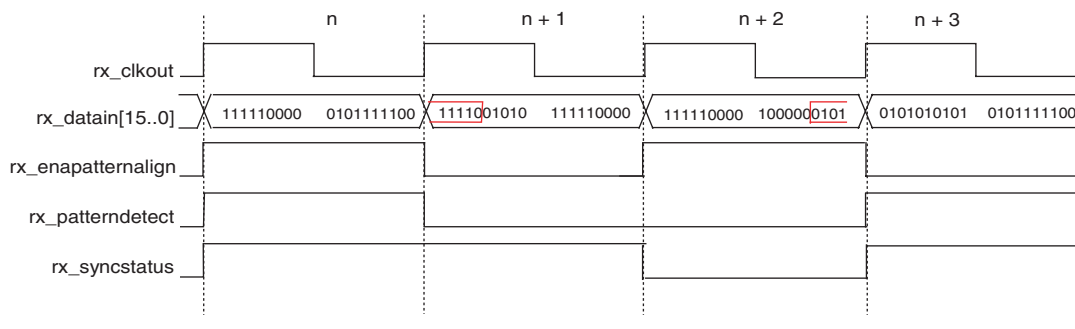


If there is no rising edge on the `rx_enapatternalign` port, the current word boundary is locked, even if the alignment pattern is detected across different boundaries.

[Figure 2–68](#) shows an example of how the word aligner signals interact in 10-bit alignment mode. For this example, a `/K28.5/` (10'b0011111010) is specified as the alignment pattern. The `rx_enapatternalign` signal is as a rising edge at time n , alignment occurs whenever an alignment pattern exists in the pattern. The `rx_patterndetect` signal is asserted for one clock cycle to signify that the pattern exists on the re-aligned boundary. The `rx_syncstatus` signal also gets asserted to signify that the boundary has been synchronized. At time $n + 1$, the `rx_enapatternalign` signal is deasserted and no rising edge occurs, which instructs the word aligner to lock the current word boundary. The alignment pattern is detected at time $n + 2$, but it exists on a different boundary than the current locked boundary. [Figure 2–68](#) shows that the alignment pattern exists across time $n + 1$ and $n + 2$. In this condition the `rx_patterndetect` signal remains low because the alignment pattern does not exist on the current word boundary. It is up to the user logic to decide whether or not to assert the `rx_enapatternalign` signal to re-initiate the word alignment process. At time $n + 3$, the `rx_patterndetect` signal is asserted for one clock cycle to signify that

the alignment pattern has been detected on the LSByte of the current word boundary. If the pattern exists on the MSByte, the `rx_syncstatus` signal goes high.

Figure 2–68. Word Aligner Symbols Interacting in 10-Bit Manual Alignment Mode



Manual 16-Bit Alignment Mode

You can enable the 16-bit alignment mode in the double-width mode. This mode aligns to the 16-bit alignment pattern you specified in the MegaWizard.

The byte boundary is locked after the pattern detector detects the first alignment pattern and then after the rising edge of the `rx_enapatternalign` port. If the byte boundary changes, the `rx_enapatternalign` port must be deasserted and reasserted to enable the alignment circuit to search for and align to the next available alignment pattern. On the rising edge of the `rx_enapatternalign` signal, the word aligner locks onto the first alignment pattern detected. In this scenario the `rx_patterndetect` signal is asserted to signify that the alignment pattern has been aligned. The `rx_syncstatus` signal is also asserted to signify that the word boundary has been synchronized.

Manual 20-bit Alignment Mode

In the 20-bit alignment mode, the pattern detector looks for the 20-bit alignment pattern (K1K2) you specified in the MegaWizard in the incoming data stream. The pattern detector checks the true and complement of the pattern. For example, if you specified a /K28.5/ and /K28.0/ (10'b0011111010, 10'b0011110100) pattern as the alignment pattern, the byte boundary is set to the pattern boundary when 10'b0011111010 or 10'b1100000101 and 10'b0011110100 or 10'b11000001011 are detected in the incoming data. It is not necessary to enter the correct disparity in the MegaWizard because the true and complement of each code group is checked automatically by the pattern detector. Do not use Dx.y codes as the alignment pattern in the 20-bit alignment mode.

In 20-bit manual word alignment mode, the word aligner looks for the 20-bit alignment pattern after detecting a rising edge on the `rx_enapatternalign` signal. On finding the alignment pattern, the word aligner locks the word boundary and asserts the `rx_syncstatus` signal. The `rx_syncstatus` signal remains high until it sees another rising edge on the `rx_enapatternalign`. After detecting a rising edge on the `rx_enapatternalign` signal, the word aligner starts looking for the 20-bit word alignment pattern again and asserts the `rx_syncstatus` signal once it finds the 20-bit alignment pattern. The `rx_enapatternalign` port can only operate in an edge-sensitive fashion in double-width mode. Deassertion of the `rx_enapatternalign` port is necessary for realignment. Altera recommends that you include the `/K28.5/` code group as one of the control codes in this alignment pattern.

Manual 32-Bit Alignment Mode

You can enable the 32-bit alignment mode in the double-width mode only. This mode aligns to the 32-bit alignment pattern you specified in the MegaWizard.

The byte boundary is locked after the first alignment pattern is detected and then after the rising edge of the `rx_enapatternalign` port. If the byte boundary changes, the `rx_enapatternalign` port must be deasserted and reasserted to enable the alignment circuit to search for and align to the next available alignment pattern. On the rising edge of `rx_enapatternalign`, the word aligner locks onto the first alignment pattern detected. In this scenario, the `rx_patterndetect` is asserted to signify that the alignment pattern has been aligned. The `rx_syncstatus` signal is asserted to signify that the word boundary has been synchronized.

Manual Bit-Slipping Alignment Mode

In the double-width mode, word alignment is also achieved by enabling the manual bit-slip option in the MegaWizard. This mode operates the same way as the bit slip in the single-width mode. With this option enabled, the transceiver shifts the word boundary one bit from the MSB to the LSB every parallel clock cycle. This occurs every time the bit-slipping circuitry detects a rising edge of the `rx_bitslip` signal. At each rising edge of `rx_bitslip`, the word boundary slips one bit. The bit that arrives at the receiver first is skipped. When the word boundary matches what you specified as the alignment pattern in the MegaWizard, the `rx_patterndetect` signal is asserted for one clock cycle. You must implement the logic in the PLD logic array to control the bit-slip circuitry.

Run-Length Violation Detection Circuit

The programmable run-length violation circuit resides in the word aligner block and detects consecutive 1s or 0s in the data. If the data stream exceeds the preset maximum number of consecutive 1s or 0s, the violation is signified by the assertion of the `rx_rlv` signal.

This signal is not synchronized to the parallel data and appears in the logic array earlier than the run-length violation data. To ensure that the PLD can latch this signal in systems where there are frequency variations between the recovered clock and the PLD logic array clock, the `rx_rlv` signal is asserted for a minimum of two clock cycles in single-width modes and a minimum of three clock cycles in double-width modes. The `rx_rlv` signal may be asserted longer, depending on the run-length of the received data.

In single-width mode, the run-length violation circuit detects up to a run length of 128 (for an 8-bit deserialization factor) or 160 (for a 10-bit deserialization factor). The settings are in increments of 4 or 5 for the 8-bit or 10-bit deserialization factors, respectively.

In double-width mode, the run-length violation circuit maximum run-length detection is 512 (with a run-length increment of 8) and 640 (with a run-length increment of 10) for the 16-bit and 20-bit deserialization factors, respectively.

Table 2–21 summarizes the detection capabilities of the run-length violation circuit.

Data Path	Deserialization Factor	Run-Length Violation Detector Range	
		Minimum	Maximum
Single-width	8	4	128
	10	5	160
Double-width	16	8	512
	20	10	640

Receiver Bit Reversal

By default, the Stratix II GX receiver assumes an LSBit to MSBit transmission. If the transmission order is MSBit to LSBit, then the receiver will put out the bit-flipped version of the data on the PLD interface. The Receiver Bit Reversal feature is available to correct this situation.

The Receiver Bit Reversal feature is available only in Basic single-width and Basic double-width modes. If the Receiver Bit Reversal feature is enabled in Basic single-width mode, the 10-bit data $D[9:0]$ at the output of the word aligner gets rewired to $D[0:9]$. If the Receiver Bit Reversal feature is enabled in Basic double-width mode without the 8B/10B decoder, the MSByte $D[15:8]$ and LSByte $D[7:0]$ at the output of the word aligner get rewired to $D[8:15]$ and $D[0:7]$, respectively. If the Receiver Bit Reversal feature is enabled in Basic double-width mode with the 8B/10B decoder, the MSByte $D[19:0]$ and LSByte $D[9:0]$ at the output of the word aligner get rewired to $D[0:19]$ and $D[0:9]$, respectively. Flipping the parallel data using this feature allows the receiver to put out the correctly bit-ordered data on the PLD interface in case of MSBit to LSBit transmission.

Since the receiver bit reversal is done at the output of the word aligner, a dynamic bit reversal would also require a reversal of word alignment pattern. As a result, the Receiver Bit Reversal feature is dynamic only if the receiver is dynamically reconfigurable (allows changing the word alignment pattern dynamically) or uses manual bit slip alignment mode (no word alignment pattern). The Receiver Bit Reversal feature is static in all other Basic mode configurations and can be enabled through the MegaWizard Plug-In. In configurations where this feature is dynamic, an `rx_revbitordwa` port is available to control the bit reversal dynamically. A high on the `rx_revbitordwa` port reverses the bit order at the input of the word aligner.

Figure 2–69 illustrates the receiver bit reversal feature in Basic single-width 10-bit wide data path configuration.

Figure 2–69. Receiver Bit Reversal in Single-Width Mode

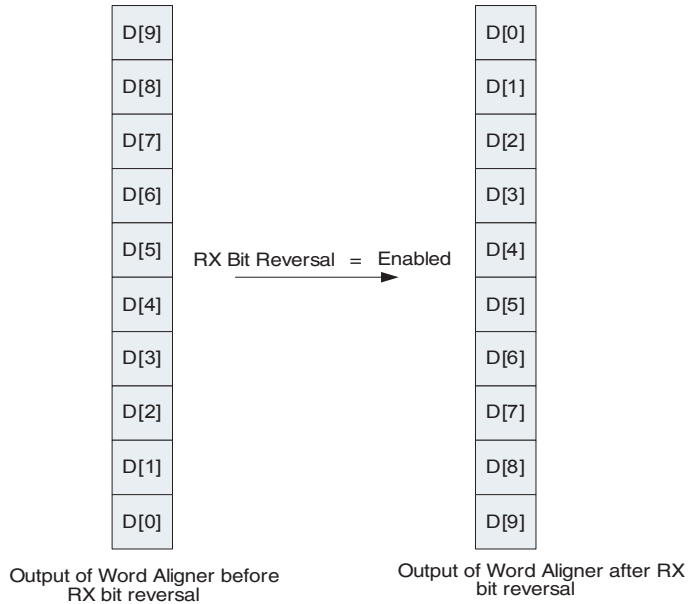
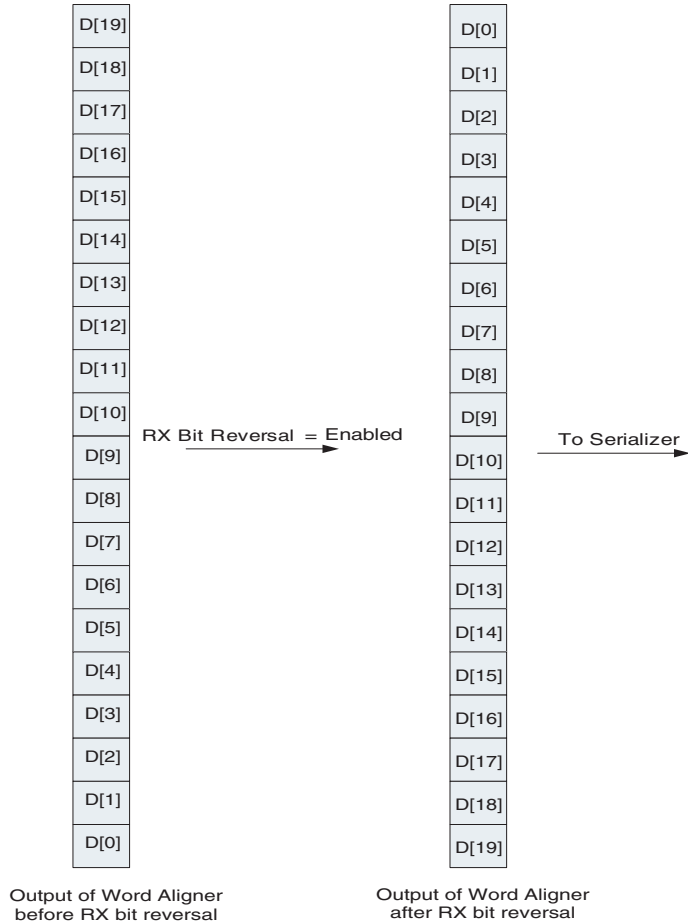


Figure 2–70 illustrates the receiver bit reversal feature in Basic double-width 20-bit wide data path configuration.

Figure 2–70. Receiver Bit Reversal in Double-Width Mode



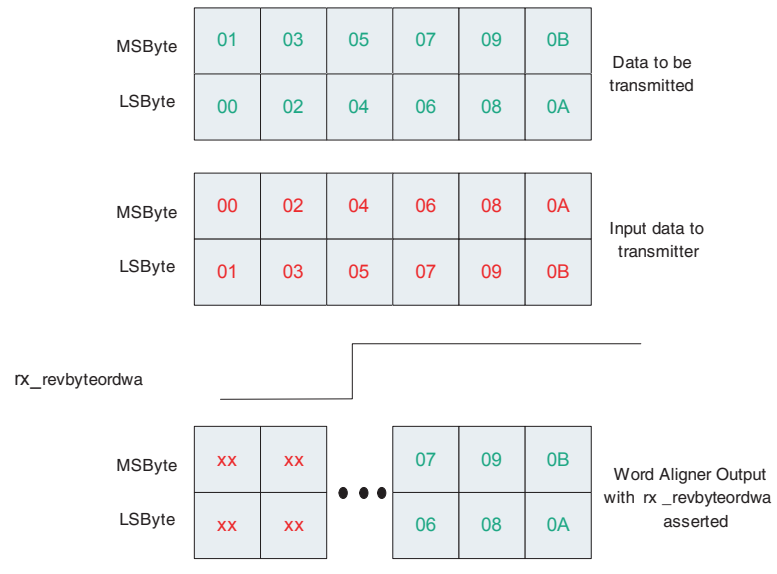
Receiver Byte Reversal

The MSByte and LSByte of the input data to the transmitter are often erroneously swapped. The receiver byte reversal feature is available to correct this situation.

An optional port `rx_revbyteordwa` is available only in Basic double-width mode to enable receiver byte reversal. In 8B/10B enabled mode, a high value on `rx_revbyteordwa` swaps the 10-bit MSByte and LSByte of the 20-bit word at the output of the word aligner in the receiver data path. In non 8B/10B enabled mode, a high value on `rx_revbyteordwa` swaps the 8-bit MSByte and LSByte of the 16-bit word at the output of the word aligner in the receiver data path. This compensates for the erroneous swapping at the transmitter and hence corrects the data received by the downstream systems. The `rx_revbyteorderwa` is a dynamic signal and may cause an initial disparity error at the receiver of an 8B/10B encoded link. The downstream system must be able to tolerate this disparity error.

Figure 2–71 illustrates the receiver byte reversal feature.

Figure 2–71. Receiver Byte Reversal Feature



Channel Aligner (Deskew)

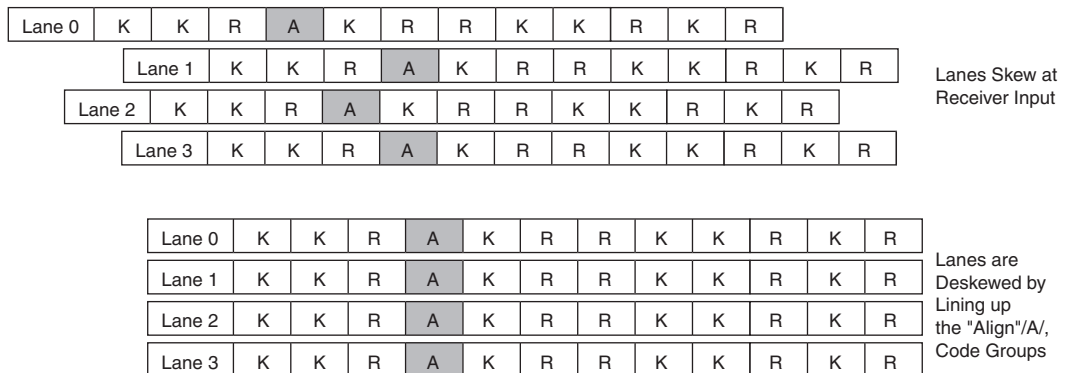
The channel aligner is automatically used when implementing the XAUI protocol to ensure that the channels are aligned with respect to each other. The channel aligner uses a 16-word-deep FIFO buffer.



The channel aligner is only available in the XAUI mode.

It is possible for ordered sets to be misaligned with respect to one another because of board skew or differences between the independent clock recoveries per serial lane. Channel alignment, also referred to as deskew or channel bonding, realigns the ordered sets by using the alignment code group, referred to as /A/. The /A/ code group is transmitted simultaneously on all four lanes, constituting an ||A|| ordered set, during idles or inter-packet gaps (IPG). XAUI receivers use these code groups to resolve any lane-to-lane skew. Skew between the lanes can be up to 40 UI (12.8ns) as specified in the standard, which relaxes the board design constraints. Figure 2-72 shows lane skew at the receiver input and how the deskew circuitry uses the /A/ code group to deskew the channels.

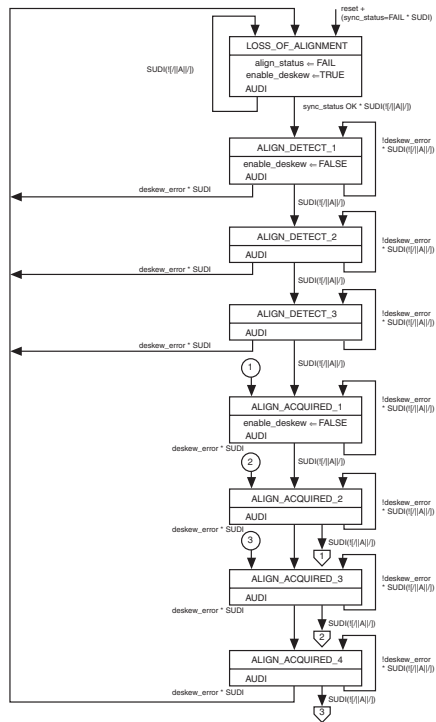
Figure 2-72. Lane Deskew With /A/ Code Group



Stratix II GX devices manage XAUI channel alignment with a dedicated deskew macro that consists of a 16-word-deep FIFO buffer that is controlled by a XAUI deskew state machine. The XAUI deskew state machine first looks for the /A/ code group within each channel. When the XAUI deskew state machine detects /A/ in each channel, the deskew FIFO buffer is enabled. The deskew state machine now monitors the reception of /A/ code groups. When four aligned /A/ code groups are received, the `rx_channelaligned` signal is asserted. The deskew state machine continues to monitor the reception of /A/ code groups and de-asserts the `rx_channelaligned` signal if alignment conditions are

lost. This built-in deskew macro is only enabled for the XAUI protocol. The PCS deskew state diagram specified in clause 48 of the IEEE P802.3ae is shown in [Figure 2–73](#).

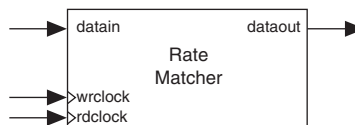
Figure 2–73. IEEE 802.3ae PCS Deskew State Diagram



Rate Matcher

The rate matcher ([Figure 2–74](#)) compensates for clock frequency differences between the upstream transmitter and the local receiver. The rate matcher operates in five modes: GIGE, XAUI, PIPE, Basic single-width mode, and Basic double-width mode.

Figure 2–74. Rate Matcher



The transceiver block can operate in multi-crystal environments, which can tolerate frequency variations of ± 300 PPM between crystals. Stratix II GX devices have embedded circuitry to perform clock rate compensation. Clock rate compensation is achieved by inserting or removing skip characters from the IPG or idle streams. This process is called rate matching or clock rate compensation.

The rate matcher in the transceiver consists of a 20-word-deep FIFO buffer and necessary logic to detect and perform the insertion and deletion functions.

XAUI

The rate matcher in XAUI mode operates in a synchronized $\times 4$ mode and supports up to a ± 100 PPM clock difference between the upstream transmitter and receiver. In this mode, the rate matcher can insert or delete a column of /R/ characters as denoted by the ||R|| designation, depending on whether the FIFO buffer is approaching an empty or full condition. The rate matcher does not operate until the XAUI synchronization state machine achieves word alignment and channel alignment. Until that point, the rate matcher is not active (read and write pointers do not move).

If the ||R|| code words are not received on all channels, rate matching does not occur and may lead to over/underflow conditions in the rate matching FIFO buffer. If this situation occurs, the data output of the receiver outputs a constant 9'h19C (8'h9C on the rx_dataout output and 1'b1 on the rx_ctrldetect output) in lane 0 (rest of the lane are data 8'h00). The receiver digital reset must be asserted and the lanes resynchronized before data can be received.

GIGE

The rate matcher in GIGE mode operates in a channel-by-channel mode and supports up to a ± 100 PPM clock difference between the upstream transmitter and the receiver. The rate matcher either inserts or deletes the /I2/ ordered set depending on whether the FIFO buffer is approaching an empty or full condition. The /I2/ order set consists of a /K28.5+/ code group and a /D16.2-/ code group (the sign after the code group signifies the running disparity at the end of the code group). The rate matcher in GIGE mode waits until the GIGE synchronization state machine achieves synchronization. Once synchronization is achieved, the rate matcher is active.

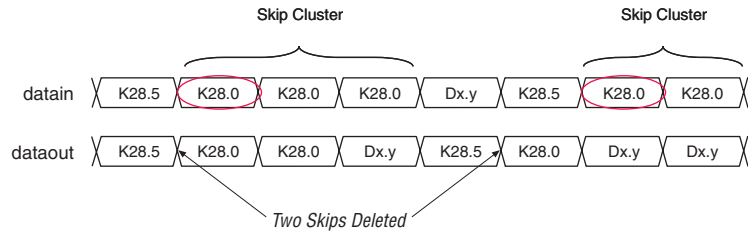
In the event the rate matching FIFO buffer in the GIGE mode approaches overflow or underflow, the transceiver outputs a sequence code group (9'h19C)—8'h9C on rx_dataout and 1'b1 on rx_ctrlrdetect. The rx_digitalreset signal must be asserted to reset the rate matcher FIFO buffer.

PIPE Mode

In PIPE mode, the rate matcher supports up to ± 300 PPM (600 PPM total) differences between the upstream transmitter and the receiver. The rate matcher looks for the skip ordered set, which is usually a /K28.5/ comma followed by three /K28.0/ skip characters. The rate matcher deletes or inserts skip characters when necessary to prevent the rate-matching FIFO buffer from overflowing or underflowing.

The rate matcher can delete only one skip character in a consecutive cluster of skip characters in PIPE mode only. Figure 2–69 shows a PIPE mode rate matcher deletion of two skip characters.

Figure 2–75. PIPE Mode With Two Deletions (One Per Cluster)



The rate matcher can perform skip character insertion one insertion per skip cluster in PIPE mode. There is no limit on the consecutive number of skip characters allowed per skip cluster.

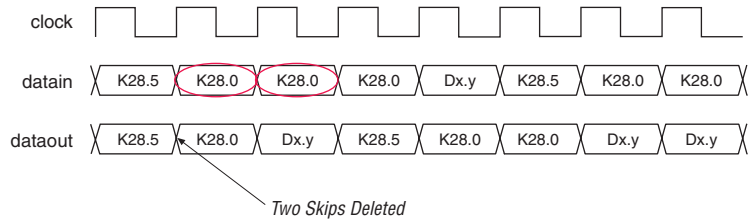
The Stratix II GX rate matcher in PIPE mode has FIFO buffer overflow and underflow protection. In the event of a FIFO buffer overflow, the rate matcher deletes any data after the overflow condition to prevent FIFO pointer corruption until the rate matcher is not full. In an underflow condition, the rate matcher inserts 9'h1FE (/K30.7/) until the FIFO buffer is not empty. These measures ensure that the FIFO buffer gracefully exits the overflow and underflow condition without requiring a FIFO buffer reset.

Single-Width General Rate Matching

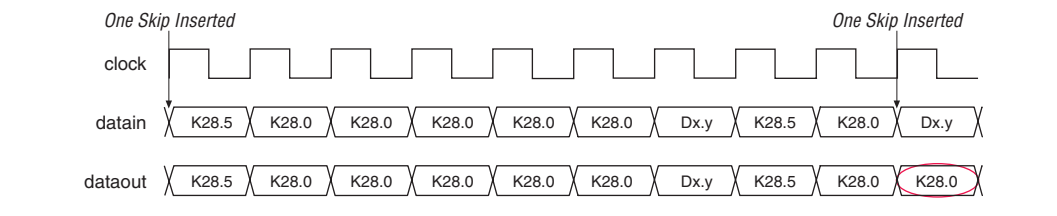
In Basic single-width mode, the rate matcher supports up to ± 300 PPM differences between the upstream transmitter and the receiver. The rate matcher looks for the skip ordered set, which is a /K28.5/ comma followed by three programmable neutral disparity skip characters (for example, /K28.0/). For general rate matching, you can customize the SOS to support a variety of protocols, including custom protocols. The SOS must contain a valid control code group (Kx.y), followed by any neutral disparity skip code group (any Kx.y or Dx.y of neutral disparity, for example, K28.0). The rate matcher deletes or inserts skip characters when necessary to prevent the rate matching FIFO buffer from overflowing or underflowing.

The rate matcher in single-width mode can delete any number of skip characters as necessary in a cluster as long as there are skip characters to delete. There are no restrictions regarding deleting more than one skip character in a cluster of skip characters. Figure 2-76 shows an example of a single-width mode rate matcher deletion of two skip characters. Although the skip characters are programmable, the /K28.0/ control group is used for illustration purposes.

Figure 2-76. Single-Width Mode Deletion of Two Skip Characters



The rate matcher inserts skip characters as required for rate matching. For a given skip ordered set, the rate matcher inserts skip characters so that the total number of consecutive skip characters does not exceed five at the output of the rate matching FIFO buffer. Figure 2-77 shows an example where a skip character insertion is made on the second set of skip ordered sets because the first set has the maximum number of skip characters.

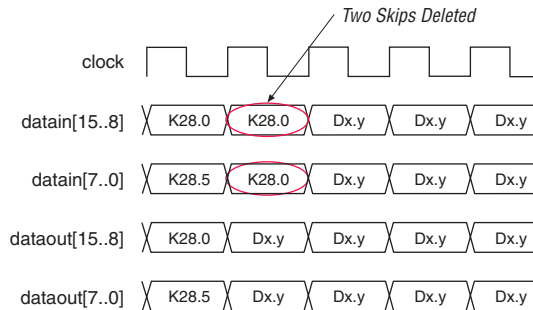
Figure 2–77. Single-Width Mode Insertion of a Skip Character

The Stratix II GX rate matcher in single-width mode has FIFO buffer overflow and underflow protection. In the event of a FIFO buffer overflow the rate matcher deletes any data after the overflow condition to prevent FIFO buffer pointer corruption until the rate matcher is not full. In an underflow condition, the rate matcher inserts 9'h1FE (/K30.7) until the FIFO buffer is not empty. These measures ensure that the FIFO buffer gracefully exits the overflow and underflow condition without requiring a FIFO buffer reset.

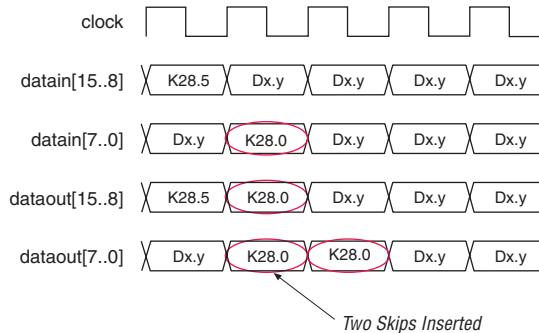
Double-Width General Rate Matching

In double-width mode, the rate matcher can support up to ± 300 PPM differences between the upstream transmitter and the receiver. The rate matcher looks for the skip ordered set, which is usually a /K28.5/ comma followed by programmable neutral disparity skip characters (for example, /K28.0/). For general rate matching, you can customize the SOS to support a variety of protocols, including custom protocols. The SOS must contain a valid control code group (Kx.y), followed by any neutral disparity skip code group (any Kx.y or Dx.y of neutral disparity, for example, K28.0). The rate matcher deletes or inserts dual skip characters when necessary to prevent the rate matching FIFO buffer from overflowing or underflowing.

The rate matcher deletes skip characters by pairs when they appear on the upper and lower bytes at the same time. There are no other restrictions for the deletion of skip characters. [Figure 2–78](#) shows an example of deleting two skip characters.

Figure 2–78. Deletion of Skip Characters in Double-Width Mode

The rate matcher inserts skip characters by pairs on the upper and lower byte (Figure 2–79). The insertion occurs after a /K28.5/ or the programmed control code group is detected by the rate matcher. If the comma is detected on the lower byte, the high byte must be a skip character. The insertion happens on the next double byte, if needed. If the comma appears on the upper byte, and the skip character is in the next lower byte, the insertion occurs on the double byte after the comma character, if needed.

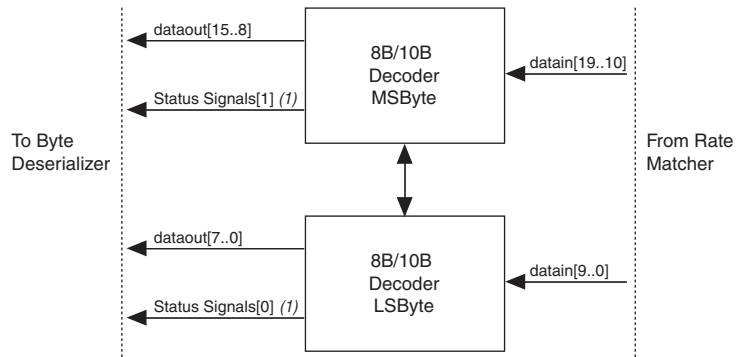
Figure 2–79. Insertion of Skip Characters in Double-Width Mode

8B/10B Decoder

The 8B/10B decoder (Figure 2–80) is part of the Stratix II GX transceiver digital blocks and lies in the receiver path between the rate matcher and the byte deserializer blocks. The 8B/10B decoder operates in two modes: single-width and double-width modes and can be bypassed if 8B/10B decoding is not needed. In single-width mode, the 8B/10B decoder

restores the 8-bit data + 1-bit control identifier from the 10-bit code. In double-width mode, there are two 8B/10B decoders cascaded together, which restores the 16-bit (2× 8-bit) data + 2-bit (2× 1-bit) control identifier from the 20-bit (2× 10-bit) code. This 8B/10B decoder conforms to the IEEE 802.3 1998 edition standards.

Figure 2–80. 8B/10B Decoder



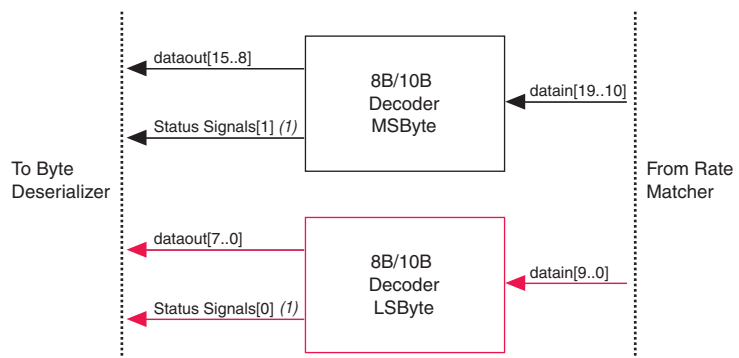
Note to Figure 2–80:

(1) Status signals include `rx_ctrldetect`, `rx_disperr`, and `rxerrdetect`.

Single-Width Mode

In single-width mode, the Stratix II GX 8B/10B decoder operates in a similar fashion as the Stratix GX 8B/10B decoder. The highlighted data path in Figure 2–81 is active in the single-width mode.

Figure 2–81. Active Data Path in Single-Width Mode



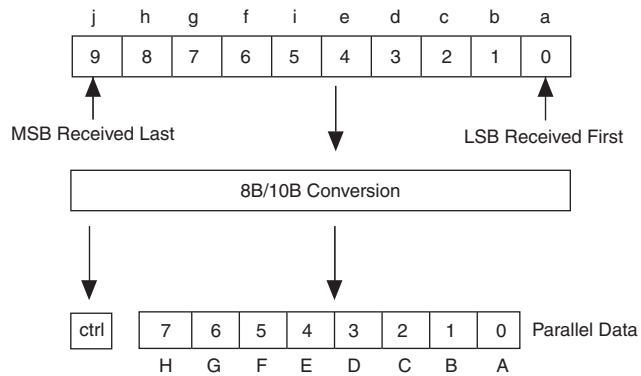
Note to Figure 2–81:

(1) Status signals include `rx_ctrldetect`, `rx_disperr`, and `rxerrdetect`.

10-Bit Decoding

The 8B/10B decoder in single-width mode translates the 10-bit encoded code into the 8-bit equivalent data or control code. The 10-bit code received must be from the supported Dx.y or Kx.y list with the proper disparity or error flag asserted. All 8B/10B control signals (disparity error, control detect, code error, and so on) are pipelined and edge-aligned with the data. Figure 2–82 shows how the 10-bit symbol is decoded to the 8-bit data, plus a 1-bit control indicator.

Figure 2–82. 10-Bit to 8-Bit Conversion



Code Error Detect

The `rx_errdetect` signal indicates when the code received contains an error. This port is optional but, if not in use, there is no way to detect if a code received is valid or not. The `rx_errdetect` signal goes high if a code received is an invalid code or if it has a disparity error. If a code is received that is not part of the valid Dx.y or Kx.y list, the `rx_errdetect` signal goes high. This signal is aligned to the invalid code word received at the PLD logic array.

In GIGE, XAUI, and PIPE mode, the invalid code is replaced by a /K30.7/ code (8'hFE on `rx_dataout` + 1'b1 on `rx_ctrldetect`). In all other modes, the value of the invalid code value can vary and should be ignored.

Disparity Error Detector


The 8B/10B decoder detects disparity errors based on which 10-bit code it received. The disparity error is indicated at the optional `rx_disper` port.



Refer to the *Specifications & Additional Information* chapter in volume 2 of the *Stratix II GX Device Handbook* for information on the disparity calculation.

If negative disparity is calculated for the last 10-bit code, a neutral or positive disparity 10-bit code is expected. If the 8B/10B decoder does not receive a neutral or positive disparity 10-bit code, the `rx_disperr` signal goes high, indicating that the code received had a disparity error.

If a positive disparity is calculated, a neutral or negative disparity 10-bit code is expected. The `rx_disperr` signal goes high if the code received is not as expected.

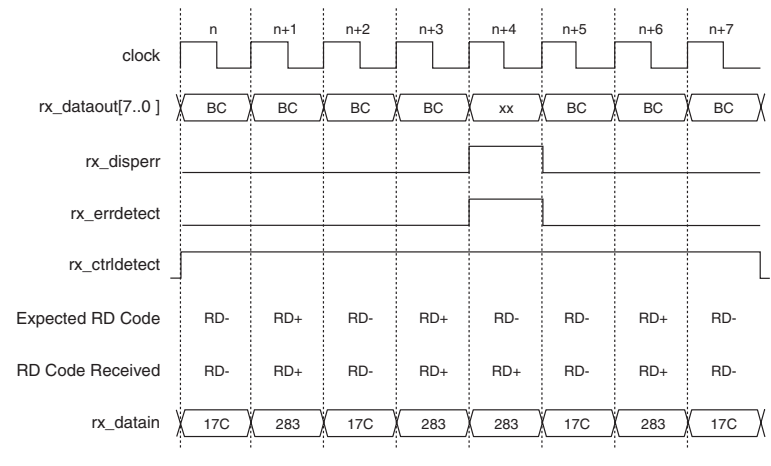
 When `rx_disperr` is high, `rx_errdetect` also goes high.

The detection of the disparity error might be delayed, depending on the data that follows the actual disparity error. The 8B/10B control codes terminate propagation of the disparity error. Any disparity errors propagated stop at the control code, terminating that disparity error.

In GIGE and XAUI modes, the code that contains a disparity error is replaced by a /K30.7/ code (8'hFE on `rx_dataout + rx_ctrldetect`). In all other modes, the code with incorrect disparity should be treated as an invalid code and ignored.

Figure 2–83 shows a case where the disparity is violated. A K28.5 code has an 8-bit value of 8'hbc and a 10-bit value that depends on the disparity calculation at the point of the generation of the K28.5 code. The 10-bit value is 10'b0011111010 (10'h17c) for RD- or 10'b1100000101 (10'h283) for RD+. If the running disparity at time $n - 1$ is negative, the expected code at time n must be from the RD- column. A K28.5 does not have a balanced 10-bit code (equal number of 1s and 0s), so the expected RD code must toggle back and forth between RD- and RD+. At time $n + 3$, the 8B/10B decoder received a RD+ K28.5 code (10'h283), which makes the current running disparity negative. At time $n + 4$, because the current disparity is negative, a K28.5 from the RD- column is expected, but a K28.5 code from the RD+ is received instead. This prompts `rx_disperr` to go high during time $n + 4$ to indicate that this particular K28.5 code had a disparity error. The current running disparity at the end of time $n + 4$ is negative because a K28.5 from the RD+ column was received. Based on the current running disparity at the end of time $n + 5$, a positive disparity K28.5 code (from the RD-) column is expected at time $n + 5$.

Figure 2–83. Disparity Error

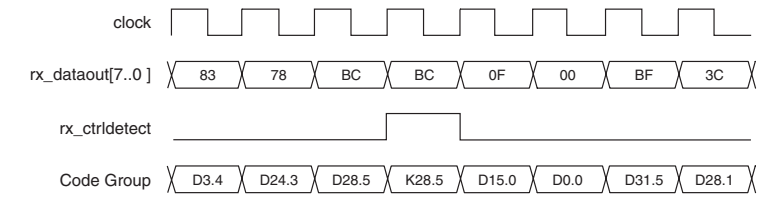


Control Detect

The 8B/10B decoder differentiates between data and control codes through the rx_ctrldetect port. This port is optional but, if not in use, there is no way to differentiate a Dx.y from a Kx.y.

Figure 2–84 shows an example waveform demonstrating the receipt of a K28.5 code (BC + ctrl). The rx_ctrldetect=1'b1 is aligned with 8'hbc, indicating that it is a control code. The rest of the codes received are Dx.y code groups.

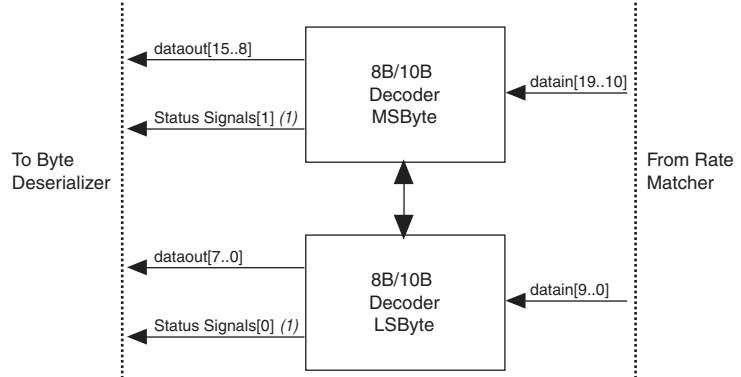
Figure 2–84. Control Code Detection



Double-Width Mode

In double-width mode, the dual 8B/10B decoder operates in cascaded fashion. The LSByte is received first, followed by the MSByte. The highlighted data path in [Figure 2–85](#) is active in the double-width mode.

Figure 2–85. Active Data Path in Double-Width Mode



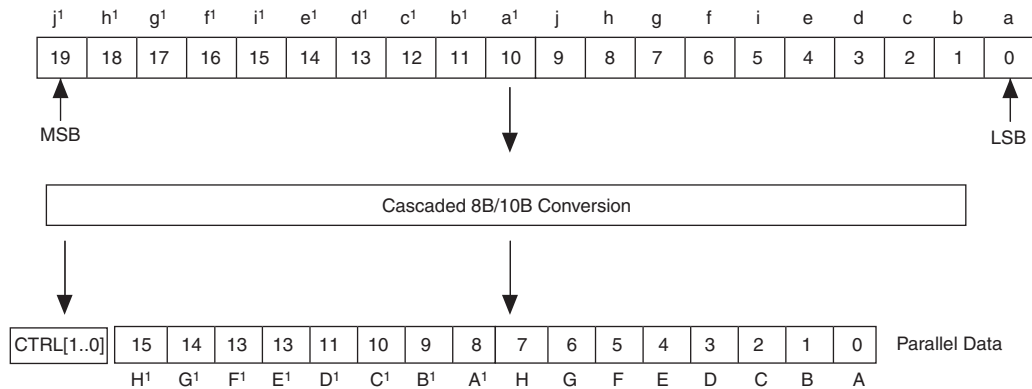
Note to [Figure 2–85](#):

(1) Status signals include `rx_ctrlldetect`, `rx_disperr`, and `rxerrdetect`.

20-Bit Decoding

The 8B/10B decoder in double-width mode translates the 20-bit (2×10 -bits) encoded code into the 16-bit (2×8 -bits) equivalent data or control code. The 20-bit upper and lower symbols received must be from the supported $Dx.y$ or $Kx.y$ list with the proper disparity or error flags asserted. All 8B/10B control signals (disparity error, control detect, and code error) are pipelined with the data in the Stratix II GX receiver block and are edge-aligned with the data. [Figure 2–86](#) shows how the 20-bit code is decoded to the 16-bit data plus a 2-bit control indicator.

Figure 2–86. 20-Bit to 16-Bit Conversion



Code Error detect

The `rx_errdetect` signal indicates when a code received contains an error. This port is optional but, if not in use, there is no way to detect if the code received is valid or not. The `rx_errdetect` signal goes high if a code received is an invalid code or if it has a disparity error. If a code is received that is not part of the valid $Dx.y$ or $Kx.y$ list, the `rx_errdetect` signal goes high. This signal is aligned to the invalid code word received at the PLD logic array.

In double-width mode, the `rx_errdetect` signal is 2-bits wide in the PCS portion of the transceiver. The lower bit indicates if the LSByte contains a code error, the upper bit indicates if the MSByte contains a code error. The value of the invalid code can vary and should be ignored.

Disparity Error Detector

The 8B/10B decoder in double-width mode forwards the current running disparity value from the LSByte decoder to the MSByte decoder to check the disparity of the symbol going into the MSByte decoder. The MSByte decoder's ending running disparity is then fed back to the LSByte decoder on the next clock cycle.



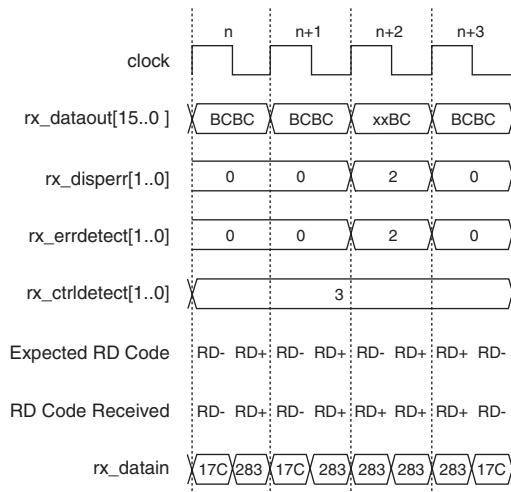
Refer to the *Specifications & Additional Information* chapter in volume 2 of the *Stratix II GX Device Handbook* for information on the disparity calculation.

The `rx_disperr` port is 2-bits wide in the PCS. When high, the lower bit indicates if the LSByte decoder detected a disparity error in the low byte code conversion. When high, the upper bit indicates if the MSByte decoder detected a disparity error in the high byte code conversion. If both of the `rx_disperr` bits are low, there is no error.

The detection of the disparity error might be delayed, depending on the data that follows the actual disparity error. The 8B/10B control codes terminate any propagation of the disparity error. Any disparity errors propagated assert `rx_disperr` on the control code byte, terminating that disparity error.

Figure 2–87 shows a case where the disparity is violated. A K28.5 code has an 8-bit value of 8'hbc and a 10-bit value that depends on the disparity calculation at the point of the generation of the K28.5 code. The 10-bit value is 10'b0011111010 (10'h17c) for RD- or 10'b1100000101 (10'h283) for RD+. This example uses double-width mode and the 20-bit codes are split into two 10-bit codes for clarity. The expected running disparity is indicated for each 10-bit code. At time n , `rx_datain` receives 10'h283 first and the decoded version goes on the LSByte of `rx_dataout`. At time $n + 2$, the high byte received a K28.5 code of incorrect disparity. The upper bits of the `rx_disperr` and `rx_errdetect` ports are asserted, resulting in the 2'h2 values shown in Figure 2–87.

Figure 2–87. Disparity Error



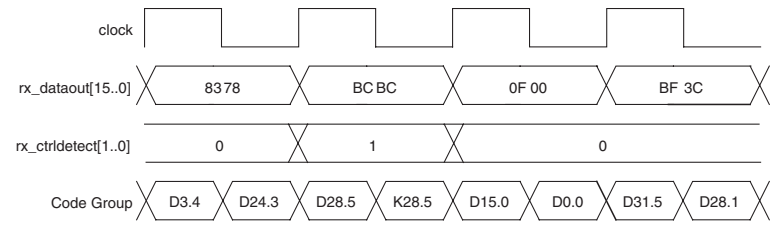
Control Detect

The 8B/10B decoder indicates to the PLD logic array the difference between data and control codes through the `rx_ctrldetect` port. This port is optional but, if not in use, there is no way to differentiate a `Dx.y` from a `Kx.y`.

In double-width mode, the `rx_ctrldetect` port is 2-bits wide inside the PCS. The lower bit indicates if the LSByte is a control word or data, and the upper bit indicates if the MSByte is a control word or data. When a control word is decoded, the corresponding `rx_ctrldetect` bit goes high. For data, the corresponding `rx_ctrldetect` bit goes low. The `rx_ctrldetect` port in the PLD logic array is edge-aligned with the code group it is associated with.

Figure 2–88 shows an example waveform that shows the receipt of a `K28.5` code (`BC + ctrl`). The `rx_ctrldetect=2'b1` is aligned with `8'hbc` on the LSByte, indicating that it is a control code. The `8'hbc` on the MSByte is a data, not a control word. The rest of the codes received are `Dk.y` control codes.

Figure 2–88. Control Code Detection



Reset

The reset for the 8B/10B decoder block is derived from the receiver digital reset (`rx_digitalreset`). When `rx_digitalreset` is asserted, the 8B/10B decoder block resets. In reset, the disparity registers are cleared and the outputs of the 8B/10B decoder block are driven low. After reset, the 8B/10B decoder starts with either a positive or negative disparity, depending on the disparity of the data it receives. The decoder calculates the initial running disparity based on the first valid code received.



The receiver block must be word aligned after reset before the 8B/10B decoder can decode valid data or control codes. If word alignment has not been achieved, the data from the 8B/10B decoder is discarded and considered invalid.

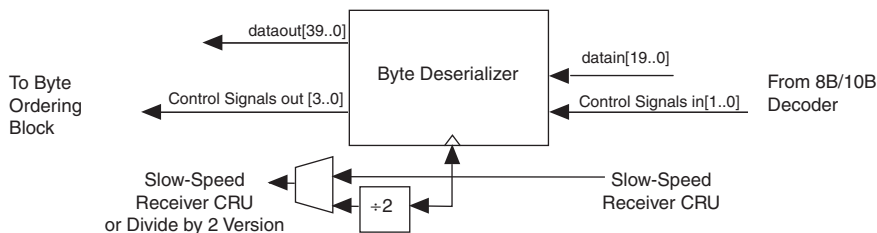
Polarity Inversion

The 8B/10B decoder has a PCI Express compatible polarity inversion on the data bus prior to 8B/10B decoding. This polarity inversion inverts the bits of the incoming data stream prior to the 8B/10B decoding block to fix potential P-N polarity inversion on the differential input buffer. You use the optional `pipe8b10binvpolarity` port to invert the inputs to the 8B/10B decoder dynamically from the PLD.

Byte Deserializer

Use the byte deserializer (Figure 2–89) to convert the one- or two-byte interface into a two- or four-byte-wide data path from the transceiver to the PLD logic (refer to Table 2–22). The PLD interface has a limit of 250 MHz, so the byte deserializer is needed to widen the bus width at the PLD interface and reduce the interface speed. For example, at 6.375 Gbps, the transceiver logic has a double-byte-wide data path that runs at 318.75 MHz in a $\times 20$ deserializer factor, which is above the maximum PLD interface speed.

Figure 2–89. Byte Deserializer



When using the byte deserializer, the PLD interface width doubles to 40-bits (36-bits when using the 8B/10B encoder) and the interface speed drops to 159.375 MHz.

Table 2–22. Byte Deserializer Input and Output Widths

Input Data Width (Bits)	Deserialized Output Data Width to the FPGA Logic Array (Bits)
20	40
16	32
10	20
8	16

If you use the byte deserializer, the byte ordering might be different than what you intended. [Figure 2–90](#) shows the byte deserializer operating in single-width mode. The expected data pattern is A at the lower byte, followed by B at the upper byte. C and D follow in the next lower and upper bytes, respectively.

Figure 2–90. Intended Transmitter Pattern

X	B	D
X	A	C

The receiver may receive the intended transmitter pattern or slip a byte, as shown in [Figure 2–91](#), where A arrives when the byte deserializer is stuffing the upper byte instead of stuffing the lower byte. This is a nondeterministic swap, because it depends on PLL lock times and link delay.

Figure 2–91. Incorrect Byte Position at Receiver After Byte Deserializer

A	C	X
X	B	D

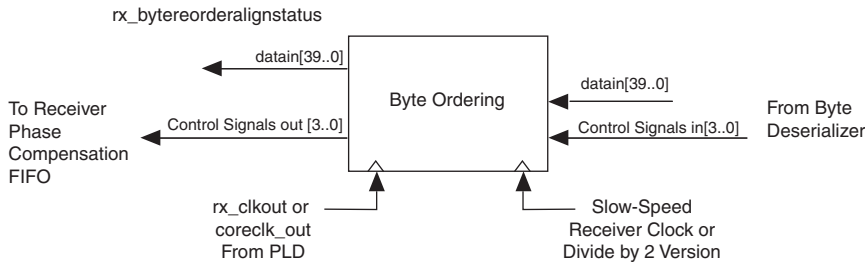
You can use the byte ordering block or a byte reordering circuit to restore the byte order to the expected pattern.

Byte Ordering

The Stratix II GX device has a dedicated byte ordering circuit on each receiver to obtain a certain byte order on multiple lanes. This circuit is used in conjunction with the byte deserializer block. The byte deserializer doubles the number of lanes for each receiver. If you use the single-width mode, the receiver output at the PLD interface 8-bits or 10-bits (single lane) if the byte deserializer is not used, and 16-bit or 20-bit (dual lanes) if it is used. If you use double-width mode, the receiver's output at the PLD interface is 16-bits or 20-bits (dual lanes) or 36-bits or 40-bits (transceiver block lanes) if the byte deserializer is used. The nature of the byte deserializer block does not lend itself to preserving the lane striping of the source transmitter. The least significant byte of the transmitter may be received in a different location. Refer to [“Byte Deserializer” on page 2–112](#) for more details on how the bytes can be re-ordered. The byte ordering block ensures that the correct lane striping is kept at the receiver output.

You cannot use the byte ordering block in conjunction with the rate matcher because it disturbs the byte ordering by adding or deleting bytes because of the data and clock PPM offset.

Figure 2–92. Byte Ordering Block



Word Alignment Based on Byte Ordering

In word alignment based on byte ordering, the byte ordering block performs lane alignment after the word aligner achieves byte alignment. The byte ordering block is triggered by the rising edge of the `rx_syncstatus` signal. To achieve lane alignment, the byte reordering block monitors the data stream for the alignment patterns. When the byte reordering block finds the correct alignment pattern, it inserts the programmable pad byte in the data stream until the alignment pattern can be placed in the LSByte position (lane 0). When the alignment pattern is placed in the LSByte position, the byte ordering process is complete and the status signal `rx_bytereorderalignstatus` asserts (stays high). If the alignment pattern is already in the LSByte position, the byte ordering block detects this, considers the byte ordering process complete, and asserts the `rx_bytereorderalignstatus` signal. Byte ordering is not performed again, even if the alignment byte exists in the data stream, until the channel is reset by the `rx_digitalreset` port (`rx_analogreset` and `gxb_powerdown` also reset the receiver channel).

Figure 2–93 shows how the byte ordering block works in a double-width mode four-lane configuration (four-byte-wide interface). The alignment character, denoted by the “A” character, goes into the byte ordering block in lane two. The byte ordering block inserts two pad bytes, denoted by PD, delaying the alignment byte until it appears in the LSByte position (lane 0).

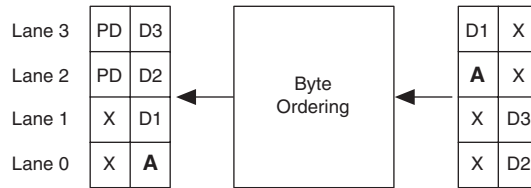
Figure 2–93. Double-Width Byte Ordering With Two Pad Byte Inserts

Figure 2–94 shows the byte ordering block in single-width mode two-lane configuration. In Figure 2–94, the alignment pattern “A” is in the MSByte position (lane 1). The byte ordering block inserts a pad character to force the alignment character to the LSByte position (lane 0). If the alignment pattern already exists in lane 0, the byte ordering process completes without any ordering done because it is unnecessary. After the ordering process is complete, the `rx_byteorderalignstatus` signal asserts and stays high until `rx_digitalreset`, `rx_analogreset`, or `gxb_powerdown` is asserted to reset the byte ordering block.

Figure 2–94. Single-Width Byte Ordering With One Pad Insert

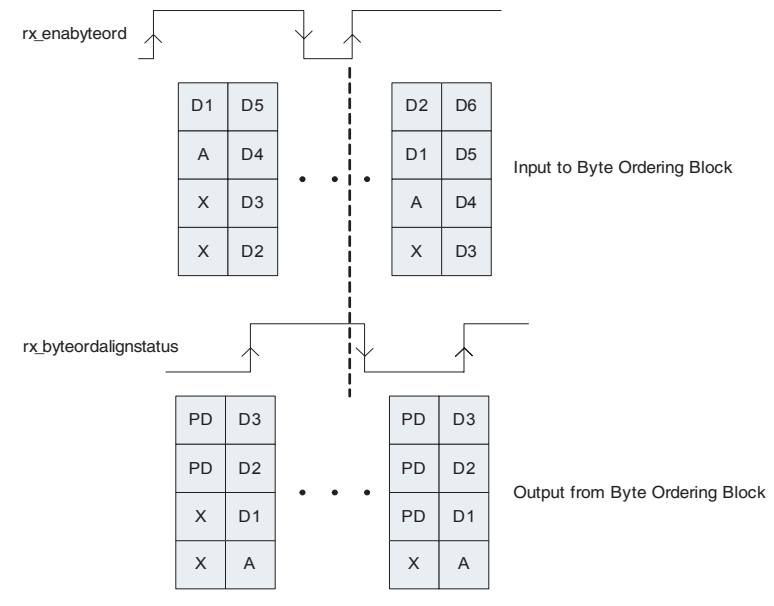
PLD-Controlled Byte Ordering

Unlike word alignment based byte ordering, PLD-controlled byte ordering provides control to the user logic to restore correct byte order at the receiver. When enabled, an `rx_enabyteord` port is available at the PLD interface. A rising edge on the `rx_enabyteord` port triggers the byte ordering block. The byte ordering block looks for the user-programmed byte ordering pattern in the data stream from the byte deserializer. When the byte reordering block finds the byte ordering pattern, it inserts the user-programmed pad byte in the data stream until the byte ordering pattern can be placed in the LSByte position. When the byte ordering pattern is placed in the LSByte position, the byte ordering process is complete and the status signal `rx_byteorderalignstatus` is asserted (stays high). If the alignment pattern is already in the LSByte position, the byte ordering block does not add any pad byte, considers the byte ordering process complete, and asserts the `rx_byteorderalignstatus` signal.

Unlike word alignment based byte ordering, PLD-controlled byte ordering does not require resetting the channel to re-trigger the byte ordering process. A rising edge on the `rx_enabyteord` signal re-triggers the process by de-asserting the `rx_byteorderalignstatus` signal. The byte ordering block starts looking for the byte ordering pattern again and adds pad bytes as necessary to achieve byte ordering. Once it completes the byte ordering process, it asserts the `rx_byteorderalignstatus` signal.

Figure 2–95 shows PLD-controlled byte ordering in Basic double-width Mode.

Figure 2–95. User-Controlled Byte Ordering in Double-Width Mode



After the first rising edge of the `rx_enabyteord` signal in Figure 2–95, the byte ordering block finds the byte ordering pattern A in the second most significant byte. It adds two pad bytes PD to push the byte ordering pattern to the least significant byte position and asserts the `rx_byteorderalignstatus` signal. After the second rising edge of the `rx_enabyteord` signal, `rx_byteorderalignstatus` is de-asserted and the byte ordering block starts looking for byte ordering pattern A. It finds the byte ordering pattern A in the second least significant byte position and adds three pad bytes PD. The byte ordering pattern A now appears at the least significant byte position and `rx_byteorderalignstatus` is asserted.

Two critical aspects related to PLD-controlled byte ordering process are:

- What to choose as the byte ordering pattern
- When to assert the `rx_enabyteord` signal

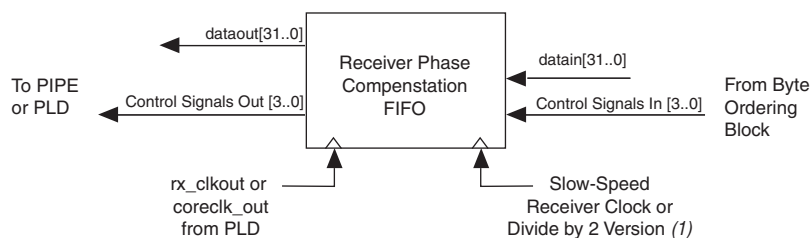
In Stratix II GX configurations, PLD-controlled byte ordering is available only in SONET/SDH OC-48 mode or Basic double-width mode. In SONET/SDH OC-48 mode, byte A2 of the A1A2 word alignment pattern is automatically selected as the byte ordering pattern. In Basic Double-Width mode, you program the byte ordering pattern while configuring the transceiver using the MegaWizard Plug-In Manager. Since the byte ordering block is designed to place the byte ordering pattern at the LSByte position, you must select a pattern that appears at the LSByte position at the source. This ensures that when the byte ordering block pushes the byte ordering pattern byte to the LSByte position at the receiver, the data is correctly byte ordered. Ideally, if this pattern is unique and is guaranteed to appear only at the LSByte position at the source, the instance at which the `rx_enabyteord` signal is asserted becomes irrelevant. For example, in packet-based 8B/10B encoded links, you could choose the Start of Packet (SOP) byte as the byte ordering pattern if it is a unique control code (say K28.0). In non 8B/10B scrambled data links, it may be difficult to find a unique pattern since there is a possibility of the pattern appearing in the scrambled payload and causing the byte ordering block to add pad bytes incorrectly. In such cases, the instance at which the `rx_enabyteord` signal is asserted becomes critical.

The `rx_enabyteord` signal must be asserted after the word aligner has aligned to the correct word boundary. This ensures that the byte ordering block does not find a byte ordering pattern between the word boundaries. If the `rx_enabyteord` signal is asserted before the intended byte ordering byte appears at the receiver, then the byte ordering block will add necessary pad bytes to achieve correct byte ordering. If the `rx_enabyteord` signal is asserted before the unintended data byte that matches the byte ordering pattern, then the byte ordering block may incorrectly add pad bytes and assert the `rx_byteorderalignstatus` signal. In the SONET/SDH OC-48 configuration, since the receiver anticipates the byte ordering pattern A2 every 125 μ s, the `rx_enabyteord` signal assertion can be easily timed to avoid incorrect byte ordering. In Basic Double-Width mode, it is up to you to either select a unique byte ordering pattern or an appropriate instance to assert `rx_enabyteord`, depending on the dynamics of the implemented protocol.

Receiver Phase Compensation FIFO Buffer

The receiver phase compensation FIFO buffer (Figure 2–96) is located at the FPGA logic array interface in the receiver block and is used to compensate for phase difference between the receiver clock and the clock from the PLD. The receiver phase compensation FIFO buffer operates in two modes: low latency and high latency. In low latency mode, the FIFO buffer is four words deep. The Quartus II software chooses the low latency mode automatically for every mode except the PCI-Express PIPE mode (which automatically uses high latency mode). In high latency mode, the FIFO buffer is eight words deep.


Figure 2–96. Receiver Phase Compensation FIFO Buffer



Note to Figure 2–96:

- (1) The receiver clock can either be the recovered clock or the transmitter CMU clock, depending on whether the rate matcher is used or not.

In Basic mode, the write port is clocked by the recovered clock from the CRU. This clock is half the rate if the byte deserializer is used. The read clock is clocked by the associated channel's recovered clock.

 The receiver phase compensation FIFO is always used and cannot be bypassed.

In four-channel (×4) and eight-channel (×8) bonding modes, all the read pointers are derived from a common source so that there is no need to synchronize the data of each channel in the PLD logic.

Receiver Phase Compensation FIFO Error Flag

Depending on the transceiver configuration, the write port of the receiver phase compensation FIFO can be clocked by either the recovered clock (`rx_clkout`) or transmitter PLL output clock (`tx_clkout` or `coreclkout`). The read port can be clocked by the recovered clock (`rx_clkout`), transmitter PLL output clock (`tx_clkout` or

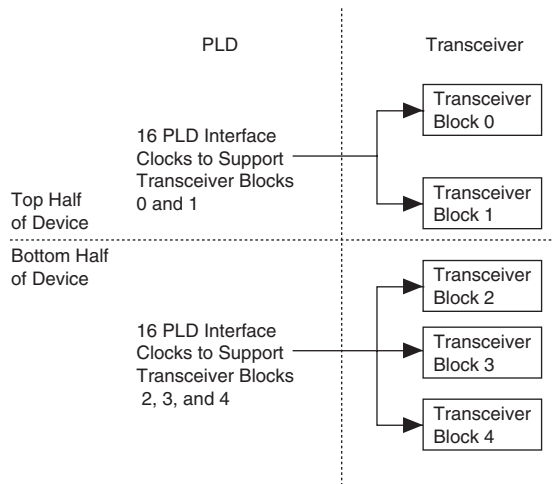
coreclkout) or a PLD clock. In all configurations, the write clock and the read clock must have 0 PPM difference to avoid overrun/underflow of the phase compensation FIFO.

An optional `debug_rx_phase_comp_fifo_error` port is available in all modes to indicate receiver phase compensation FIFO overrun/underflow condition. `debug_rx_phase_comp_fifo_error` is asserted high when the phase compensation FIFO gets either full or empty. This feature is useful to verify the phase compensation FIFO overrun/underflow condition as a probable cause of link errors.

PLD-Transceiver Interface Clocking

There are 32 PLD interface clocks available between the PLD logic and the transceiver blocks. The 32 PLD interface clocks are divided equally between the top and bottom half of the device (16 PLD interface clocks for the top half of the device and 16 PLD interface clocks for the lower half). The PLD interface clocks are used as the receiver and transmitter phase compensation FIFO clocks. [Figure 2-97](#) shows the PLD clock interface.

Figure 2-97. PLD Interface Clock (2SGX130G)



The following clock inputs utilize the PLD interface clocks:

- `rx_crucclk` (if driven from the PLD clock tree)
- `pll_inclk` (if driven from the PLD clock tree)
- `tx_coreclk`
- `rx_coreclk`
- `cal_blk_clk`

The `rx_cruclk` and the `pll_inclk` are reference clocks to the transceiver receiver PLL and transmitter PLL. These two ports can take the reference clock from the dedicated `REFCLK` pins or from the PLD global clock pins. If the PLD global clock pins are used to feed the transceiver PLLs, a PLD interface clock will be used for each independent reference clock feeding the transceiver.

Each transceiver block has one possible PLD connection to `pll_inclk` and four possible connections to `rx_cruclk`. Only one reference clock frequency can be fed from the PLD to each transceiver block. The receiver PLLs of each channel can possibly have a different reference clock frequency as long as there are PLD interface clocks available.

The `tx_coreclk` and `rx_coreclk` are input clocks to the transmitter and receiver phase compensation FIFOs, respectively. By default, the Quartus II software automatically routes the `tx_clkout` or `coreclk_out` to the `tx_coreclk`, and the `rx_clkout`, `tx_clkout`, or `coreclk_out` to the `rx_coreclk` port, depending on the transceiver block and channel configuration as listed in the above section.

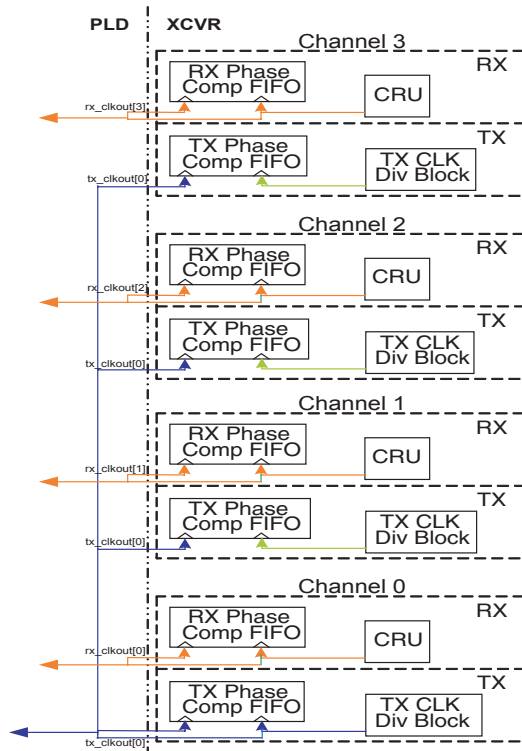
There are options to route other PLD clocks to the `tx_coreclk` and `rx_coreclk` ports. The non-transceiver clocks that feed these ports are required to be frequency locked (0 ppm) to the transceiver output clocks of the associated channel or transceiver block, depending on the configuration. The method of using this option is discussed in a later section.

The `cal_blk_clk` feeds the calibration block.

If a single clock from the PLD feeds multiple ports listed above, then only one PLD interface clock will be used. It is recommended that whenever possible, utilize a common clock. This will save PLD clock resources and PLD interface resources.

Each transceiver block (with all channels running in the same configuration), by default, uses a minimum of five PLD interface clocks as seen in [Figure 2-98](#). This is with each `rx_coreclk` clocked by the associated `rx_clkout` of each RX channel, and since all the TX channels are the same, the Quartus II software will automatically route `tx_clkout[0]` to all the `tx_coreclk` inputs. The reference clock can use the dedicated `REFCLK` pins to save on PLD interface clocks. The Quartus II software does not cross the transceiver block boundary when combining like TX channels. Also, the Quartus II software does not combine RX clocks automatically.

Figure 2–98. Minimum PLD Interface Clock Utilization

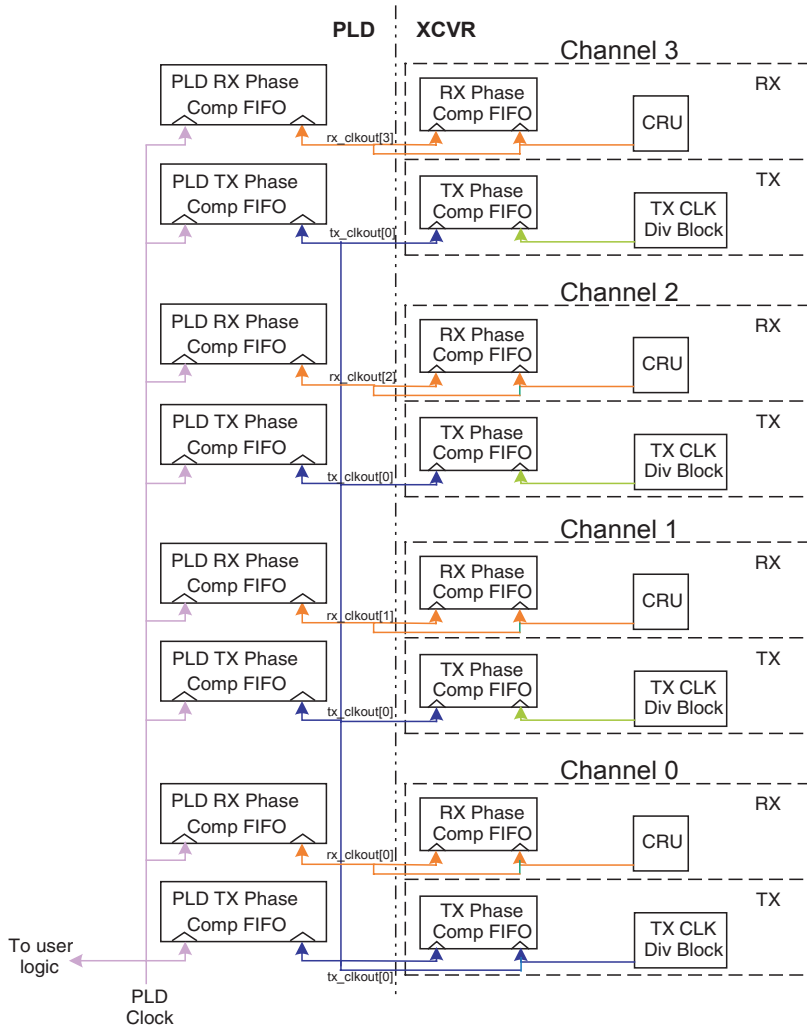


Note that in this configuration, the user logic needs to be clocked by the `tx_clkout [0]` for the TX path and the individual RX channel logic needs to be clocked by its associated `rx_clkout`.

If you use the default configuration, and the user logic is clocked by another clock than the transceiver clock associated with that channel, a PLD phase compensation must decouple the phase difference.

Figure 2–99 shows of TX and RX PLD phase compensation FIFOs decoupling the user logic from the transceiver.

Figure 2–99. Phase Compensation FIFO Implementation in PLD Logic



If more TX channels are used across transceiver blocks, and/or RX channels are also configured in a similar fashion and have the same data rate and PLD output clock frequency, you will need to manually connect the `tx_coreclk` and `rx_coreclk` ports.

The PLD interface clock utilization can be further reduced by directly feeding the `tx_coreclk` and/or the `rx_coreclk` ports of like channels directly with a single clock. The source clock must be frequency locked to the associated transceiver output clock. Any PPM difference results in data corruption.

To help guard against incorrect usage, the use of the `tx_coreclk` and `rx_coreclk` options requires clock assignments in the assignment organizer. If no assignments are used, the Quartus II software will give a compilation error.

There are 4 settings to enable the PLD interface clocking options:

- **Stratix II GX GXB Shared Clock Group Setting**
- **Stratix II GX GXB Shared Clock Group Driver Setting**
- **Stratix II GX 0PPM Clock Group Setting**
- **Stratix II GX 0PPM Clock Group Driver Setting**

As the name indicates, there are two main settings, each with a driver and clock group setting. When specifying clock groups, an integer identifier is used as the group name in order to differentiate other clock group settings from one another.

The **Stratix II GX GXB Shared Clock Group Setting** is the safest assignment. The Quartus II compiler will analyze the netlist during compilation to ensure TX channel members are derived from the same source. The Quartus II software will give a fitting error for incompatible assignments. The software cannot check for the output of the RX frequency locked to the driving clock as the exact frequency is dictated by the upstream transmitter's source clock. It will be up to you to ensure that the `rx_coreclk` is derived from the same source clock as the upstream transmitter.

The **Stratix II GX GXB Shared Clock Group Driver Setting** assignment must be made to the source channel of the `tx_clkout` or `coreclk_out`. Specifying anything except the TX channels (the source for the `tx_clkout` or `coreclk_out`) will result in a fitter error. If the source clock is not from `tx_clkout` or `coreclk_out` (for example, the source is from `rx_clkout` or from a PLD clock input), the 0 PPM setting must be used instead.

For example, in a synchronous system, the TX and RX are of the same data rate and configuration. The clock output of the channel 0 is used (but any TX clock output can be used). The **Stratix II GX GXB Shared Clock Group Driver Setting** is made in the assignment editor on the `tx_dataout [0]` name. You can use a group identifier value of "1" to identify the group that this driver feeds. The **Stratix II GX GXB shared**

Clock Group Setting is made to all the rx_datain channels that the tx_dataout [0] output clock drives. (note that the other tx_dataout channels do not need an assignment as the Quartus II software automatically groups the like transmitters in a transceiver block). A group identifier value of "1" is also made to the rx_datain assignments.

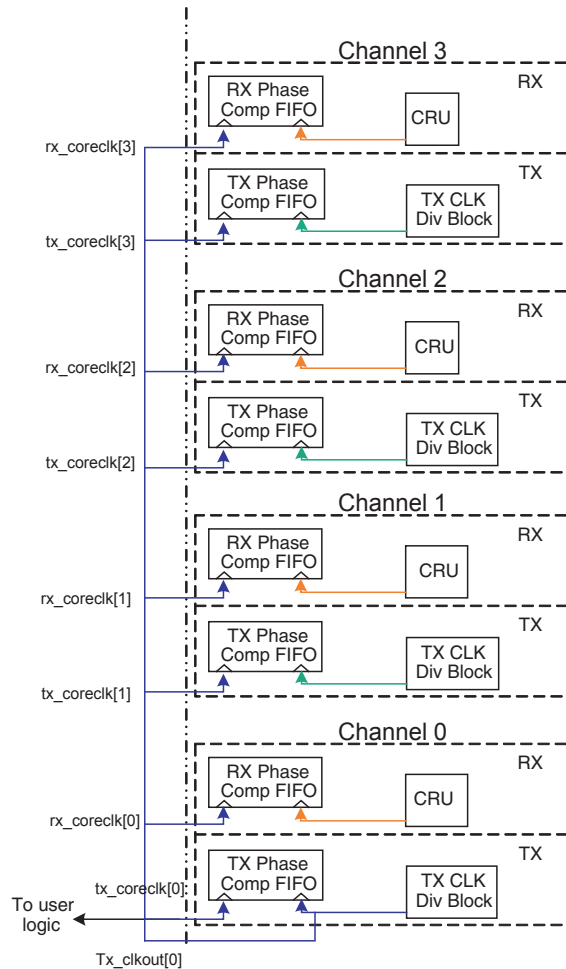
A breakdown of the assignment in the assignment editor is shown in [Table 2-23](#):

<i>Table 2-23. Assignment Editor</i>	
To:	tx_dataout [0]
Assignment name:	Stratix II GX GXB Shared Clock Group Driver Setting
Value:	1
To:	rx_datain [] (note that the [] signifies the entire rx_datain group)
Assignment name:	Stratix II GX GXB Shared Clock Group Setting
Value:	1

[Figure 2-100](#) shows the clocking configuration of the example.



For x4 transceiver block configurations, the coreclk_out can replace the tx_clkout [0] but the driver assignment still remains tx_dataout [0].

Figure 2-100. *tx_clkout[0]* Feeding All the *tx_coreclk* and *rx_coreclk* Posts of a Transceiver Block

The **Stratix II GX 0PPM Clock Group Setting** is for more advanced users that know the clocking configuration of the entire system and wants to reduce the PLD global clock resource and PLD interface clock resource utilization.

The Quartus II compiler does not perform any checking on the clock source. It is up to you to ensure that there is no frequency difference from the associated transceiver clock of the group and the driving clock to the *tx_coreclk* and *rx_coreclk* ports.

The **Stratix II GX 0PPM Clock Group Driver Setting** can be made to any of the transceiver output clocks (`tx_clkout`, `rx_clkout`, and `coreclk_out`) as well as any PLD clock input pins, transceiver dedicated REFCLK pin, or PLD PLL output. User logic cannot be used as a driver. As with the shared clock group setting, the driver setting for the transceiver output clocks is made to the associated channel. For example, for `tx_clkout` or `coreclk_out`, the transmitter channel name is specified. For the `rx_clkout` being the driver, the receiver channel name of the associated `rx_clkout` is specified. For the PLD input clock pins and the transceiver REFCLK pins, the name of the clock pin can be specified. For the PLL output, the PLL clock output port of the PLL can be found in the node finder and entered as the driver name. An integer value is specified for the group identification.

The **Stratix II GX 0 PPM Clock Group Setting** is made to the TX or RX channel names.

A breakdown of the assignment in the assignment editor is shown in [Table 2-24](#):

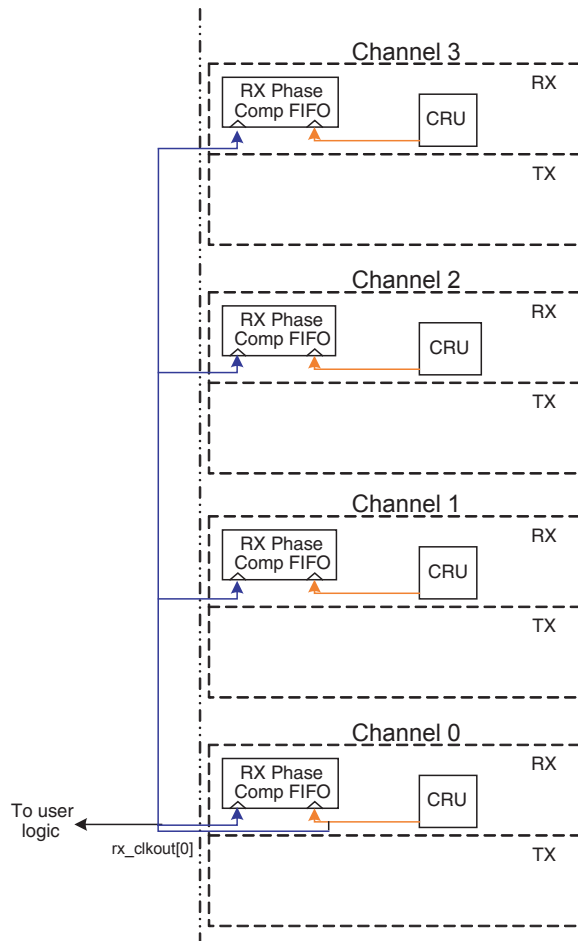
Table 2-24. Assignment Editor	
To:	<code>tx_dataout</code> , <code>rx_datain</code> , <code>pld_clk_pin_name</code> , <code>refclk_pin</code> , and <code>pll_outclk</code>
Assignment name:	Stratix II GX GXB 0 PPM Clock Group Driver Setting
Value:	1
To:	<code>rx_datain[]</code> and <code>tx_dataout[]</code>
Assignment name:	Stratix II GX GXB 0 PPM Clock Group Setting
Value:	1

The following are examples of clocking configurations that can use the 0PPM assignment:

All RX channels are configured the same and one recovered clock is feeding the `rx_coreclk` as shown in [Figure 2-101](#). Though this example shows all the RX channels reside in a transceiver block, `rx_clkout [0]` of this transceiver block can feed other RX channels of other transceiver blocks. Note that this example is not showing any TX channels. If the `rx_clkout` is used as a driver, it can only feed RX channels. If these channels are used in a duplex mode, the `tx_clkout` from the TX channels should be used as the driver and feed the TX as well as the RX phase compensation FIFOs. The `rx_clkout` cannot feed the `tx_coreclk` ports.

It is important to note that asserting the `rx_analogreset` of the RX channel associated with the driver clock will flatline the clock. All logic and RX Phase Compensation FIFOs read port that it feeds will not be receiving a clock during analog reset of the driving channel. If the reset state machine is clocked by the driver clock, the reset state machine will hang and may not come out of reset. All the RX channels will need to go through a digital reset in order to restore the phase compensation FIFO pointers.

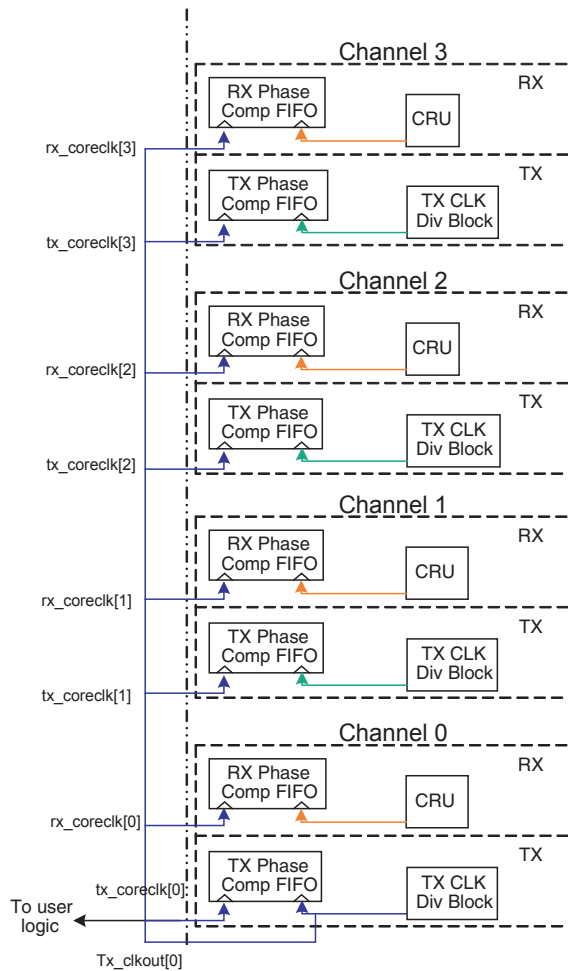
Figure 2–101. rx_clkout[0] Feeding All Receivers of the Same Transceiver Block



The next example is similar to the shared clock example of a `tx_clkout` feeding all the RX and TX channels, except that with the 0PPM setting, the `tx_clkout` can drive across transceiver blocks, as shown in [Figure 2-102](#). The upstream device feeding the RX channels must be frequency locked to the `tx_clkout` used.

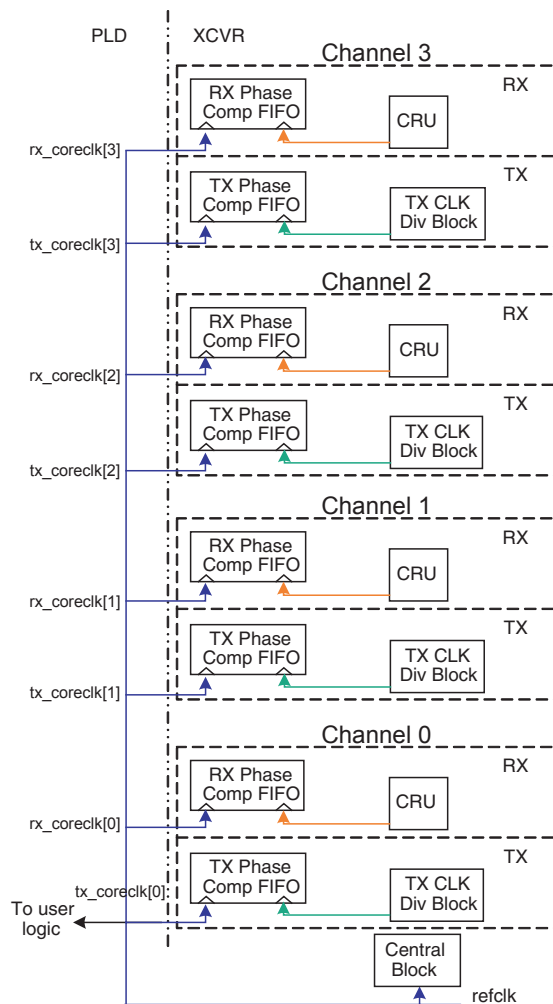
As with the RX channel example above, it is important to note that powering down the transceiver block where the driving channel resides will flatline the `tx_clkout`. All logic and the write ports of all the TX phase compensation FIFO will the driving clock feeds will flatline. A digital reset must be done on all channels after a driving transceiver block power down event.

Figure 2–102. tx_clockout[0] Feeding TX and RX Phase Compensation FIFOs Across Multiple Transceiver Blocks



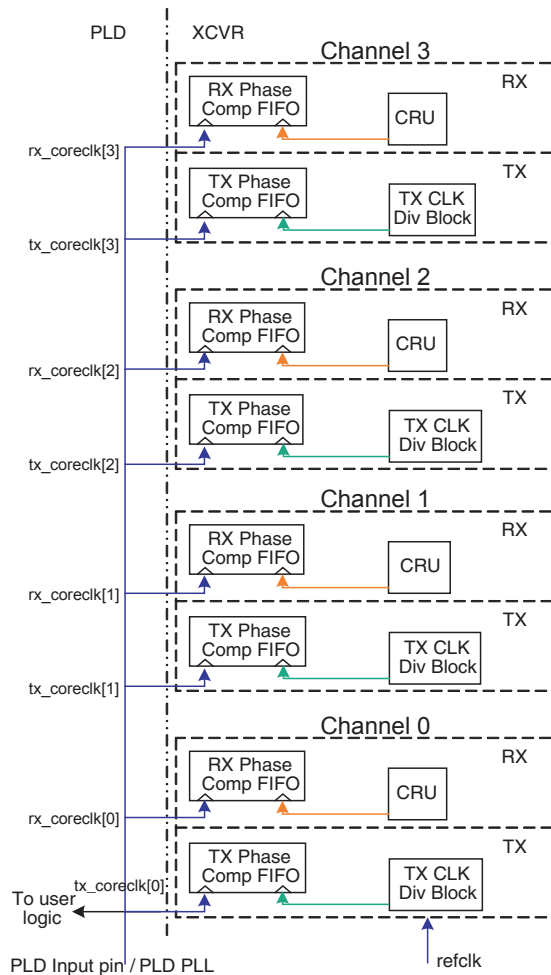
The next example features the dedicated REFCLK feeding the tx_coreclk and rx_coreclk ports as well as supplying the transceiver with the reference clock (Figure 2–103). This requires that the frequency of the reference clock at the REFCLK pin be of the same frequency as the transceiver output clocks of the associated channels. Any frequency difference yields corruption of data.

Figure 2–103. Dedicated REFCLK Feeding PCFIFO Clock Ports



Another example is where the PLD input clock or PLL output is feeding the tx_coreclk and rx_coreclk ports. Note that the driver clock must be the same frequency as the transceiver output clocks. Also, though this example shows the channels within a single transceiver block, the 0PPM setting will also allow TX and/or RX channel PCFIFOs of multiple transceiver blocks to be clocked by a common clock.

Figure 2-104. PLD-Driven Clock Feeding the Transceiver Phase Compensation FIFOs



PLD Interface Clock Resources

For the regional or global clock network to route into the transceiver, a local route input output (LRIO) channel is required. Each LRIO clock region has up to eight clock paths and each transceiver block has maximum of eight clock paths for connecting with LRIO clocks. These resources are limited and determine the number of clocks that can be used between the PLD and transceiver blocks. Tables 2-25 through 2-28 give the number of LRIO resources available for Stratix II GX devices with different numbers of transceiver blocks.

Table 2–25. Available Clocking Connections for Transceivers in 2SGX30D

	Clock Resource		Transceiver	
	Global Clock	Regional Clock	Bank13 8 Clock I/O	Bank14 8 Clock I/O
Region0 8 LRIO Clock	✓	RCLK 20-27	✓	
Region1 8 LRIO Clock	✓	RCLK 12-1		✓

Table 2–26. Available Clocking Connections for Transceivers in 2SGX60E

	Clock Resource		Transceiver		
	Global Clock	Regional Clock	Bank13 8 Clock I/O	Bank14 8 Clock I/O	Bank15 8 Clock I/O
Region0 8 LRIO Clock	✓	RCLK 20-27	✓		
Region1 8 LRIO Clock	✓	RCLK 20-27	✓	✓	
Region2 8 LRIO Clock	✓	RCLK 12-19		✓	✓
Region3 8 LRIO Clock	✓	RCLK 12-19			✓

Table 2–27. Available Clocking Connections for Transceivers in 2SGX90F (Part 1 of 2)

	Clock Resource		Transceiver			
	Global Clock	Regional Clock	Bank13 8 Clock I/O	Bank14 8 Clock I/O	Bank15 8 Clock I/O	Bank16 8 Clock I/O
Region0 8 LRIO Clock	✓	RCLK 20-27	✓			
Region1 8 LRIO Clock	✓	RCLK 20-27		✓		

Table 2–27. Available Clocking Connections for Transceivers in 2SGX90F (Part 2 of 2)

	Clock Resource		Transceiver			
	Global Clock	Regional Clock	Bank13 8 Clock I/O	Bank14 8 Clock I/O	Bank15 8 Clock I/O	Bank16 8 Clock I/O
Region2 8 LRIO Clock	✓	RCLK 12-19			✓	
Region3 8 LRIO Clock	✓	RCLK 12-19				✓

Table 2–28. Available Clocking Connections for Transceivers in 2SGX130G

	Clock Resource		Transceiver				
	Global Clock	Regional Clock	Bank13 8 Clock I/O	Bank14 8 Clock I/O	Bank15 8 Clock I/O	Bank16 8 Clock I/O	Bank17 8 Clock I/O
Region0 8 LRIO Clock	✓	RCLK 20-27	✓				
Region1 8 LRIO Clock	✓	RCLK 20-27					
Region2 8 LRIO Clock	✓	RCLK 12-19			✓	✓	
Region3 8 LRIO Clock	✓	RCLK 12-19				✓	✓

Clock Signal

provides information about clock signal on the transceiver block.

Table 2–29. Clock Port List (Part 1 of 2)

Clock Name	Maximum Number per Transceiver Block	Notes
p1l_inclk	2	(1)
rx_cruclk	4	rx_cruclk can share resources with p1l_inclk
tx_clkout / tx_coreclk	4	
rx_clkout / rx_coreclk	4	

Table 2–29. Clock Port List (Part 2 of 2)

Clock Name	Maximum Number per Transceiver Block	Notes
reconfig_clock	1	Used only for dynamic reconfiguration.
cal_blk_clk	1	One for all transceiver blocks.
fixedclk	1	PCI-Express (PIPE) mode only.
coreclkout	1	Used only for four-lane configurations.

Note to Table 2–29:

- (1) Altera recommends using the REFCLK pin for pll_inclk. The REFCLK pin uses inter-transceiver line; therefore, there is no need to use the LRIO clock resource. This usage helps with performance and resource.

Example of clock usage

In a five transceiver block device or using channel reconfiguration, it is possible to exceed the number of LRIO clocks available.

Table 2–30 shows an example of LRIO clock resource usage in a EP2SGX130G device. In this case, the Quartus II software does not give errors for transceiver configurations.

Table 2–30. Example of LRIO Clock Resource Usage (Part 1 of 2)

	Clock Name	Number of Signals in the Transceiver Block	LRIO Resource Usage	Routing
Bank13 XAUI	pll_inclk	1	0	Using one REFCLK pin.
	rx_cruclk	4	0	Connect to pll_inclk.
	coreclkout	1	1	To PLD fabric.
Region0 LRIO Clock			1/8	
Bank14 GIGE 4ch	pll_inclk	1	0	Using one REFCLK pin.
	rx_cruclk	4	0	Connect to pll_inclk.
	tx_clkout	4	1	To PLD fabric.
Region1 LRIO Clock			1/8	

Table 2–30. Example of LRIO Clock Resource Usage (Part 2 of 2)

	Clock Name	Number of Signals in the Transceiver Block	LRIO Resource Usage	Routing
Bank15 Basic 4ch	pll_inclk	1	0	Using one REFCLK pin.
	rx_cruclk	4	0	Connect to pll_inclk.
	rx_clkout	4	4	To PLD fabric.
	tx_clkout	4	1	To PLD fabric.
	cal_blk_clk	1	1	From PLD fabric.
Bank16 Basic 4ch	pll_inclk	1	0	Using one REFCLK pin.
	rx_cruclk	4	0	Connect to pll_inclk.
	rx_clkout	4	4	To PLD fabric.
	tx_clkout	4	1	To PLD fabric.
Bank17 PCIE x4	pll_inclk	1	0	Using one REFCLK pin.
	rx_cruclk	4	0	Connect to pll_inclk.
	coreclkout	1	1	To PLD fabric.
	fixedclk	1	1	From PLD fabric.
Region2 LRIO Clock			8/8 (or 7/8, 6/8)	
Region3 LRIO Clock			5/8 (or 6/8, 7/8)	

Multiple Protocols and Data Rates in a Transceiver Block

Stratix II GX supports multiple protocols and/or data rates in a single transceiver block. This allows for better utilization of the channels and power savings. There can be up to four independent data rates supported and up to two separate frequencies for the TX channels and up to four separate frequencies for the RX channels within a single transceiver block.

On the TX side, the TX local dividers and the two TXPLLs in the central block can be used together or separately to achieve multiple data rates and/or protocols in a transceiver block. Refer to [“Transmitter Local Clock Divider Block” on page 2–16](#) and [“Central Clock Divider Block” on page 2–14](#) for more information regarding the TX local divider blocks and the dual TXPLL configuration.

On the RX side, it is possible to have up to four receiver channels to be of different data rates and configurations as long as there are enough PLD interface clocks to support the channels. Since each receiver channel contains a dedicated RXPLL, there are no data rate or configuration restrictions.

Quartus II software automatically combines multiple ALT2GXB Megafunction instances into a transceiver block if possible. If there is a particular placement required, Altera recommends that you force the placement via TX and/or RX channel pin assignment in the assignment editor. Quartus II software checks to see if the desired placement is possible. If not, the Quartus II software sends a fitter error.

Since the TX channel parameters are the deciding factors on whether the channels can be combined in a single transceiver block, the following sections will concentrate on the TX side.

Transceiver Block-Based Controls

First, in order to combine channels into a single transceiver block, the transceiver block-based control signals must be driven from the same source.

The following is a list of transceiver block-based control signals:

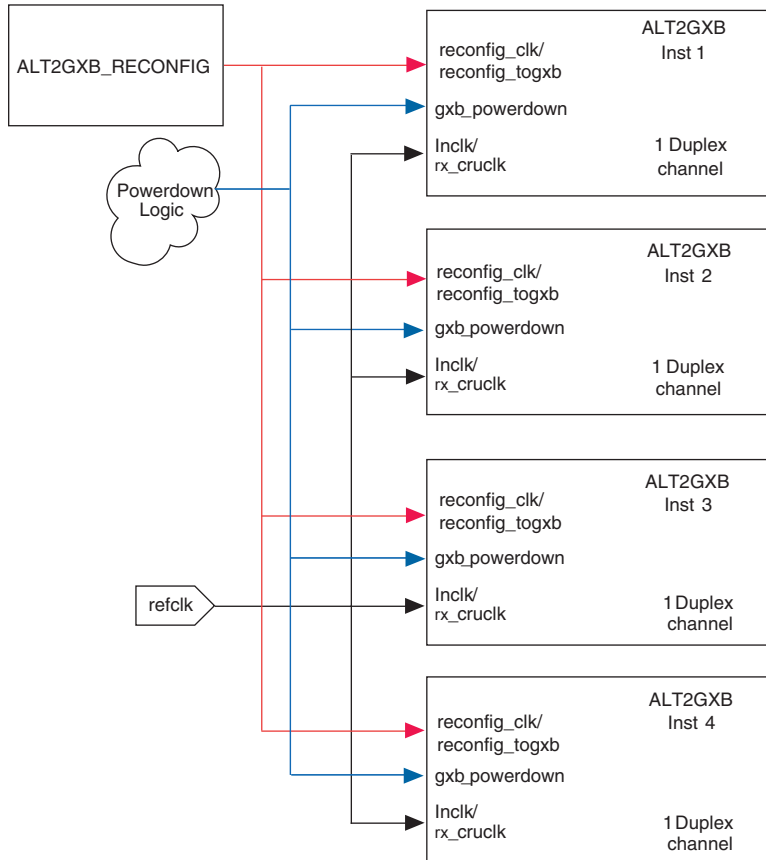
- `gxb_powerdown`
- `reconfig_clk`
- `reconfig_togxb`

The `gxb_powerdown` signal of all the instances that are to be combined in a single transceiver block must be connected to a single point; for example, the same input pin or same logic. Any driving logic differences will prevent the instance from being combined in a single transceiver block.

If you use dynamic reconfiguration, only one dynamic reconfiguration controller is allowed to drive the instances to be combined in a single transceiver block. If you use multiple dynamic reconfiguration controllers, the Quartus II software will not be able to combine the instances into a single transceiver block as the PLD logic cannot be combined. All the `reconfig_clk` and `reconfig_togxb` ports need to be tied to a single dynamic reconfiguration block.

Figure 2–105 is an example of correct transceiver block-control signal connection.

Figure 2–105. Transceiver Block-Control Signal Connections



TXPLL Sharing

Multiple channels in using the Basic protocol can share a common TXPLL. Their data path configuration can differ as long as they are within the same width mode; for example, if all operate in single-width mode or if all operate in double-width mode. Mixing single-width and double-width modes is not allowed for TXPLL sharing.

The data rate range of single-width versus double-width mode is listed in [Table 2–31](#). If the desired data rate range falls within a mode using the TX local divider factors ($/1$, $/2$, or $/4$), you can use a single TXPLL to support the desired data rate range. If the desired data rate range straddles two modes, TXPLL sharing cannot be done. They can still be in the same transceiver block, but you will need to use two TXPLLs.


Mode	Minimum Data Rate	Maximum Data Rate
Single Width	600 Mbps	3.125 Gbps
Double Width	1 Gbps	6.375 Gbps

In order to share a TXPLL, their PLL configurations are required to be the same. The following is a list of TXPLL parameters that must be identical:

- PLL bandwidth
- Primary data rate
- Reference clock frequency
- Pre-divider on the dedicated REFCLK (if applicable)

Though the TXPLL settings must be identical, the TX local divider settings on the channels can vary. You will need to set up the TXPLL for the highest data rate and use the dividers to drop the data rate down on the slower channels to $/4$ or $/2$ of the primary data rate. Use the ALT2GXB MegaWizard to make your changes.

Non-Basic protocol modes cannot share a TXPLL unless they are all of the same protocol and sub protocol.

 TXPLL sharing is restricted to the channels within a transceiver block.

Also, the analog buffer voltage setting `VCCH` must be the same across all the channels in the transceiver block. The Quartus II software will not allow channels with different `VCCH` settings into the same transceiver block.

The following is an example of instances that can be combined into a single transceiver block. We have two 4 Gbps, one 2 Gbps, and one 1 Gbps links. Since the target data rate is either a $/1$, $/2$, or $/4$ division factor from the highest data rate, it is possible to combine this into a single transceiver block. It is assumed that the `VCCH`, transceiver block signals

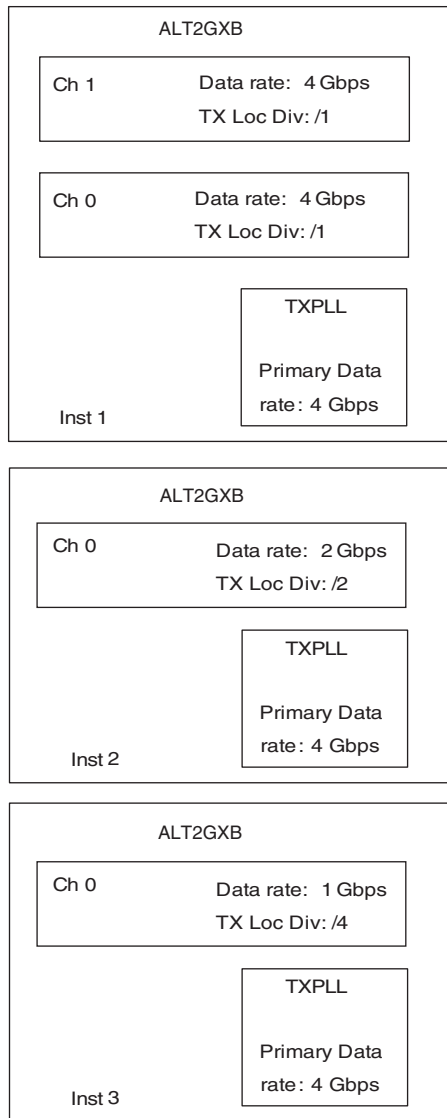
(for example, `gxb_powerdown`), Dynamic Reconfig ports, reference clock frequency, and PLL bandwidth of each instance are the same and/or are driven from the same point.

Each instance must be created with the same primary data rate set a 4 Gbps so that the TXPLLs can be combined. Use the local dividers to achieve the desired data rate for each channel. Because the goal is to have all four channels in a transceiver block driven off of a single TXPLL, the reference clock will need to be from the same point - either from a single dedicated `REFCLK` pin or from

Though the example shows that there are two 4 Gbps channels configured in a single `ALT2GXB` instance, duplicate channels may not need to be configured in the same instance. If the two 4 Gbps channels were to be configured in separate instances, the resultant transceiver block configuration will not have changed.

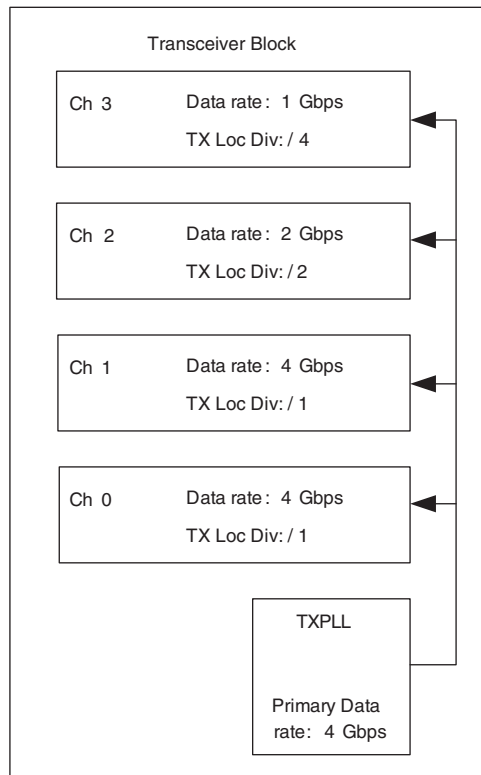
Figure 2-106 shows an example of TXPLL sharing compatible instances.

Figure 2-106. TXPLL Sharing Compatible Instances



The instances shown in Figure 2-106 net the transceiver block configuration shown in Figure 2-107 after compilation. The specific data rate on the channel location may differ depending on either the placement algorithm or your assignments. Since all the channels in this transceiver block are utilized, the second TXPLL is not used. However, you can use the second dedicated REFCLK input to feed the IQ lines or the PLD logic if you do not use the first dedicated REFCLK to drive the IQ lines or PLD logic.

Figure 2-107. Resultant Transceiver Block Configuration After Combining Instances



Using Two TXPLLs

If the desired data rates and/or protocols cannot utilize a single TXPLL within a single transceiver block, the other TXPLL within that transceiver block may be able to support the additional data rates and/or protocol configuration.

This is useful if combining channels operating in single-width and double-width modes, or two Quartus II software-defined protocols (for example, Basic, GIGE, SONET/SDH, SDI, or PCI Express [PIPE] modes) in the same transceiver block. You can configure channels in the same protocol group from individual ALT2GXB instances, or you can configure the whole group in a single instance.

The example in [Figure 2-108](#) will not be able to share a single TXPLL due to incompatible primary data rates. Since there are only two different primary data rates, you can merge the four channels into a single transceiver block. It is assumed that the channels sharing the same TXPLL will meet the requirements described in the above example. For the channels not sharing the same TXPLL, the PLL parameters may be different; for example, different data rate, reference clock input frequency, reference clock input pin, and PLL bandwidth. Also, the primary data rates need not be a multiple of the other TXPLL primary data rate. The transceiver block-based signals will have to be the same across all channels.

In addition to the primary data rate differences, the mode of operations also differs which require the use of separate TXPLLs. The 4 Gbps channels operate in a double-width mode while the 2 Gbps and 1 Gbps channels operate in a single-width mode (due to the sub 3.125 Gbps primary data rate).

Figure 2-108. TXPLL Sharing Incompatible Instances

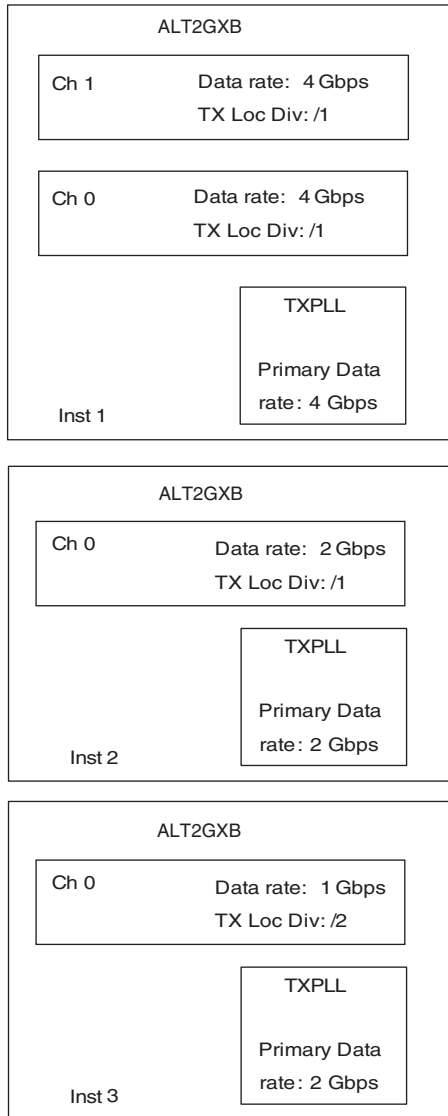
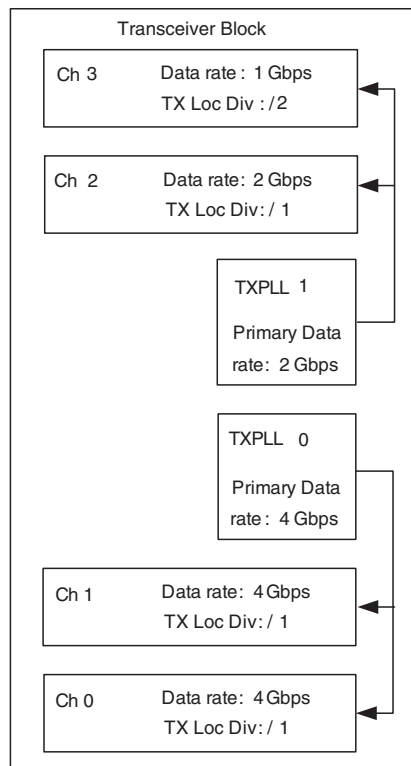


Figure 2-109 is the resultant transceiver block configuration after combining the above instances. Since the two TXPLLs primary data rates can be derived from the same reference clock frequency, one reference clock input is needed. If the reference clock frequency differs, or if the primary data rates differ (for example, 2.488 Gbps instead of 2 Gbps), two reference clocks will be needed.



If the reference clock is driven from the PLD, only one connection exists for each transceiver block. Both TXPLLs in a transceiver block cannot be driven from separate PLD clock pins. If the reference clock frequencies are the same, it is possible to drive both TXPLLs in a transceiver block from a single PLD clock pin (or the dedicated REFCLK pin).

Figure 2-109. Resultant Transceiver Block Configuration with Two TXPLLs



Combining RX Channels in a Transceiver Block

Receiver channels of different configuration and data rates can be combined in to a single transceiver block. Since each receiver channel contains a dedicated RXPLL, there are no data rate or configuration restrictions. However, there are some restrictions. One restriction is for the dedicate reference clock usage. All the RX channels referenced off of the same dedicated REFCLK pin must have the same input frequency and the same usage of the REFCLK pre-divider. The other restriction is that since this is a RX-only configuration, the use of the rate matcher block is not allowed. If rate matching is needed, you will need to implement the rate matching FIFO in the PLD core.

It is possible to have up to four receiver channels to be different data rates and configurations, as long as there are enough PLD interface clocks to support the channels. As with the TX channels, each receiver channel can be configured from separate ALT2GXB instances or the entire group (as long as all the RX channels are the same) can be configured in one instance.

Combining RX and TX channels in a Transceiver Block

You can combine duplex channels and/or a mixture of TX and RX channel instances into a single transceiver block. For combining the duplex channel configuration, the TX rules and restrictions is a superset of the RX side and is covered in [“TXPLL Sharing” on page 2–137](#) and [“Using Two TXPLLs” on page 2–142](#).

The rules and restrictions for a combination of separate RX and TX channel instances in the same transceiver block are the same as outlined in [“TXPLL Sharing” on page 2–137](#) and [“Using Two TXPLLs” on page 2–142](#) for the transmitter and [“Combining RX Channels in a Transceiver Block” on page 2–145](#).

Native Modes

The Stratix II GX transceiver operates in one of nine native modes:

- Basic
 - Single-width mode (600 Mbps to 3.125 Gbps)
 - Double-width mode (1 Gbps to 6.375 Gbps)
- PCI Express (PIPE) mode (2.5 Gbps)
- XAUI (3.125 Gbps up to “HiGig” 3.75 Gbps)
- GIGE (1.25 Gbps)
- SONET/SDH mode (OC-12, OC-48, and OC-96)
- (OIF) CEI PHY Interface (>3.135 Gbps to 6.375 Gbps)
- Serial RapidIO (1.25 Gbps, 2.5 Gbps, 3.125 Gbps)
- SDI (HD, 3G)
- CPRI (614 Mbps, 1.228 Gbps, 2.456 Gbps)

Basic Single-Width Mode

Use the Basic single-width mode for custom protocols that are not part of the pre-defined supported protocols, for example PIPE. With some restrictions, the following PCS blocks are available:

- Transmitter phase compensation FIFO buffer
- Transmitter byte serializer
- 8B/10B encoder
- Word aligner
- Rate matcher
- 8B/10B decoder
- Byte deserializer
- Byte ordering block
- Receiver phase compensation FIFO buffer

The byte ordering block is available only in reverse serial loopback configuration in Basic mode. The rate matcher is coupled with the 8B/10B code groups, which requires the use of the 8B/10B encoder or decoder either in the PCS or PLD logic array.

Basic Single-Width Mode with x4 Clocking

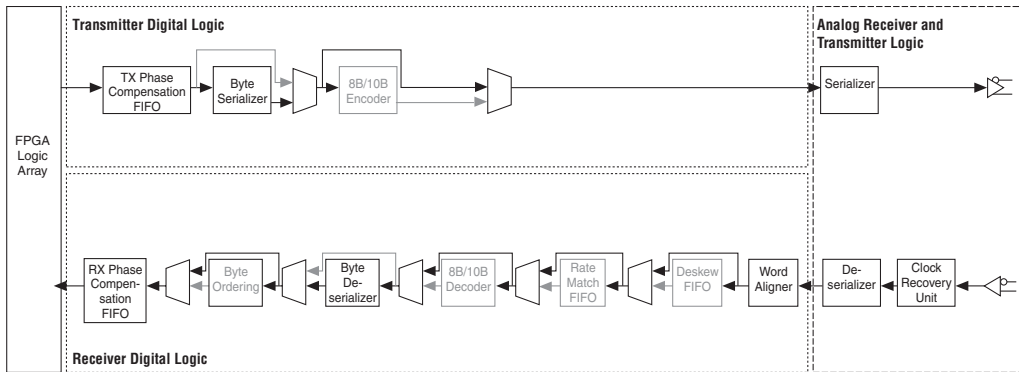
In basic single-width mode, the ALT2GXB MegaWizard provides a x4 option under the **Which subprotocol will you be using?** option. If you select this option, all four transmitter channels within the transceiver block are clocked by clocks generated from the central clock divider block (refer to “[Transmitter Clocking \(Bonded Channels\)](#)” on page 2-29). The low-speed clock from the central clock divider block clocks the bonded transmitter PCS logic in all four channels. This reduces the transmitter channel-to-channel skew within the transceiver block. Each receiver channel within the transceiver block is clocked individually by the recovered clock from its own clock recovery unit (CRU).



Configuring transceivers in this mode yields low transmitter channel-to-channel skew within a transceiver block. It does not provide skew reduction for channels placed across transceiver blocks.

Figure 2–110 shows the data path in this mode.

Figure 2–110. Basic Single-Width Mode with $\times 4$ Clocking



The transmitter data path consists of a 16-bit PLD-transceiver interface, transmitter phase compensation FIFO, 16:8-bit byte serializer, and 8:1 serializer.

The receiver data path consists of the clock recovery unit (CRU), 1:8 deserializer, bit-slip word aligner, 8:16 byte deserializer, receiver phase compensation FIFO, and 16-bit Transceiver-PLD interface.

Transceiver Placement Limitations

If one or more channels in a transceiver block are configured to Basic single-width mode with $\times 4$ clocking option enabled, the remaining channels in that transceiver block must either have the same configuration or must be unused. All used channels within a transceiver block configured to this mode must also run at the same data rate. All channels within the transceiver block configured to this mode must be instantiated using the same ALT2GXB MegaWizard instance.

Figures 2–111 and 2–112 show examples of legal and illegal transceiver placements with respect to the Basic single-width mode with ×4 clocking enabled.

Figure 2–111. Examples of Legal Transceiver Placement

Ch0	Basic Single-Width mode with x4 clocking option enabled	Ch0	Basic Single-Width mode with x4 clocking option disabled
Ch1	Basic Single-Width mode with x4 clocking option enabled	Ch1	Basic Single-Width mode with x4 clocking option disabled
Ch2	Unused Channel	Ch2	Serial RapidIO
Ch3	Unused Channel	Ch3	Serial RapidIO

Figure 2–112. Examples of Illegal Transceiver Placement

Ch0	Basic Single-Width mode with x4 clocking option enabled	Ch0	Basic Single-Width mode with x4 clocking option enabled
Ch1	Basic Single-Width mode with x4 clocking option enabled	Ch1	Basic Single-Width mode with x4 clocking option enabled
Ch2	Serial RapidIO	Ch2	Basic Single-Width mode with x4 clocking option disabled
Ch3	Serial RapidIO	Ch3	Basic Single-Width mode with x4 clocking option disabled

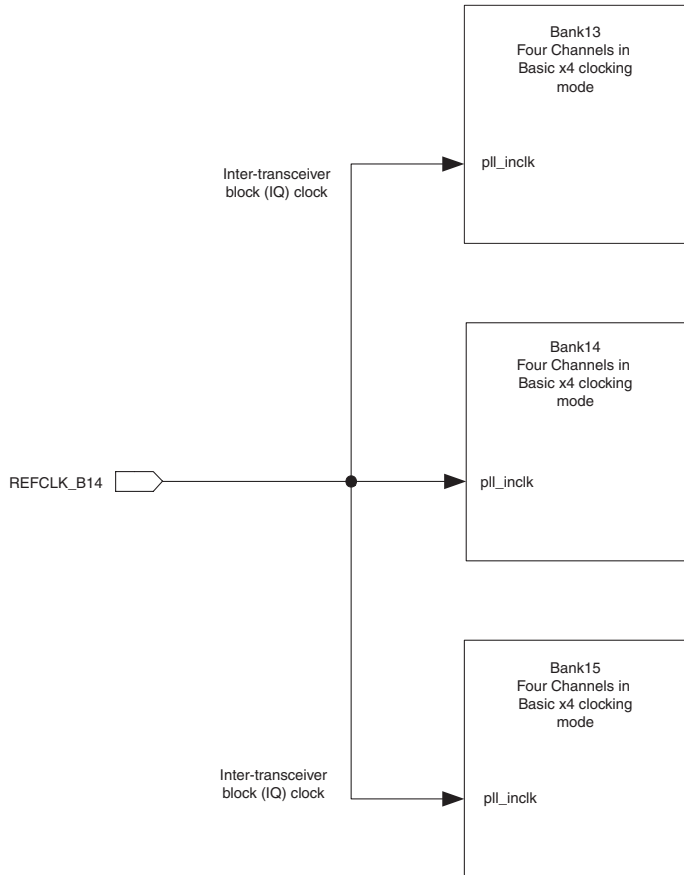
Clocking and Reset Recommendations

To minimize the transmitter channel to channel skew across transceiver blocks, Altera recommends:

- Using the dedicated REFCLK pins of the centrally located transceiver block in your design to provide the input reference clock for all transceiver blocks. This reduces the skew on the input reference clock driving the CMU PLL in each transceiver block. For example, in a design with 12 channels placed across Banks 13, 14, and 15, use the REFCLK pins of Bank 14 to provide the input reference clock. In a design with 16 channels placed across Banks 13, 14, 15, and 16, use the REFCLK pins of either Bank 14 or 15.
- De-asserting the tx_digitalreset signal of all used transceiver blocks simultaneously after pll_locked signal from all active transceiver blocks goes high.

Figure 2–113 shows the recommended clocking for 12 transceiver channels across transceiver banks 13, 14, and 15 in the EP2SGX90EF1152 device.

Figure 2–113. Clocking Recommendations to Minimize Transmitter Channel-To-Channel Skew



Basic Double-Width Mode

Use Basic double-width mode for custom protocols that are not part of the pre-defined supported protocols, for example, PIPE. With some restrictions, the following PCS blocks are available:

- Transmitter phase compensation FIFO buffer
- Transmitter byte serializer

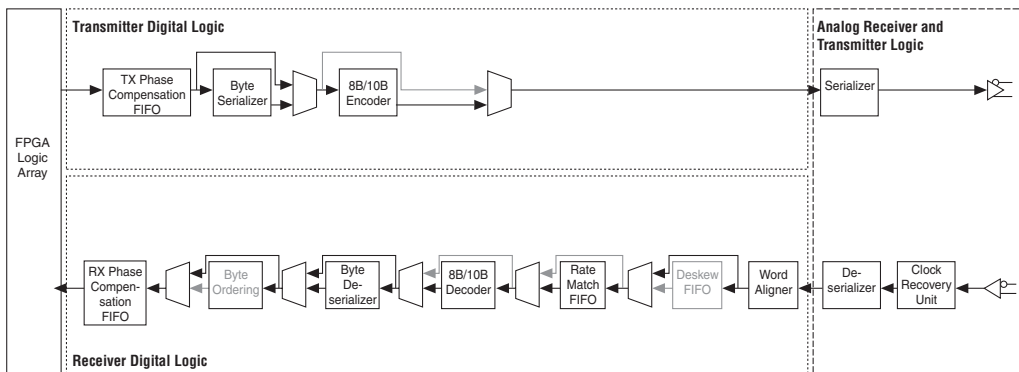
- 8B/10B encoder
- Word aligner
- Rate matcher
- 8B/10B decoder
- Byte deserializer
- Byte ordering block
- Receiver phase compensation FIFO buffer

The rate matcher is not available with the byte ordering block and vice versa. Because the rate matcher removes one byte at a time, the ordering of the blocks changes if rate matching occurs. The rate matcher is coupled with the 8B/10B code groups, which require the use of the 8B/10B encoder or decoder either in the PCS or PLD logic array.

PCI Express (PIPE) Mode

PCI Express is an evolution of Peripheral Component Interconnect (PCI). PCI is bandwidth-limited for today's applications because it relies on synchronous single-ended type signaling with a wide multi-drop data bus (refer to [Figure 2-114](#)). Clock and data-trace matching is required with PCI. PCI Express uses differential serial signaling with an embedded clock to enable an effective through-put of 2 Gbps per link to circumvent the limitations of PCI. PCI Express operates in $\times 1$, $\times 4$, $\times 8$, $\times 16$, and $\times 32$ configurations and is also backward compatible with PCI on a software and driver level.

Figure 2-114. PIPE Mode



Stratix II GX devices support the PIPE standard in $\times 1$, $\times 4$, and $\times 8$ configurations.

The Stratix II GX device has dedicated circuits to support the PCI Express protocol, including the following:

- 8B/10B encoder and decoder
- Rate matcher, which supports a multi-crystal environment up to ± 300 PPM (600 PPM total) clock difference
- PIPE interface (Physical Interface for PCI Express)
- Receiver detection
- Beacon transmit capability
- Loopback
- Inversion
- Disparity control

The PHY state machines, except for rate matching, are not included in the transceiver. Those state machines can be created in the PLD logic. This mode of operation is called the PIPE mode. The PIPE mode has a separate reset sequence. Refer to [“Reset Sequence for PIPE Mode” on page 2–218](#) for more information.

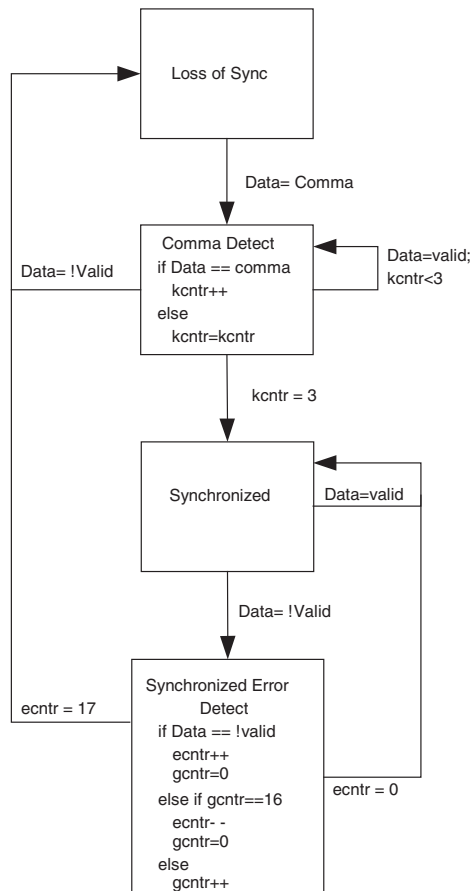


The equalizer DC gain value in the MegaWizard Plug-In Manager for PIPE mode is set to a default value of 1. If the equalizer DC gain is controlled by the ALT2GXB_RECONFIG controller, the `rx_eqdcgain` input to the ALT2GXB_RECONFIG controller should be tied to “01” to be PCI E-compliant. Refer to the [ALT2GXB Megafunction User Guide](#) chapter in volume 2 of the *Stratix II GX Device Handbook* for more information.

Synchronization

In PIPE mode, the synchronization automatically occurs when the receiver receives 4 good /K28.5/ commas and 16 good code groups. Synchronization can be accomplished through the reception of four good PCI Express training sequences (TS1 or TS2) or four fast training sequences. [Figure 2–115](#) shows a state diagram of the PCI-E synchronization.

Figure 2-115. PIPE Mode Synchronization State Machine



Tables 2-32 and 2-33 describe the TS1 and TS2 training sequences, respectively. A PCI Express fast training sequence consists of a /K28.5/, followed by three /K28.1/ code group.

If there is one code group error during the synchronization process, resynchronization must be performed.

<i>Table 2–32. PCI Express TS1 Ordered Set</i>			
Symbol Number	Allowed Values	Encoded Values	Description
0		K28.5	Comma code group for symbol alignment
1	0–255	D0.0–D31.7, and K23.7	Link number with component
2	0–31	D0.0–D31.0, and K23.7	Lane number within port
3	0–255	D0.0–D31.7	N_FTS. The number of fast training ordered sets required by the receiver to obtain reliable bit and symbol lock.
4	2	D2.0	Data rate identifier Bit 0–Reserved, set to 0 Bit 1 = 1, generation 1 (2.5 Gbps) data rate supported Bit 2..7–Reserved, set to 0
5	Bit 0 = 0, 1 Bit 1 = 0, 1 Bit 2 = 0, 1 Bit 3 = 0, 1 Bit 4..7 = 0	D0.0, D1.0, D2.0, D4.0, and D8.0	Training control Bit 0–Hot reset Bit 0 = 0, deassert Bit 0 = 1, assert Bit 1–Disable link Bit 1 = 0, deassert Bit 1 = 1, assert Bit 1–Loopback Bit 2 = 0, deassert Bit 2 = 1, assert Bit 3–Disable scrambling Bit 3 = 0, deassert Bit 3 = 1, assert Bit 4..7–Reserved Bit 0 = 0, deassert Set to 0
6–15		D10.2	TS1 identifier

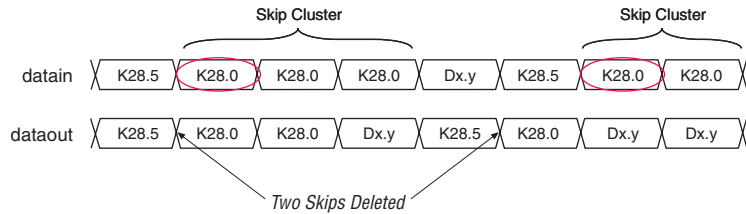
<i>Table 2–33. PCI Express TS2 Ordered Set</i>			
Symbol Number	Allowed Values	Encoded Values	Description
0		K28.5	Comma code group for symbol alignment.
1	0–255	D0.0–D31.7, and K23.7	Link number with component.
2	0–31	D0.0–D31.0, and K23.7	Lane number within port.
3	0–255	D0.0–D31.7	N_FTS. The number of fast training ordered sets required by the receiver to obtain reliable bit and symbol lock.
4	2	D2.0	Data rate identifier Bit 0–Reserved, set to 0 Bit 1 = 1, generation 1 (2.5 Gbps) data rate supported Bit 2..7–Reserved, set to 0
5	Bit 0 = 0, 1 Bit 1 = 0, 1 Bit 2 = 0, 1 Bit 3 = 0, 1 Bit 4..7 = 0	D0.0, D1.0, D2.0, D4.0, and D8.0	Training control Bit 0–Hot reset Bit 0 = 0, deassert Bit 0 = 1, assert Bit 1–Disable link Bit 1 = 0, deassert Bit 1 = 1, assert Bit 1–Loopback Bit 2 = 0, deassert Bit 2 = 1, assert Bit 3–Disable scrambling Bit 3 = 0, deassert Bit 3 = 1, assert Bit 4..7–Reserved Bit 0 = 0, deassert Set to 0
6–15		D5.2	TS2 identifier

Rate Matching

In PIPE mode, the rate matcher supports up to ± 300 PPM (600 PPM total) differences between the upstream transmitter and the receiver. The rate matcher looks for skip ordered sets, which usually contain a /K28.5/comma followed by three /K28.0/ skip characters. The rate matcher deletes or inserts skip characters when necessary to prevent the rate matching FIFO buffer from overflowing or underflowing.

The rate matcher can delete skip characters on only one skip character in a consecutive cluster of skip characters. Figure 2–116 shows an example of a PIPE mode rate matcher deletion of two skip characters.

Figure 2–116. PIPE Mode With Two Deletions (One Deletion Per Cluster)



The rate matcher can insert skip characters one insertion per skip cluster. There is no limit on the consecutive number of skip characters allowed per skip cluster.

The Stratix II GX rate matcher in PIPE mode has FIFO buffer overflow and underflow protection. In the event of a FIFO buffer overflow, the rate matcher deletes any data after the overflow condition to prevent FIFO pointer corruption until the rate matcher is not full. In an underflow condition, the rate matcher inserts 9'h1FE (/K30.7/) until the FIFO buffer is not empty. These measures ensure that the FIFO buffer can gracefully exit the overflow/underflow condition without requiring a FIFO reset.

Power State

There are four supported power states in Stratix II GX devices: P0, P0s, P1, and P2. P0 is the normal power state. P0s is a low recovery time power state that is lower than P0. P1 is a lower power state than P0s and have higher latency to come out of this state. P2 is the lowest power state supported by this mode.

The `powerdn` port transitions the transceiver into different power states. The encoded value is shown in [Table 2–34](#). The `pipephydonestatus` signal reacts to the `powerdn` request and pulses high for one parallel clock cycle.

There are specific functions that are performed at each of the power states. The power-down states are for PIPE emulation. The transceiver does not go into actual power saving mode, with the exception of the transmitter buffer for Electrical Idle. This shouldn't matter because the power consumption for the PLD logic is much greater than the transceiver power consumption. Powering down the transceiver does not save much when compared to the overall power consumption of the entire device. [Table 2–34](#) shows each power state and its function.

Table 2–34. Power State Functions and Descriptions			
Power State	powerdn	Function	Description
P0	2'b00	Transmits normal data, transmits Electrical Idle, or enters into loopback mode.	Normal operation mode
P0s	2'b01	Only transmits Electrical Idle.	Low recovery time power saving state
P1	2'b10	Transmitter buffer is powered down and can do a receiver detect while in this state.	High recovery time power saving state
P2	2'b11	Transmits Electrical Idle or a beacon to wake up the downstream receiver.	Lowest power saving state

There are two signals associated with the power states: `tx_detectrxloopback` and `tx_forceelecidle`. The `tx_detectrxloopback` signal controls whether the channel goes into loopback when the power state is in P0 or receiver detect when in P1 state. This signal does not have any affect in any other power states. The `tx_forceelecidle` signal governs when the transmitter goes into an Electrical Idle state. The `tx_forceelecidle` signal is asserted in P0s and P1 states and deasserted in P0 state. In P2 state, under normal conditions, the `tx_forceelecidle` signal is asserted and then deasserted when the beacon signal must be sent out, signifying the intent to exit the P2 power-down state. [Table 2–35](#) shows the behavior of the `tx_detectrxloopback` and `tx_forceelecidle` signals in the power states.

Table 2–35. Power States and Functions Allowed in Each Power State

Power State	tx_detectrxloopback	tx_forceelecidle
P0	0: normal mode 1: data path in loopback mode	0: Must be deasserted. 1: Illegal mode
P0s	Don't care	0: Illegal mode 1: Must be asserted in this state
P1	0: Electrical Idle 1: receiver detect	0: Illegal mode 1: Must be asserted in this state
P2	Don't care	Deasserted in this state for sending beacon. Otherwise asserted.

Receiver Status

The PIPE interface for PCI Express has a receiver status indicator that reports the status of the PHY (PCS and PMA). The receiver status is communicated to the PLD logic by the 3-bit `pipestatus` port. This port enumerates the status as shown in Table 2–36. If more than one event occurs at the same time, the signal is resolved with the higher priority status. The skip character added and removed flags (3'b001 and 3'b010) are not supported. The `pipestatus` port may be encoded to 3'b'001 and 3'b'010, which should be ignored. It does not indicate that a skip has been added or removed and should be considered the same as 3'b'000—received data. If the upper MAC layer must know when a skip character was added or removed, Altera recommends monitoring the number of skip characters received. The transmitter should send three skip characters in a standard skip-ordered set.

Table 2–36. pipestatus Description and Priority

pipestatus	Description	Priority
3'b000	Received data	6
3'b001	One skip character added (not supported)	N/A
3'b010	One skip character removed (not supported)	N/A
3'b011	Receiver detected	1
3'b100	8B/10B decoder error	2
3'b101	Elastic buffer overflow	3
3'b110	Elastic buffer underflow	4
3'b111	Received disparity error	5

An additional status port, `rx_pipedatavalid`, indicates that the data on the `rx_dataout` port is valid. This signal is equivalent to the `rx_syncstatus` port. The `rx_pipedatavalid` port operates in parallel with the `pipestatus` signal.

Receiver Detect

The receiver detect circuitry is available for PCI Express applications. The receiver detect circuitry is only available in the P1 power state and is set through the `tx_detectrxloopback` port and requires a 125 MHz `fixedclk` signal. In the P1 power state, a high on `tx_detectrxloopback` port triggers the receiver detect circuitry to alter the transmitter buffer common mode voltage. The sudden change in common mode voltage effectively appears as a step voltage at the serial link. If a receiver (that complies to PCI-Express input impedance requirements) is present at the far end, the time constant of the step voltage is higher. If a receiver is not present or powered down, the time constant of the step voltage is lower. The receiver detect circuitry snoops the serial line (`tx_dataout`) for the time constant of the step voltage to detect the presence of the receiver at the far end.

Upon receiver detect, the `pipestatus` port indicates if a receiver is detected or not. There is some latency after asserting the `tx_detectrxloopback` signal, before the receiver detection is indicated. The `tx_forcelecidle` port must be asserted at least 10 parallel clock cycles prior to the `tx_detectrxloopback` to ensure that the transmitter buffer is tri-stated.

Beacon Transmission

The beacon is an optional 30 kHz to 500 MHz in-band signal that wakes the receiver from a P2 power state. This signal is optional, and the Stratix II GX device does not have dedicated beacon transmission circuitry. The Stratix II GX device supports the transmission of the beacon signal through a 10-bit encoded code word that has a five 1's pulse (for example, K28.5).

Because the beacon signal is a pulse that ranges from 2 ns to 500 ns, sending out a K28.5 at 2.5 Gbps meets the lower requirement with its five 1's pulse. (Though other 8B/10B code groups might meet the beacon requirement, this document uses the K28.5 control code group as the beacon signal.) The beacon transmission takes place only in the P2 power state. The `tx_forcelecidle` port controls when the transmitter is in Electrical Idle or not. This port must be deasserted in order to transmit the K28.5 code group.

Compliance Pattern Transmission Support

PCI Express has an option to transmit a compliance pattern for testing purposes. The compliance pattern must be transmitted with a negative disparity. In PIPE mode, you set the negative disparity with the `tx_forcedispcompliance` port.

Asserting the `tx_forcedispcompliance` port sets the associated byte in the `tx_datain` port to be encoded, by the 8B/10B encoder, to a negative disparity. If a wider PLD interface is used, only the LSB byte is encoded with a negative disparity. The `tx_forcedispcompliance` port must be deasserted after the first byte of the compliance pattern is clocked into the transceiver.

The compliance pattern generator is not part of the Stratix II GX transceiver and must be designed using the PLD logic. However, you can set the beginning of the disparity of the compliance pattern to negative by asserting the `tx_forcedispcompliance` port.

NTFS Fast Recovery IP (NFRI)

The PCI-E specification fast training sequences (FTS) are used for bit and byte synchronization to transition from L0s state to L0 (Stratix II GX P0s to P0) power states. The PCI-E base specification states that the required time period for this transaction be within 16 ns to 4 us. Currently, the default PIPE ALT2GXB settings do not meet these requirements. Therefore, Altera developed NTFS fast recovery IP (NFRI), a soft IP that enables the receiver to transition from the P0s to the P0 state within 4 us. The Quartus II software creates this NFRI soft IP when the **Enable fast recovery mode** option is selected in the ALT2GXB MegaWizard Plug-In Manager. This option is available from the Quartus II software version 6.0, SP1.

FTS ordered sets are used by the receiver to detect exit from Electrical Idle (EIdle) and align the receiver's bit/symbol receiver circuitry to the incoming data. If the FTS time period (4 us) expires prior to the receiver obtaining alignment and deskew on all lanes, the receiver transitions to the recovery state (the PCI Express [PIPE] ALT2GXB performs only word alignment. The deskew operation should be done in the user logic). Following the electrical idle condition, the Stratix II GX device requires 255 FTS sequences to recover valid data.

PIPE Mode Default Settings

In the PIPE mode default settings, the receiver PLL is in automatic lock mode. The PLL moves from lock to reference mode to lock to data mode based on the `rx_freqlocked` being asserted. For the `rx_freqlocked` signal to be asserted, the CRU PLL clock should be within the PPM threshold settings of the CRU reference clock.

The PPM detector checks the PPM threshold settings by comparing the CRU PLL clock output with the reference clock for approximately 32768 clock cycles. For a 250 MHz PLD interface clock frequency, this comparison time period exceeds 4 us, which violates the PCI-E specification. The NTFS soft IP overcomes the restriction.

Enable Fast Recovery Mode Option

When you select the **Enable fast recovery mode** option, consider the following:

- NFRI is created in the PLD side for each PCI-E channel
- NFRI is a soft IP, so it consumes logic resources
- This block is self-contained, so no input/output ports are available to access the soft IP

The NFRI takes control of the `rx_locktorefclk` and `rx_locktodata` signals on the ALT2GXB transceiver and therefore avoids the delay of the PPM detector discussed in [“PIPE Mode Default Settings” on page 2–160](#).



If you select the `rx_locktorefclk` and `rx_locktodata` signals in the MegaWizard Plug-In Manager, the **Enable fast recovery mode** option cannot be used.

NTFS Fast Recovery IP (NFRI) in Software Versions before Quartus II 7.2

This section discusses the solution to control the sequence of events around the results of the 4 us timing requirement during the P0s to P0 state transition. This is only applicable for software versions before Quartus II 7.2 with the transceiver channel configured in PCI Express (PIPE) mode and the optional **fast recovery mode** enabled in the ALT2GXB MegaWizard. Use this for synchronous and non-synchronous PCI Express (PIPE) modes. The solution requires that you use the data in user logic in the core only if the data is valid.

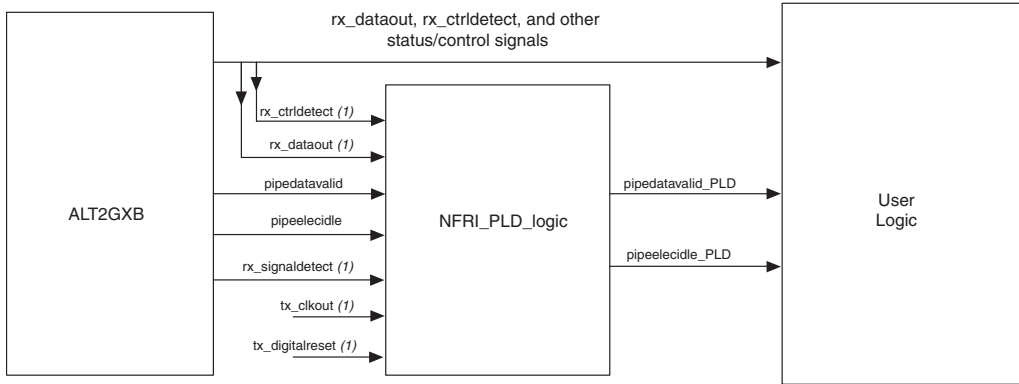


In the Quartus II software version 7.2, the solution is built into the software when you enable the **fast recovery mode** option in the ALT2GXB MegaWizard.

Some logic can be added per channel which controls the circuit with different possible results. The proposed logic in the PLD is referred to as **NFRI_PLD_logic**. The output signals generated by the **NFRI_PLD_logic** are referred to as **pipeelecidle_PLD** and **pipedatavalid_PLD**.

Figure 2–117 shows the top-level block diagram of the overall system.

Figure 2–117. **NFRI_PLD_LOGIC Top Level Diagram**



Note to Figure 2–117:

(1) This signal is also provided to the user logic.



The proposed logic runs with the `tx_clkout`. To reset this logic, use the same signal that connects to the `tx_digitalreset` port of the PCI-Express (PIPE) ALT2GXB instance.

The **NFRI_PLD_logic** contains:

- A state machine sequence to generate the `pipeelecidle_PLD` and `pipedatavalid_PLD` signals.
- `4us_timer`: A user-implemented timer in the PLD logic to count 4 us. This timer represents the maximum time period to transition from P0s to P0 per the protocol specification.
- `3.2us_timer`: A user-implemented timer in the PLD logic to count 3.2 us. This timer represents the minimum time period to wait before looking for valid data.

Sequence to Generate pipeelectedle_PLD and pipedatavalid_PLD Signals

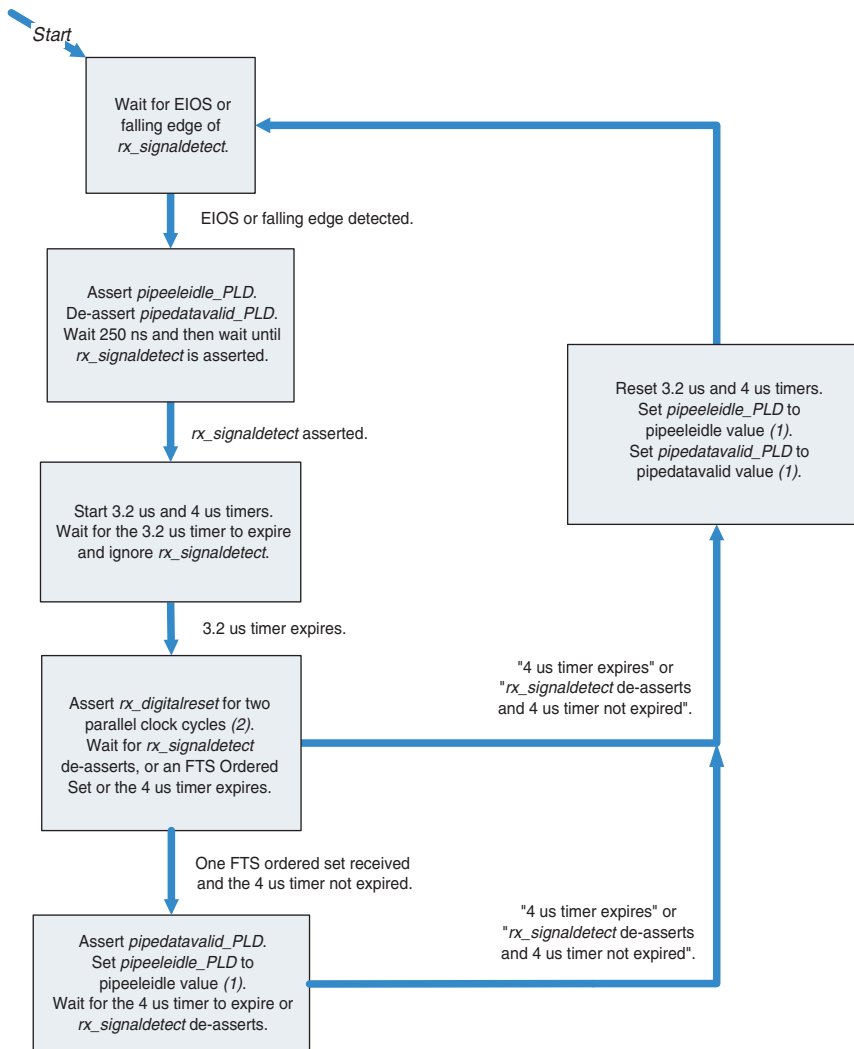
To generate the pipeelectedle_PLD and pipedatavalid_PLD signals, follow these steps:

1. When you detect EIOS ordered set or the falling edge of rx_signaldetect:
 - assert pipeelectedle_PLD
 - deassert pipedatavalid_PLD
2. Wait 250 ns. This is the minimum time required for rx_signaldetect to get deasserted when there is no valid signal at the receive input.
3. Wait for rx_signaldetect to get asserted.
4. Start the 3.2us_timer and 4us_timer. Ignore rx_signaldetect during this step.
5. Wait for the 3.2us_timer to expire. Ignore rx-signaldetect during this step.
6. If one FTS ordered set is received and the 4us_timer has NOT expired:
 - Set pipeelectedle_PLD to the pipeelectedle value (that is, forward the pipeelectedle value from the ALT2GXB to the user logic).
 - Assert pipedatavalid_PLD.
 - Reset and pause the 3.2us_timer and 4us_timer.
 - Return to Step 1.
7. If rx_signaldetect gets deasserted and the 4us_timer has NOT expired:
 - Reset and pause the 3.2us_timer and 4us_timer.
 - Set pipeelectedle_PLD to the pipeelectedle value (that is, forward the pipeelectedle value from the ALT2GXB to the user logic). Set the pipedatavalid_PLD to the pipedatavalid value (that is, forward the pipedatavalid value from the ALT2GXB to the user logic).
 - Return to Step 1.

8. If the `4us_timer` has expired:
 - Reset and pause the `3.2us_timer` and `4us_timer`.
 - Set the `pipeecidle_PLD` to the `pipeecidle` value (that is, forward the `pipeecidle` value from the ALT2GXB to the user logic).
 - Set `pipedatavalid_PLD` to the `pipedatavalid` value (that is, forward the `pipedatavalid` value from the ALT2GXB to the user logic).
 - Return to Step 1.

Figure 2–118 shows the NFRI_PLD_logic state machine.

Figure 2–118. NFRI_PLD_logic State Machine

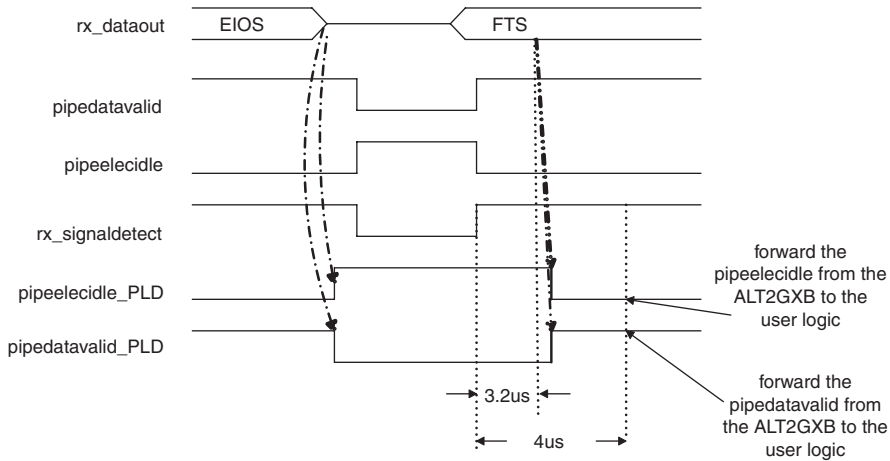


Notes to Figure 2–118:

- (1) Forward the ALT2GXB value to the NFRI_PLD_logic output. For example, forward the pipedatavalid value from the ALT2GXB to the pipedatavalid_PLD for use in user logic.
- (2) After the 3.2us_timer expires, it is important to reset rx_digitalreset for two parallel clock cycles. For example, now that correct data is coming through, the FIFO should be cleared and the pointers reset to the middle. Following this reset, look for the FTS ordered set. For other cases to assert rx_digitalreset, refer to “Reset Sequence for PIPE Mode” on page 2–218.

Figure 2–119 shows the timing diagram for the signals generated by the NFRI_PLD_logic.

Figure 2–119. Timing for the Signals Generated by the NFRI_PLD_logic



Low-Latency PIPE mode

The Stratix II GX receiver data path employs a rate match FIFO in PIPE mode to compensate up to ± 300 PPM difference between the upstream transmitter and the local receiver reference clock. The rate match FIFO adds a latency of 12-16 parallel clock cycles to the link. The low-latency PIPE mode allows bypassing the rate match FIFO in synchronous systems that derive the transmitter and receiver reference clocks from the same source. You can bypass the rate match FIFO by not selecting the **Enable Rate match FIFO** option in the ALT2GXB MegaWizard Plug-In Manager.

You can bypass the rate match FIFO in single-lane ($\times 1$), four-lane ($\times 4$) and eight-lane ($\times 8$) PIPE modes. In normal PIPE mode, the receiver blocks following the rate match FIFO are clocked by `tx_clkout` ($\times 1$ mode) or `coreclkout` ($\times 4$ and $\times 8$ modes) of the local port. In low-latency mode, since the rate match FIFO is bypassed, these receiver blocks are clocked by the recovered clocks of the respective channels. As a result, the channels in a multi-lane ($\times 4$ or $\times 8$) low-latency PIPE mode are unbonded.

Except for the rate match FIFO being bypassed and the resulting changes in transceiver internal clocking, the low-latency PIPE shares the same data path and state machines as the normal PIPE mode. However, some features supported in normal PIPE mode are not supported in low-latency PIPE mode.



Unlike regular PIPE mode, the Quartus II software in low-latency PIPE mode does not automatically clock the transmitter and receiver phase compensation FIFO write and read clock with the `tx_clkout` signal from channel 0. You must use the **Shared Clock Grouping** or **0 PPM Clock Grouping** assignments to manually clock the phase compensation FIFOs. Refer to “[PLD-Transceiver Interface Clocking](#)” on page 2–119 for more information on clock grouping.

PIPE Reverse Parallel Loopback

In normal PIPE mode, if the transceiver is in P1 power state, a high value on the `tx_rxdetectloop` signal forces a reverse parallel loopback as discussed in “[PCI Express PIPE Reverse Parallel Loopback](#)” on page 2–206. Parallel data at the output of the receiver rate match FIFO gets looped back to the input of the transmitter serializer.

In low-latency PIPE mode, since the rate match FIFO is bypassed, this feature is not supported. A high value on the `tx_rxdetectloop` signal, when the transceiver is in P1 power state, will not force it to perform reverse parallel loopback.

Link Width Negotiation

In normal multi-lane (×4 and ×8) PIPE configuration, the receiver phase compensation FIFO control signals (for example, `write/read enable`) are shared among all lanes within the link. As a result, all lanes are truly bonded and the lane-lane skew meets the PCI Express specification.

In low-latency PIPE configuration, the receiver phase compensation FIFO of individual lanes do not share control signals. The write port of the receiver phase compensation FIFO of each lane is clocked by its recovered clock. As a result, the lanes within a link are not bonded. You should perform external lane de-skewing to ensure proper link width negotiation.

PIPESTATUS Signal

Since the rate match FIFO is bypassed in low-latency PIPE mode, status signal combinations related to the rate match FIFO on `pipestatus[2:0]` port become irrelevant and must not be interpreted ([Table 2–37](#)).

pipestatus[2:0]	Normal PIPE	Low-Latency PIPE
000	Received Data OK	Received Data OK
001	Not supported	Not supported

Table 2–37. Normal and Low-Latency PIPE Status (Part 2 of 2)

pipestatus[2:0]	Normal PIPE	Low-Latency PIPE
010	Not supported	Not supported
011	Receiver Detected	Receiver Detected
100	8B/10B Decoder Error	8B/10B Decoder Error
101	Elastic Buffer Overflow	Not supported
110	Elastic Buffer Underflow	Not supported
111	Received Disparity Error	Received Disparity Error

XAUI Mode

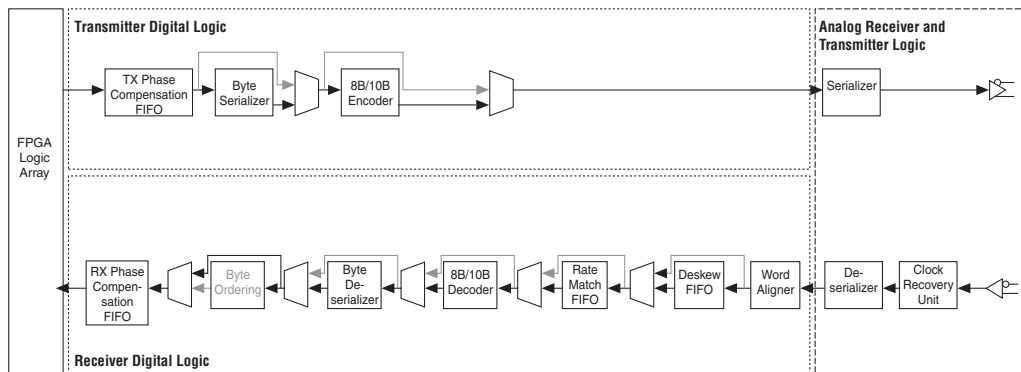
This section briefly introduces the XAUI standard (refer to [Figure 2–120](#)) and the code groups and ordered sets associated with this self-managed interface. For full details on the XAUI standard, refer to clause 47 and 48 in the 10 Gigabit Ethernet standard (IEEE 802.3ae).

Stratix II GX devices contain embedded macros dedicated to the XAUI protocol, including synchronization, channel deskew, rate matching, XGMII Extender Sublayer (XGXS) to 10 Gigabit Media Independent Interface (XGMII) and XGMII to XGXS code-group conversion macros.

For HiGig, the Stratix II GX XAUI data rate protocol has been extended from 3.125 Gbps up to 3.75 Gbps. For HiGig data rates, select the XAUI protocol and type in the increased data rate.

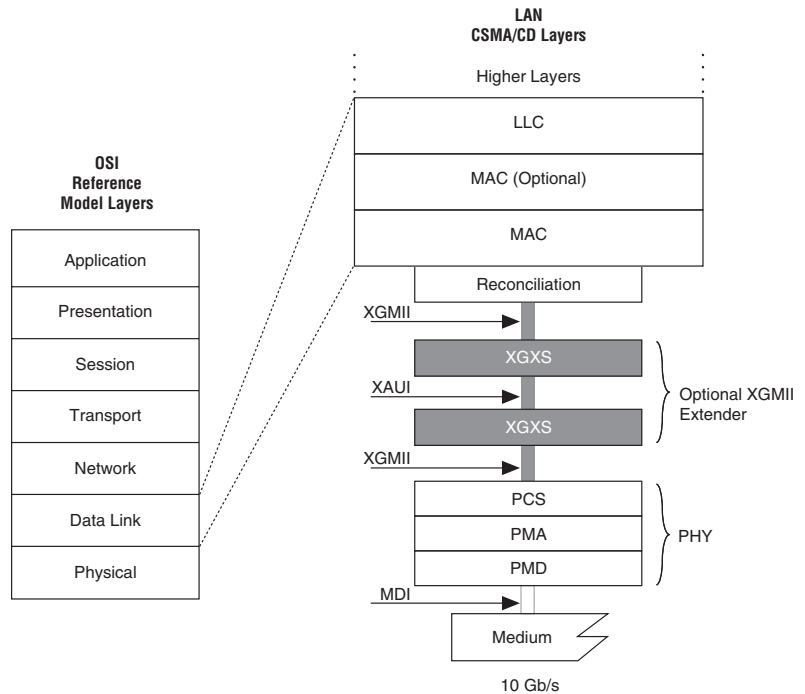
The XAUI standard is an optional self-managed interface that is inserted between the reconciliation sublayer and the PHY layer to transparently extend the physical reach of XGMII.

Figure 2–120. XAUI Mode



XAUI addresses several physical limitations of XGMII. XGMII signaling is based on the HSTL Class I single-ended IO standard, which has an electrical distance limitation of approximately 7 cm. XAUI utilizes a low-voltage differential signaling method, so the electrical limitation is increased to approximately 50 cm. Another advantage of XAUI is the simplification of backplane and board trace routing. XGMII is composed of 32 transmit channels, 32 receive channels, 1 transmit clock, 1 receive clock, 4 transmitter control characters, and 4 receive control characters for a total of a 74-pin wide interface. XAUI consists of 4 differential transmitter channels and 4 differential receiver channels for a total of a 16-pin-wide interface. This reduction in pin count significantly simplifies the routing process in the layout design. Figure 2–121 shows the relationships between the XGMII and XAUI layers.

Figure 2–121. XGMII and XAUI Relationship



Media Access Control (MAC)
 Medium Dependent Interface (MDI)
 Physical Coding Sublayer (PCS)
 Physical Layer Device (PHY)
 Logical Link Control (LLC)

Physical Medium Attachment (PMA)
 Physical Medium Dependent (PMD)
 10 Gigabit Attachment Unit Interface (XAUI)
 10 Gigabit Media Independent Interface (XGMII)
 XGMII Extender Sublayer (XGXS)

The XGMII interface consists of four lanes of 8 bits. At the transmit side of the XAUI interface, the data and control characters are converted within the XGXS into an 8B/10B encoded data stream. Each data stream is then transmitted across a single differential pair running at 3.125 Gbps (3.75 Gbps for HiGig). At the XAUI receiver, the incoming data is decoded and mapped back to the 32 bit XGMII format. This provides a transparent extension of the physical reach of the XGMII and also reduces the interface pin count.

XAUI functions as a self-managed interface because code group synchronization, channel deskew, and clock domain decoupling is handled with no upper layer support requirements. This functionality is

based on the PCS code groups that are used during the IPG time and idle periods. PCS code groups are mapped by the XGXS to XGMII characters specified in [Table 2–38](#).

Table 2–38. XGMII Character to PCS Code-Group Mapping

XGMII TXC	XGMII TXD (1)	PCS Code Group	Description
0	00 through FF	Dxx.y	Normal data transmission
1	07	K28.0, K28.3, or K28.5	Idle in
1	07	K28.5	Idle in T
1	9C	K28.4	Sequence
1	FB	K27.7	Start
1	FD	K29.7	Terminate
1	FE	K30.7	Error
1	Other value		Reserved XGMII character
1	Any other value	K30.7	Deleted XGMII character

Note to Table 2–38:
 (1) Values in TXD column are in hexadecimal.

[Figure 2–122](#) shows an example of the mapping between XGMII characters and the PCS code groups that are used in XAUI. The idle characters are mapped to a pseudo-random sequence of /A/, /R/, and /K/ code groups.

Figure 2–122. XGMII Character to PCS Code-Group Mapping

XGMII																		
T/RxD<7:0>			S	Dp	D	D	D	---	D	D	D	D						
T/RxD<15:8>			Dp	Dp	D	D	D	---	D	D	D	T						
T/RxD<23:16>			Dp	Dp	D	D	D	---	D	D	D							
T/RxD<31:24>			Dp	Dp	D	D	D	---	D	D	D							

PCS																		
Lane 0	K	R	S	Dp	D	D	D	---	D	D	D	D	A	R	R	K	K	R
Lane 1	K	R	Dp	Dp	D	D	D	---	D	D	D	T	A	R	R	K	K	R
Lane 2	K	R	Dp	Dp	D	D	D	---	D	D	D	K	A	R	R	K	K	R
Lane 3	K	R	Dp	Dp	D	D	D	---	D	D	D	K	A	R	R	K	K	R

The PCS code-groups are sent via PCS ordered sets. PCS ordered sets consist of combinations of special and data code groups defined as a column of code groups. These ordered sets are composed of four code groups beginning in lane 0. Table 2–39 lists the defined idle ordered sets (| |I| |) that are used for the self-managed properties of XAUI.

Code	Ordered Set	Number of Code Groups	Encoding
 I 	Idle		Substitute for XGMII Idle
K	Synchronization column	4	/K28.5/K28.5/K28.5/K28.5
R	Skip column	4	/K28.0/K28.0/K28.0/K28.0
A	Align column	4	/K28.3/K28.3/K28.3/K28.3

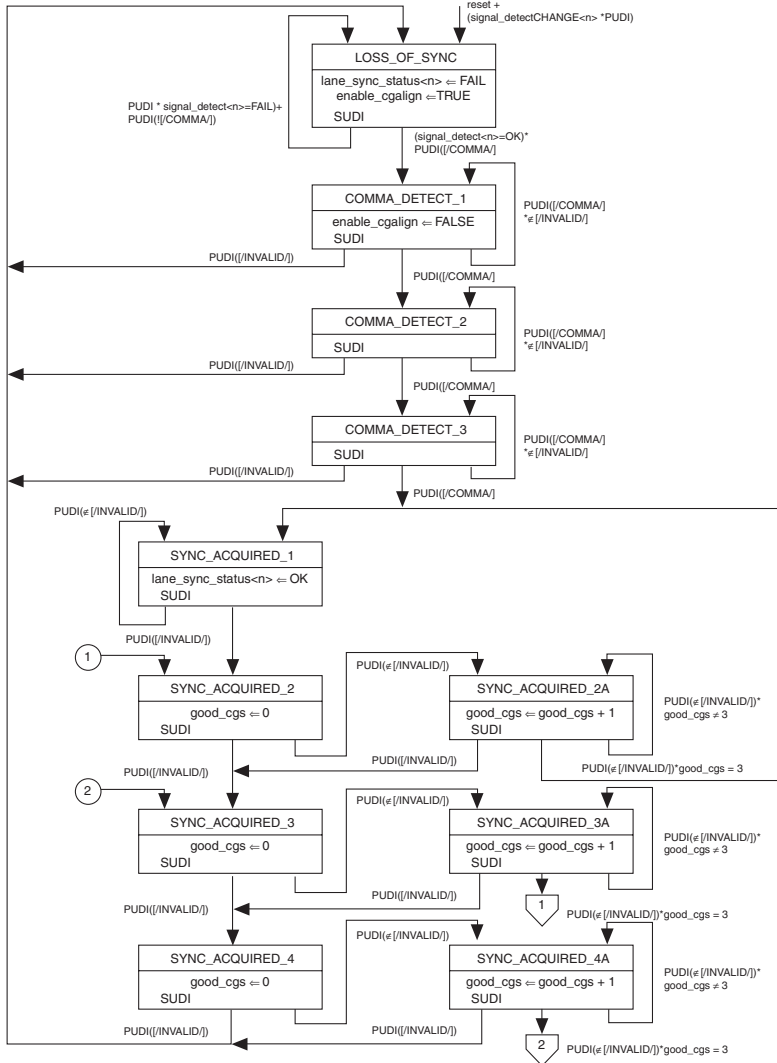
Synchronization ||K|| (Word Aligner)

XAUI uses an embedded clocking scheme that re-times the data that can potentially alter the code-group boundary. The boundaries of the code groups are re-aligned through a synchronization process specified in clause 48 of the IEEE P802.3ae standard, which states that synchronization is achieved upon the reception of four /K28.5/ commas. Each comma can be followed by any number of valid code groups. Invalid code groups are not allowed during the synchronization stage.

When you configure Stratix II GX devices to the XAUI protocol, the built-in pattern detector, word aligner, and XAUI state machines adhere to the PCS synchronization specification. After all the conditions for synchronization have been met, the `rx_syncstatus` signal is asserted and only de-asserts if synchronization is lost.

Figure 2–123 shows the PCS synchronization state diagram specified in clause 48 of the IEEE P802.3ae.

Figure 2–123. IEEE 802.3ae PCS Synchronization State Diagram

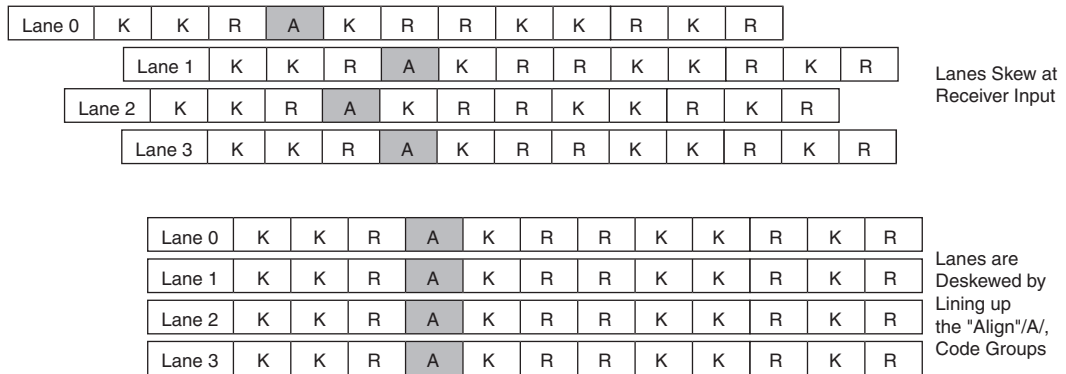


Channel Alignment | | A | | (Deskew)

It is possible for ordered sets to be misaligned with respect to one another because of board skew or differences between the independent clock recoveries per serial lane. Channel alignment, also referred to as deskew or channel bonding, realigns the ordered sets by using the alignment code group, referred to as /A/. The /A/ code group is transmitted simultaneously on all four lanes, constituting an | | A | | ordered set, during idles or IPG. XAUI receivers use these code groups to resolve any lane-to-lane skew. Skew between the lanes can be up to 40 UI (12.8 ns) as specified in the standard, which relaxes the board design constraints.

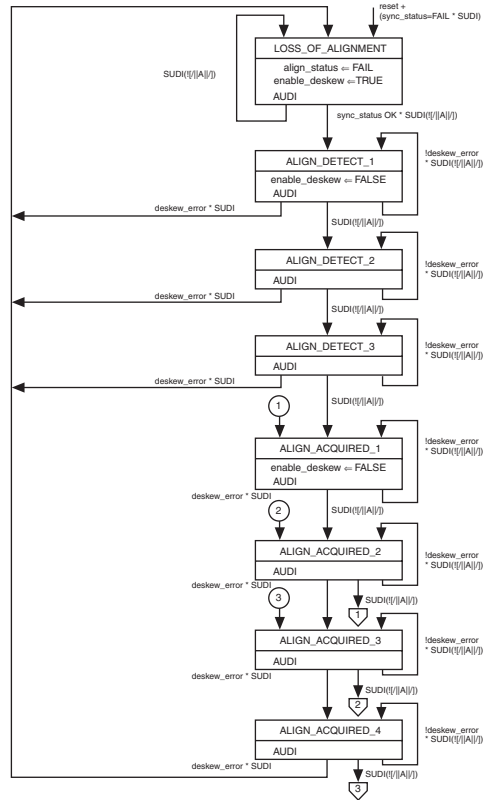
Figure 2–124 shows lane skew at the receiver input and how the deskew circuitry uses the /A/ code group to deskew the channels.

Figure 2–124. Lane Deskew With the /A/ Code Group



Stratix II GX devices manage XAUI channel alignment with a dedicated deskew macro that consists of a 16-word-deep FIFO buffer that is controlled by a XAUI deskew state machine. The XAUI deskew state machine first looks for the /A/ code group within each channel. When the XAUI deskew state machine detects /A/ in each channel, the deskew FIFO buffer is enabled. The deskew state machine now monitors the reception of /A/ code groups. When four aligned /A/ code groups have been received the `rx_channelaligned` is asserted. The deskew state machine continues to monitor the reception of /A/ code groups and de-asserts the `rx_channelaligned` signal if alignment conditions are lost. This built-in deskew macro is only enabled for the XAUI protocol. Figure 2–125 shows the PCS deskew state diagram specified in clause 48 of the IEEE P802.3ae.

Figure 2–125. IEEE 802.3ae PCS Deskew State Diagram




Clock Compensation //R// (Rate Matcher)

XAUI can operate in multi-crystal environments, which can tolerate frequency variations of ± 100 PPM between crystals. Stratix II GX devices contain embedded circuitry to perform clock rate compensation, which is achieved by inserting or removing the PCS SKIP code group (/R/) from the IPG or idle stream. This process is called rate matching and is sometimes referred to as clock rate compensation.

The rate matcher in Stratix II GX devices consists of a 12-word-deep FIFO buffer along with control logic that you can configure to support XAUI, GIGE, or custom modes. In XAUI mode the controller begins to write data into the FIFO buffer whenever the `rx_channel_aligned` signal is asserted. Within the control logic there is a FIFO counter that keeps track of the read and write executions. When the FIFO counter reaches a value of greater than nine, the receivers delete the /R/ code-group

simultaneously across all channels during IPG or idle conditions. If the FIFO counter is less than five, the receivers insert the /R/ code-group simultaneously across all channels during IPG or idle conditions.

 This circuitry compensates for ± 100 PPM frequency variations.

PCS Code Group to XGMII Character Mapping

In XAUI mode, the 8B/10B decoder in Stratix II GX devices is controlled by a global receiver state machine that maps various PCS code groups into specific 8-bit XGMII codes. [Table 2-40](#) lists the PCS code group to XGMII character mapping.

XGMII RXC	XGMII RXD (1)	PCS Code Group	Description
0	00 through FF	Dxx.y	Normal data transmission
1	07	K28.0, K28.3, or K28.5	Idle in I
1	07	K28.5	Idle in T
1	9C	K28.4	Sequence
1	FB	K27.7	Start
1	FD	K29.7	Terminate
1	FE	K30.7	Error
1	FE	Invalid code group	Received code group

Note to [Table 2-40](#):

(1) Values in RXD column are in hexadecimal.

XGMII Character to PCS Code-Group Mapping

In XAUI mode, the 8B/10B encoder in Stratix II GX devices is controlled by a global transmitter state machine that maps various 8-bit XGMII codes to 10-bit PCS code groups. This state machine complies with the IEEE 802.3ae PCS transmit specification. [Figure 2-126](#) shows the PCS transmit source state diagram specified in clause 48 of the IEEE P802.3ae.

Figure 2–126. IEEE 802.3ae PCS Transmit Source State Diagram

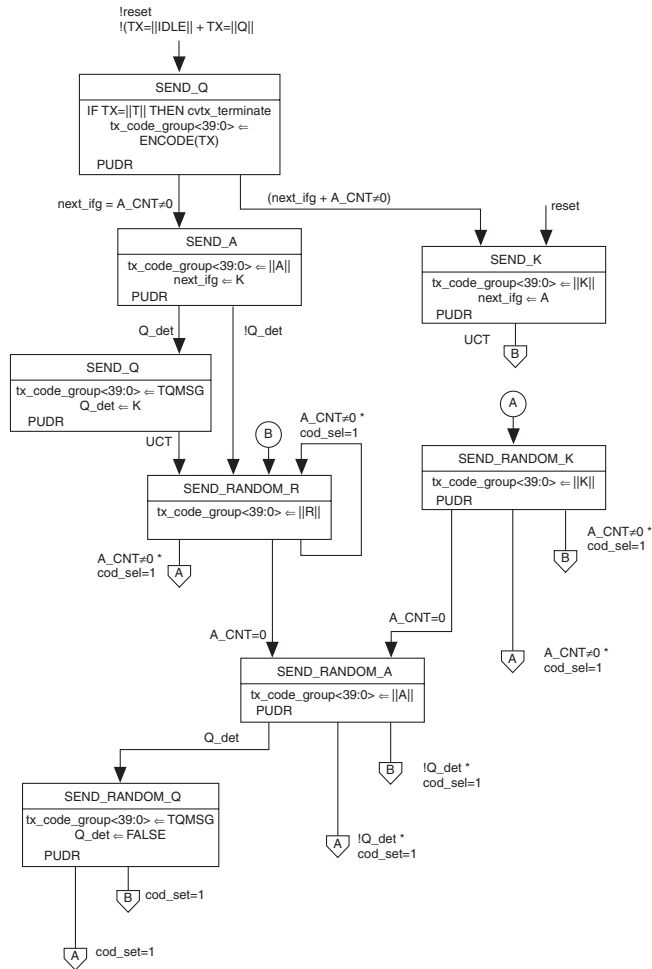


Table 2–41 lists the XGMII character to PCS code-group mapping.

XGMII TXC	XGMII TXD (1)	PCS Code Group	Description
0	00 through FF	Dxx.y	Normal data transmission
1	07	K28.0, K28.3, or K28.5	Idle in I
1	07	K28.5	Idle in T
1	9C	K28.4	Sequence
1	FB	K27.7	Start
1	FD	K29.7	Terminate
1	FE	K30.7	Error
1	Other value		Reserved XGMII character
1	Any other value	K30.7	Invalid XGMII character

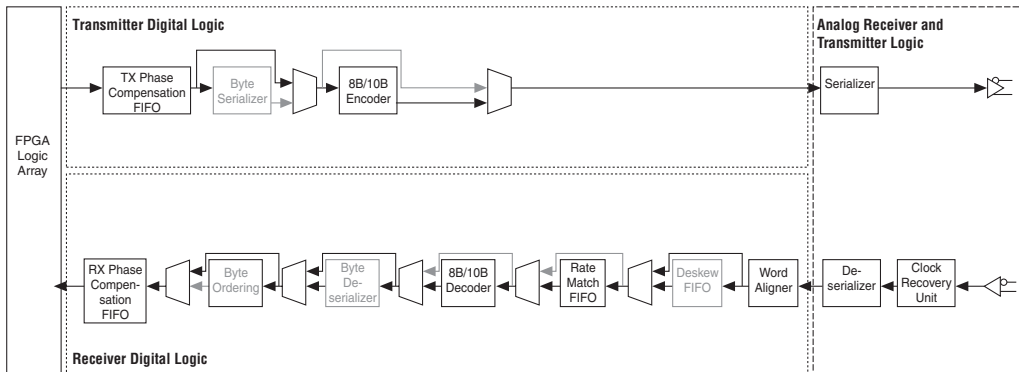
Note to Table 2–41:

(1) Values in TXD column are in hexadecimal.

GIGE Mode

The Gigabit Media Independent Interface (GMII) is an intermediate, or transition, layer that interfaces various mediums with the media access control (MAC) in a GIGE system (refer to Figure 2–127). The GMII is divided into three sublayers: the PCS, the PMA, and the physical medium dependent (PMD) layers. GMII offers data rates up to 1000 Mbps at either half- or full-duplex modes.

Figure 2–127. GIGE Mode



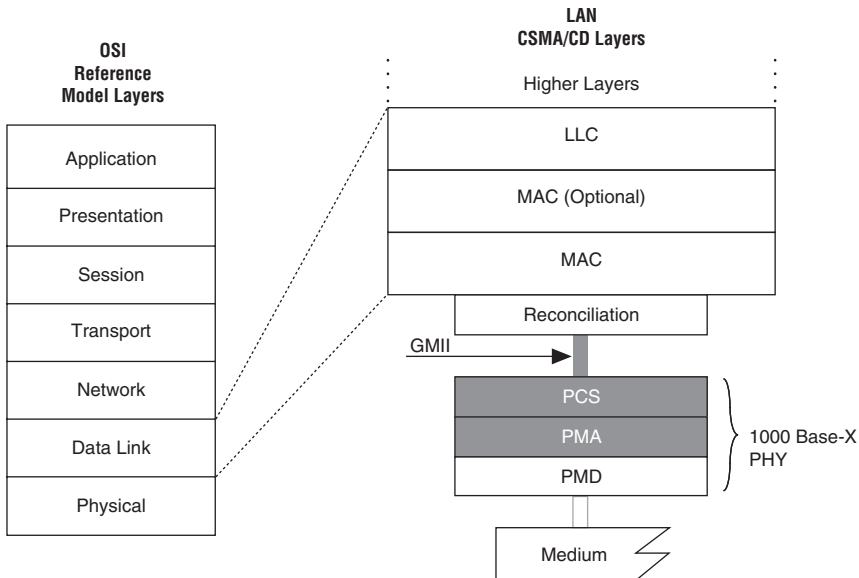
The PCS sublayer provides synchronization, encoding and decoding, and rate matching services to the MAC. The PCS also provides auto negotiation to the network to negotiate speeds, carrier, and collision detect signals.

The PMA sublayer provides the PCS with a media-independent interface that a variety of serial physical media can be connected to. This layer handles the serialization and deserialization of the data.

The PMD sublayer defines actual physical attachment, such as connectors for different media types.

Figure 2–128 shows the GMII position relative to the OSI reference model.

Figure 2–128. GMII Position Relative to the OSI Reference Model



The Stratix II GX transceiver can be used for the PCS and the PMA layers of the GMII. Stratix II GX devices in GIGE mode use the 8B/10B encoder/decoder, rate matcher, and synchronizer built-in hard macros. The rate matcher and synchronizer have a dedicated state machine governing their functions. This state machine is only active in GIGE mode.

Table 2–42 shows the code groups used in the GIGE protocol. If required for your design, you must implement the remaining functions of the PCS—auto negotiation, collision detect, and carrier detect—in user logic or external circuits.

Table 2–42. GIGE Code Groups (Part 1 of 2)

Code	Ordered Set	Number of Code Groups	Encoding
/C/	Configuration		Alternating /C1/ and /C2/
/C1/	Configuration 1	4	/K28.5/D21.5/Config_Reg (1)

Table 2–42. GIGE Code Groups (Part 2 of 2)

Code	Ordered Set	Number of Code Groups	Encoding
/C2/	Configuration 2	4	/K28.5/D2.2/Config_Reg (1)
//	IDLE		Correcting /I1/, Preserving /I2/
/I1/	IDLE 1	2	/K28.5/D5.6
/I2/	IDLE 2	2	/K28.5/D16.2
	Encapsulation		
/R/	Carrier_Extend	1	/K23.7/
/S/	Start_of_Packet	1	/K27.7/
/T/	End_of_Packet	1	/K29.7/
/V/	Error_Propagation	1	/K30.7/

Note to Table 2–42:

(1) Two data code groups representing the Config_Reg value.

Synchronization (Word Aligner)

Synchronization is required in GIGE mode to align the byte boundary of the receiver to the byte boundary of the transmitter, because the Stratix II GX transceiver block uses a non-source-synchronous serial stream. To correctly align the byte boundary at the receiver, a unique synchronization pattern must be received that does not occur between any Dx.y and/or Kx.y code combinations. A /K28.5/ 10-bit comma is used for this purpose.

Synchronization is achieved when the receiver sees three consecutive ordered sets. An ordered set defined for synchronization is a /K28.5/ comma followed by any odd number of valid /Dx.y/ code (/Dx.y/ denotes any valid data code group). Although you can have a number of synchronization patterns based on the synchronization rule, three /K28.5/ followed by one /Dx.y/ code is the fastest synchronization pattern.

Once the synchronization is achieved, the state machine considers a bad code group received (cgbad) if one of the following two conditions are met:

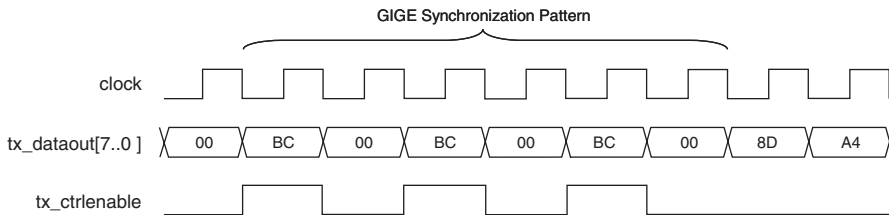
- The incoming code group has a disparity error or a code group violation
- The incoming code group is a comma character and `rx_even=true`. This condition occurs when an even number of non-comma code groups are received between two comma code groups.

GIGE Transmitter Synchronization

The transmitter must send out the GIGE synchronization sequence to synchronize the target receiver. Stratix II GX devices do not contain a built-in macro that provides this function upon power-up or reset. You must implement this function in user logic to send out a `/K28.5/, /Dx.y/, /K28.5/, /Dx.y/` sequence.

Figure 2–130 shows an example of the GIGE synchronization pattern. Although the example shows one `D0.0` (8'h00) as the `/Dx.y/` code, any `/Dx.y/` and any odd number of `/Dx.y/` can be used.

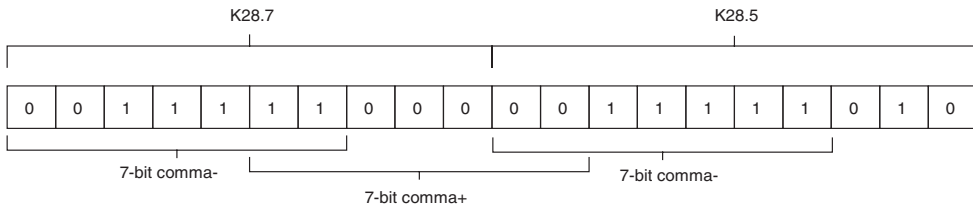
Figure 2–130. GIGE Synchronization Transmit Pattern



GIGE Receiver Synchronization

You must pre configure the receiver with a `K28.5` (10'b0101111100 or 10'b1010000011) word align pattern (`ALIGN_PATTERN = 0101111100` or `ALIGN_PATTERN = 1010000011`). The `ALIGN_PATTERN_LENGTH` must be set to 10 even though a 7-bit comma string (7'b00111111 as a comma- or 7'b11000000 as a comma+) is allowed as specified in IEEE 802.3. This 7-bit comma is located within the `/K28.1/, /K28.5/, and /K28.7/` code groups. Using a 10-bit `/K28.5/` helps prevent a 7-bit comma from being detected across boundaries when a `/K28.7/` code is followed by a `/K28.x/, /D3.x/, /D3.x/, /D11.x/, /D12.x/, /D19.x/, /D20.x/, or /D28.x/`, where `x` is a value from 0 to 7 (Figure 2–131).

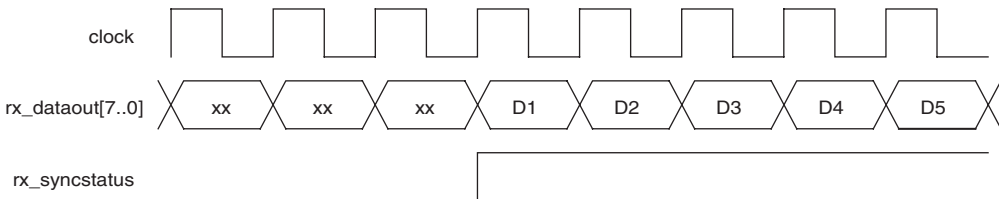
Figure 2–131. Cross Boundary 7-bit Comma When a /K28.7/ is Followed by a /K28.5/ Code Group



The receiver outputs a K28.4 (8'h9c + rx_ctrldetect) at the rx_dataout port and de-asserts the rx_syncstatus (1'b0) signal when the receiver is not synchronized. Once synchronized, the receiver asserts the rx_syncstatus signal (1'b1). This signal is aligned with the first valid data received from the rx_dataout port.

Figure 2-132 shows the receiver synchronization waveform. The rx_syncstatus port goes high when synchronization is complete, indicating that the data is valid. In Figure 2-132, D1 is the first valid data.

Figure 2-132. Synchronization Complete



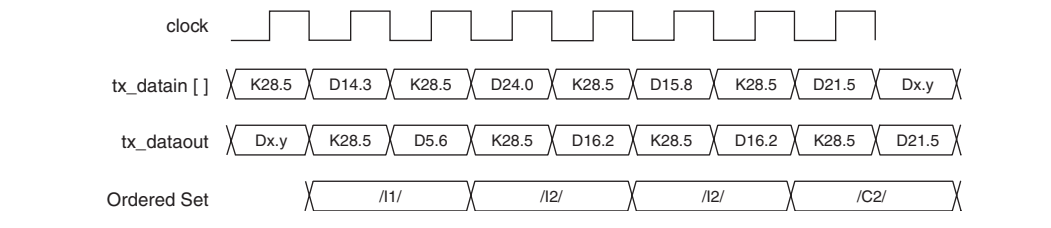
The receiver remains synchronized until it detects a string of bad code groups or is reset. A bad code group is defined by the IEEE 802.3 standard as four invalid code groups separated by less than three valid code groups. If the receiver detects a bad code group or is reset, the rx_syncstatus signal goes high, then low and a /K28.4/ appears on the rx_outrx_dataout port.

Idle Generation

In GIGE mode, any /Dx.y/ following a /K28.5/ comma is replaced by the transmitter with either a /D5.6/ (8'hc5) or a /D16.2/ (8'h50) depending on the current running disparity, except when the data following the /K28.5/ is /D21.5/ (8'hb5) or /D2.2/ (8'h42). This ensures the generation of the /I1/ (/K28.5/, /D5.6/) and /I2/ (/K28.5/, /D16.2/) ordered sets and to allows the configuration ordered sets /C1/ (/K28.5/, /D21.5/) and /C2/ (/K28.5/, /D2.2/) to be received. If the running disparity before the idle ordered set is positive, a /I1/ is chosen. If the running disparity is negative, an /I2/ is chosen. The disparity at the end of an /I1/ is the opposite of that at the beginning of the /I1/. The disparity at the end of an /I2/ is the same as the beginning running disparity (right before the idle code). This ensures a negative running disparity at the end of an idle ordered set. A /Kx.y/ following a /K28.5/ is not replaced.

Figure 2–133 shows the input data codes versus the output data codes. /D14.3/, /D24.0/, and /D15.8/ were replaced by /D5.6/ or /D16.2/ (for /I1/, /I2/ ordered sets) and /D21.5/ (part of the /C2/ order set) was not replaced.

Figure 2–133. Example of the Input Data Codes Versus the Output Data Codes



Rate Matching (Rate Matcher)

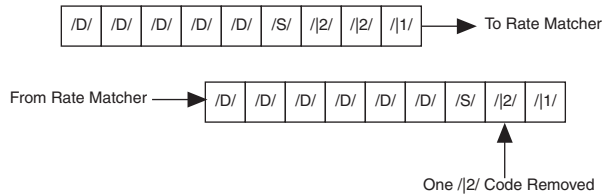
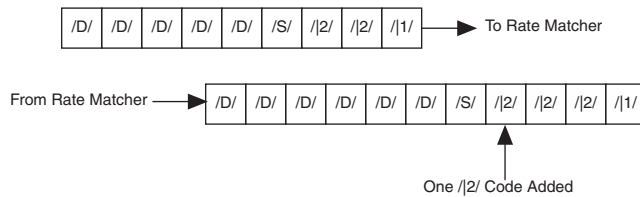
GIGE can operate in a multi crystal environment, so rate matching is necessary to compensate for the frequency variations from different crystals. Stratix II GX devices contain a built-in rate matcher (12-word-deep FIFO buffer with control logic) that can tolerate up to, and compensate for, a ± 100 PPM frequency variation.

In the GIGE mode, rate matching occurs automatically in the rate matcher. If the GMII protocol is followed, the /I/ sets (/I1/, /I2/) are sent during the inter-frame gap (IFG). (The GMII protocol specifies 96-bits long). The /I2/ ordered set (/K28.5/, /D16.2/) is added or deleted based on how full or empty the rate matcher FIFO buffer is and if the current running disparity is negative. The /I2/ order set contains two 10-bit code groups. Two 10-bit groups (20-bits total) are deleted or added at a time. If the number of words in the FIFO buffer (FIFO count) is greater than nine, the FIFO buffer stops writing when the /I2/ ordered set is detected (Figure 2–134). If the FIFO count is less than five, the FIFO buffer stops reading and inserts the /I2/ ordered sets in place of the next FIFO data (Figure 2–135).



The GIGE rate matcher does not have the capability of inserting or deleting /C1/ or /C2/ ordered sets.

If the rate matching FIFO buffer is in an underflow or overflow condition (empty or full), the receiver outputs a /K28.4/ (8'h9C + ctrl). This might happen if the PPM (parts per million) difference in the read and write clock is too great, the IFG or IPG is too small (there are not enough /I2/ code groups to remove), and/or the frame or packet size is too big.

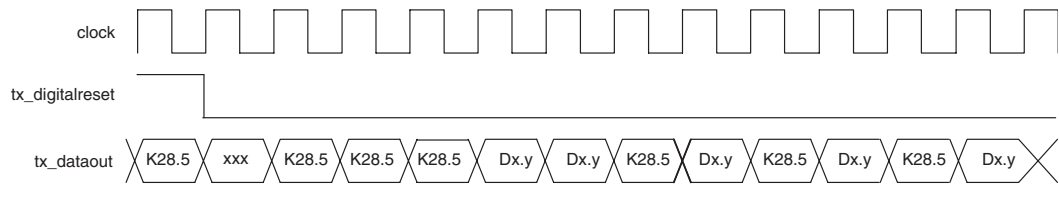
Figure 2–134. /I2/ Deleted When FIFO Count is Greater Than Nine**Figure 2–135. /I2/ Added When FIFO Count is Less Than Five**

Reset Condition

After power up or reset, the GIGE transmitter outputs three /K28.5/ commas before user data can be sent. This affects the synchronization ordered set transmission.

After reset (`tx_digitalreset`), the 8B/10B encoder automatically sends three /K28.5/ commas (refer to “8B/10B Decoder” on page 2–103 for additional information). Depending on when you start outputting the synchronization sequence, there could be an even or odd number of /Dx.y/ sent as the transmitter before the synchronization sequence. The last of the three automatically sent /K28.5/ and the first user-sent /Dx.y/ are treated as one idle ordered set. This can be a problem if there are an even number of /Dx.y/ transmitted before the start of the synchronization sequence.

Figure 2–136 shows an example of even numbers of /Dx.y/ between the last automatically sent /K28.5/ and the first user-sent /K28.5/. The first user-sent ordered set is ignored, so three additional ordered sets are required for proper synchronization. Figure 2–136 shows one don’t care data between the `tx_digitalreset` signal going low and the first of three automatic K28.5, but there could be more.

Figure 2–136. GIGE Synchronization Ordered Set Considerations After Reset

SONET/SDH Mode

SONET/SDH is one of the most common serial-interconnect protocols used in backplanes deployed in communications and telecom applications. SONET/SDH defines various optical carrier (OC) subprotocols for carrying signals of different capacities through a synchronous optical hierarchy.

Stratix II GX transceivers can be employed as physical layer devices in a SONET/SDH system. These transceivers provide support for SONET/SDH protocol-specific functions and electrical features; for example, alignment to A1A2 or A1A1A2A2 pattern.

Stratix II GX transceivers are designed to support the following three SONET/SDH subprotocols:

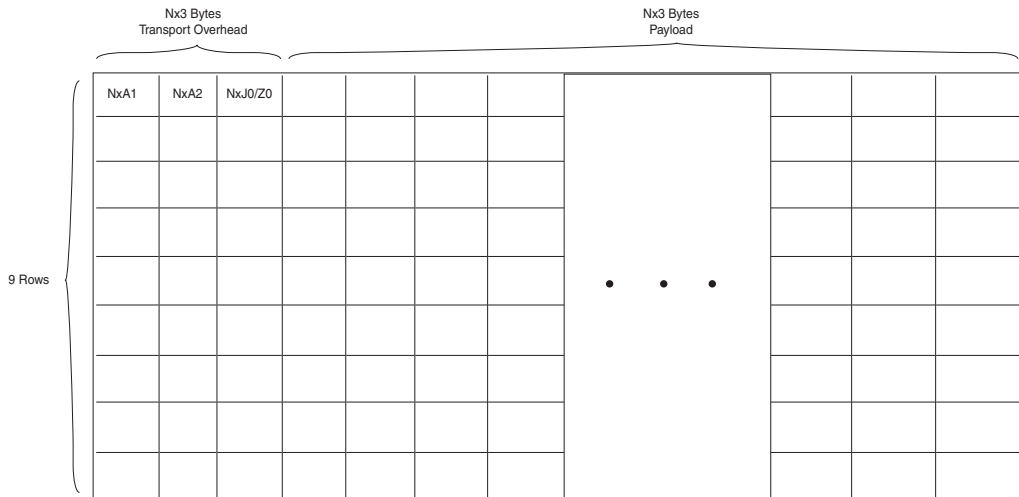
- OC-12 at 622 Mbps with 8-bit channel width
- OC-48 at 2488.32 Mbps with 16-bit channel width
- OC-96 at 4976 Mbps with 32-bit channel width

SONET/SDH Frame Structure

Base OC-1 frames are byte-interleaved to form SONET/SDH frames. For example, twelve OC-1 frames are byte-interleaved to form one OC-12 frame; forty-eight OC-1 frames are byte-interleaved to form one OC-48 frame and so on. SONET/SDH frame sizes are constant, with a frame transfer rate of 125 μ s.

Figure 2–137 shows the SONET/SDH frame structure.

Figure 2–137. SONET/SDH Frame Structure



Note to Figure 2–137

(1) N=12 for OC-12, 48 for OC-48, and 96 for OC-96.

Transport overhead bytes A1 and A2 are used for recovering the frame boundary from the serial data stream. Frame sizes are fixed, so the A1 and A2 bytes appear within the serial data stream every 125 μ s. In an OC-12 backplane system, twelve A1 bytes are followed by twelve A2 bytes. Similarly, in an OC-48 backplane system, forty-eight A1 bytes are followed by forty-eight A2 bytes.

In SONET/SDH systems, byte values of A1 and A2 are fixed as follows:

- A1 = "11110110" or 8'hF6
- A2 = "00101000" or 8'h28

OC-12 and OC-48 Data Paths

OC-12 and OC-48 configurations have similar data paths, as seen in Figures 2–138 and 2–139. The only difference is that OC-48 has a 16-bit PLD interface as compared to the 8-bit PLD interface in an OC-12 configuration. The OC-48 configuration employs the byte serializer and deserializer and a byte ordering block to translate 16-bit PLD interfaces into an 8-bit transceiver data path.

Figure 2–138. OC-12 Data Path

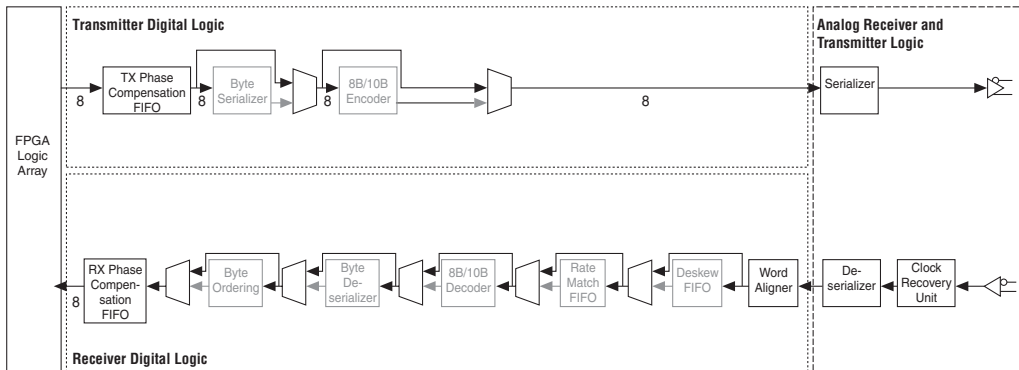
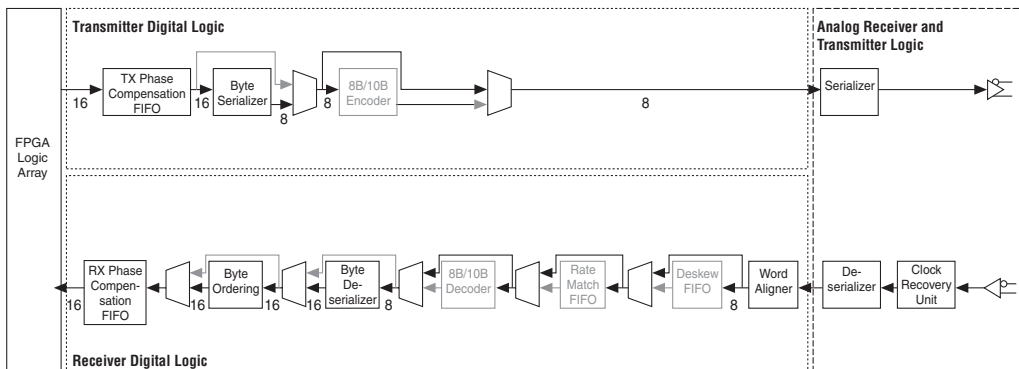


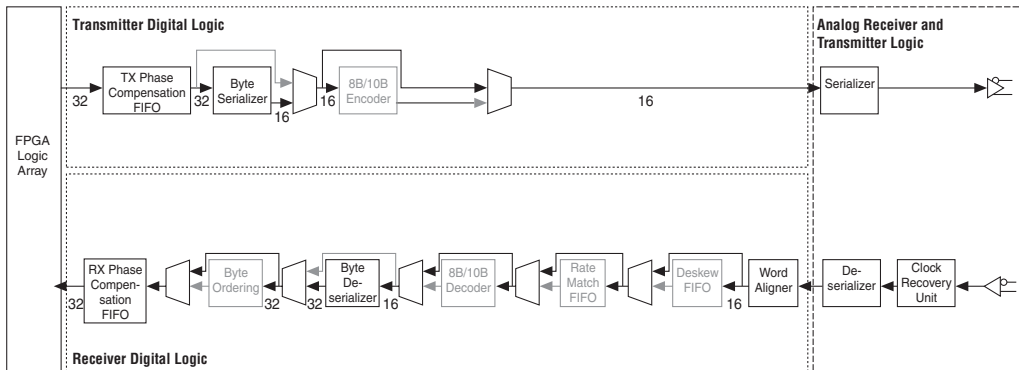
Figure 2–139. OC-48 Data Path



OC-96 Data Path

The OC-96 data path is wider than the OC-12 and OC-48 data paths (refer to Figure 2–140). It has a 32-bit wide PLD interface that is translated to a 16-bit wide transceiver data path by the byte serializer and deserializer. As a result, the OC-96 configuration has a bit serialization factor of 16, unlike OC-12 and OC-48 with bit serialization factors of 8. Also, the OC-96 configuration does not have the byte ordering block in the transceiver data path. If required, you should implement byte ordering logic in the PLD logic array in OC-96 configurations.

Figure 2–140. OC-96 Data Path



SONET/SDH Serial Data Transmission Bit Order

Unlike Ethernet where the least significant bit of the data byte is transferred first, SONET/SDH requires the most significant bit to be transferred first and the least significant bit to be transferred last. To facilitate MSBit to LSBit transfer, you must enable the following options in the MegaWizard:

- Flip Transmitter Input Data Bits (when used in transmit only or duplex mode)
- Flip Receiver Output Data Bits (when used in receive only or duplex mode)

Depending on whether data bytes are transferred MSBit to LSBit or LSBit to MSBit, you must select appropriate word aligner settings in the MegaWizard. Table 2–43 lists correct word aligner settings for each bit transmission order.

OC-12 and OC-48 Word Alignment

SONET/SDH mode uses manual word alignment as described in “Manual SONET/SDH Alignment Mode (Two Consecutive 8-bit Characters (A1A2) or Four Consecutive 8-bit Characters (A1A1A2A2))” on page 2–78.

In OC-12 and OC-48 configurations, you can configure the word aligner to either align to a 16-bit A1A2 pattern or a 32-bit A1A1A2A2 pattern. This is controlled by the `rx_a1a2size` input port to the transceiver. A

LOW level on the `rx_a1a2size` port configures the word aligner to align to a 16-bit A1A2 pattern and a high level configures it to align to a 32-bit A1A1A2A2 pattern.

You can configure the word aligner to flip the alignment pattern bits programmed in the MegaWizard and compare it them with the incoming data for alignment. This feature offers flexibility to the SONET/SDH system for either an MSBit to LSBit or LSBit to MSBit data transfer. [Table 2-43](#) lists word alignment patterns that you must program in the MegaWizard based on the bit-transmission order and the word aligner bit-flip option.

Serial Bit Transmission Order	Word Alignment Bit Flip	Word Alignment Pattern
MSBit to LSBit	On	1111011000101000 (16'hF628)
MSBit to LSBit	Off	0001010001101111 (16'h146F)
LSBit to MSBit	Off	0010100011110110 (16'h28F6)

The behavior of the SONET/SDH word aligner control and status signals along with an operational timing diagram are explained in [“Manual SONET/SDH Alignment Mode \(Two Consecutive 8-bit Characters \(A1A2\) or Four Consecutive 8-bit Characters \(A1A1A2A2\)\)”](#) on page 2-78.

OC-96 Word Alignment

In OC-96 configuration, the word aligner is only allowed to align to a A1A1A2A2 pattern, so input port `rx_a1a2size` is unavailable. Barring this difference, the OC-96 word alignment operation is similar to that of the OC-12 and OC-48 configurations.

OC-48 Byte Serializer and Deserializer

The OC-48 transceiver data path includes the byte serializer and deserializer to allow the PLD interface to run at a lower speed. The OC-12 configuration does not use the byte serializer and deserializer blocks.

The byte serializer and deserializer blocks are explained in the sections [“Byte Serializer”](#) on page 2-32 and [“Byte Deserializer”](#) on page 2-112, respectively. The OC-48 byte serializer converts 16-bit data words from

the PLD logic array and translate the 16-bit data words into two 8-bit data bytes at twice the rate. The OC-48 byte deserializer takes in two consecutive 8-bit data bytes and translates them into a 16-bit data word to the PLD logic array at half the rate.

OC-96 Byte Serializer and Deserializer

The OC-96 byte serializer converts 32-bit data words from the PLD logic array and translates them into two 16-bit data bytes at twice the rate. The OC-48 byte deserializer takes in two consecutive 16-bit data bytes and translates them into a 32-bit data word to the PLD logic array at half the rate.

OC-48 Byte Ordering

Because of byte deserialization, the most significant byte of a word may appear at the `rx_dataout` port along with the least significant byte of the next word.

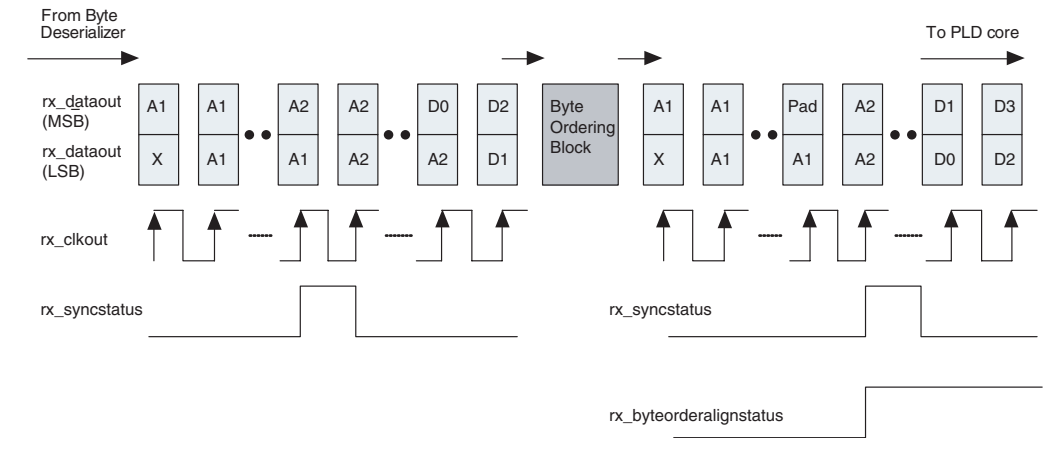
In an OC-48 configuration, the byte ordering block is built into the data path and can be leveraged to perform byte ordering. In an OC-96 configuration, the byte ordering block is unavailable and ordering must be performed in the PLD logic array.

The byte ordering in an OC-48 configuration is automatic as explained in [“Word Alignment Based on Byte Ordering” on page 2–114](#). In automatic mode, the byte ordering block is triggered by the rising edge of the `rx_syncstatus` signal. As soon as the byte ordering block sees the rising edge of the `rx_syncstatus` signal, it compares the least significant byte coming out of the byte deserializer with the A2 byte of the A1A2 alignment pattern. If the least significant byte coming out of the byte deserializer does not match A2 byte set in the MegaWizard, the byte ordering block inserts a pad character as seen in [Figure 2–141](#). Insertion of this pad character enables the byte ordering block to restore the correct byte order. Note that the pad character is defaulted to the A1 byte of the A1A2 alignment pattern.

Once the byte ordering is achieved, the `rx_byteorderalignstatus` signal remains asserted high until `rx_digitalreset` is asserted. The byte ordering within the transceiver is a one-time event after the receiver comes out of `rx_digitalreset`. So, if a byte ordering operation is required, the receiver must go through an `rx_digitalreset` cycle.

If successful byte ordering occurs without successful word alignment, you should assert receive digital reset (`rx_digitalreset`) so that the byte ordering block performs another round of byte ordering (one time after asserting `rx_digitalreset`). This is only required when the byte ordering block picks an incorrect byte order.

Figure 2–141. Byte Ordering Block Operation in OC-48



(OIF) CEI-PHY Interface Mode

The (OIF) CEI PHY Interface mode is intended to support two main protocols:

- Common Electrical I/O (CEI-6G) protocol defined by the Optical Internetworking Forum (OIF) at data rates between 4.976 Gbps and 6.375 Gbps
- Interlaken protocol at data rates between 3.135 Gbps and 6.375 Gbps

Stratix II GX transceivers support a data rate between 3.135 Gbps and 6.375 Gbps in (OIF) CEI PHY Interface Mode.

Figure 2–142 shows the ALT2GXB transceiver data path when configured in this mode.

Figure 2–142. (OIF) CEI-PHY Interface Data Path

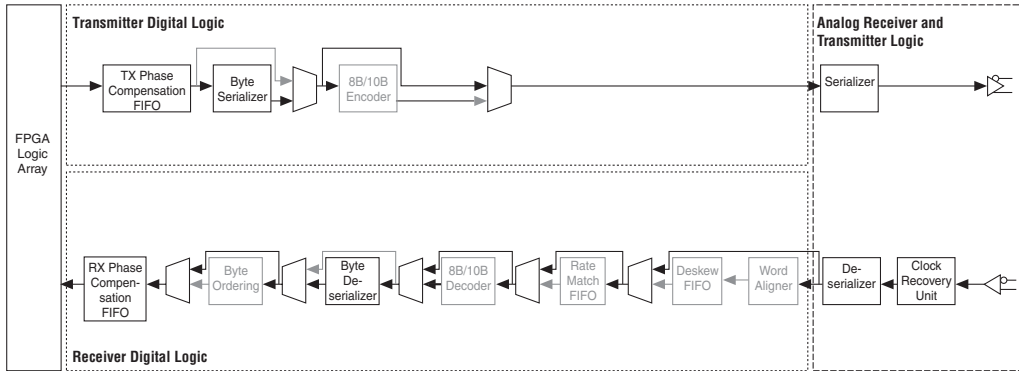


Table 2–44 shows ALT2GXB configurations supported by the Stratix II GX transceivers in (OIF) CEI PHY Interface mode.

Data Rate (Mbps)	REFCLK Frequency (PLL Multiplication Factor)	Channel Width
3135 < Data Rate ≤ 5700	Data-Rate/10 (M = 10) (1) Data-Rate/20 (M = 10) Data-Rate/40 (M = 20)	32 bit
5800 < Data Rate ≤ 6375	Data-Rate/10 (M = 10) (1) Data-Rate/20 (M = 10)	32 bit

Note to Table 2–44:

- (1) Selecting the REFCLK frequency of Data-Rate/10 requires the use of the /2 REFCLK pre-divider; for example, selecting 500 MHz REFCLK frequency for 5000 Mbps data rate.

(OIF) CEI PHY Interface Mode Clocking

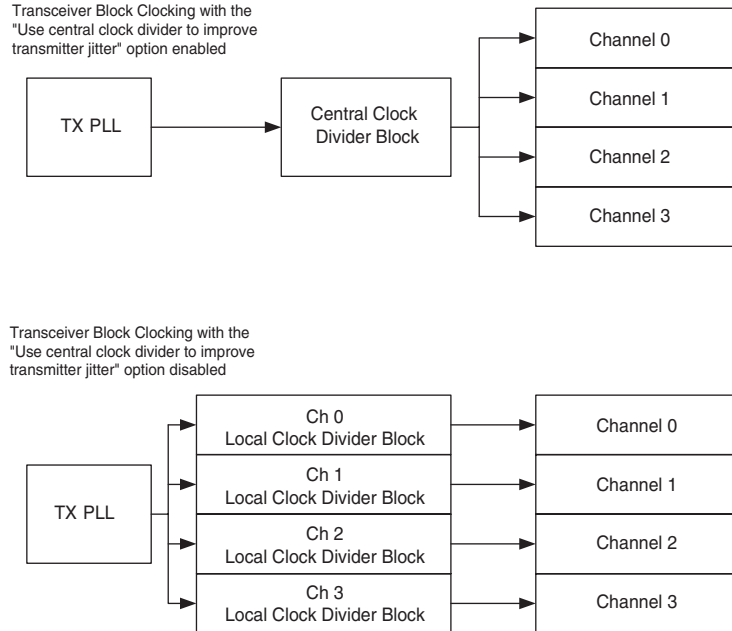
For improved transmitter jitter performance, the ALT2GXB MegaWizard Plug-In Manager provides an **Use central clock divider to improve transmitter jitter** option. If you select this option, clocks generated by the central clock divider clock all four transceiver channels within the same transceiver block. Otherwise, clocks generated by the local clock divider in each channel clock the respective channel.



Unlike PIPE x4, XAUI or Basic x4 mode, the transmitter PCS is not bonded in the (OIF) CEI PHY Interface with the low-jitter option selected.

Figure 2–143 shows transceiver clocking in (OIF) CEI PHY Interface mode with and without the improved transmitter jitter option enabled.

Figure 2–143. (OIF) CEI PHY Interface Mode Clocking



Transceiver Placement Limitations with Improved Jitter Clocking Option

If one or more channels in a transceiver block are configured to (OIF) CEI PHY Interface mode with the improved jitter clocking option enabled, the remaining channels in that transceiver block must either be configured in (OIF) CEI PHY Interface mode with this option enabled or must be unused. All used channels within a transceiver block configured in (OIF) CEI PHY Interface mode with improved jitter clocking option enabled must also run at the same data rate.

Figures 2–144 and 2–145 show two examples each of legal and illegal transceiver placements with respect to the improved jitter clocking option in (OIF) CEI PHY Interface mode.

Figure 2–144. Examples of Legal Transceiver Placement in (OIF) CEI PHY Interface Mode

Ch 0	(OIF) CEI PHY Interface Mode with the low-jitter option enabled (Data Rate = 5 Gbps)	Ch 0	(OIF) CEI PHY Interface Mode with the low-jitter option disabled
Ch 1	(OIF) CEI PHY Interface Mode with the low-jitter option enabled (Data Rate = 5 Gbps)	Ch 1	(OIF) CEI PHY Interface Mode with the low-jitter option disabled
Ch 2	Unused Channel	Ch 2	Serial RapidIO
Ch 3	Unused Channel	Ch 3	Serial RapidIO

Figure 2–145. Examples of Illegal Transceiver Placement in (OIF) CEI PHY Interface Mode

Ch 0	(OIF) CEI PHY Interface Mode with the low-jitter option enabled	Ch 0	(OIF) CEI PHY Interface Mode with the low-jitter option enabled (Data Rate = 5 Gbps)
Ch 1	(OIF) CEI PHY Interface Mode with the low-jitter option enabled	Ch 1	(OIF) CEI PHY Interface Mode with the low-jitter option enabled (Data Rate = 5 Gbps)
Ch 2	Serial RapidIO	Ch 2	(OIF) CEI PHY Interface Mode with the low-jitter option enabled (Data Rate = 6 Gbps)
Ch 3	Serial RapidIO	Ch 3	(OIF) CEI PHY Interface Mode with the low-jitter option enabled (Data Rate = 6 Gbps)

Serial Digital Interface (SDI) Mode

The Society of Motion Picture and Television Engineers (SMPTE) defines various Serial Digital Interface (SDI) standards for transmission of uncompressed video.

The following three SMPTE standards are popular in video broadcasting applications:

- SMPTE 259M standard—more popularly known as the standard-definition (SD) SDI, is defined to carry video data at 270 Mbps.

- SMPTE 292M standard—more popularly known as the high-definition (HD) SDI, is defined to carry video data at either 1485 Mbps or 1483.5 Mbps.
- SMPTE 424M standard—more popularly known as the third-generation (3G) SDI, is defined to carry video data at either 2970 Mbps or 2967 Mbps.

You can configure Stratix II GX transceivers in HD-SDI or 3G-SDI configuration using the ALT2GXB MegaWizard Plug-In Manager.

Figure 2–146 shows the ALT2GXB transceiver data path in SDI mode.

Figure 2–146. SDI Mode Data Path

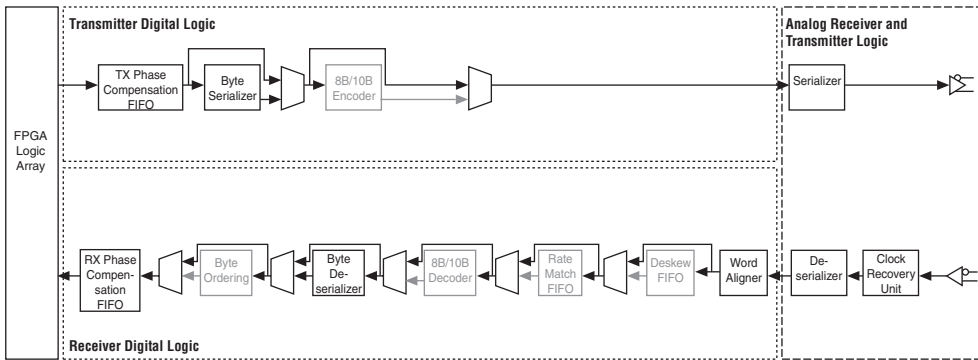



Table 2–45 shows ALT2GXB configurations supported by the Stratix II GX transceivers in SDI mode.

Configuration	Data Rate (Mbps)	REFCLK Frequencies (MHz)	Channel Width
HD	1485	74.25, 148.5	10 bit, 20 bit
	1483.5	74.175, 148.35	10 bit, 20 bit
3G	2970	148.5, 297	Only 20-bit interface allowed in 3G
	2967	148.35, 296.7	Only 20-bit interface allowed in 3G


Transmitter Data Path

In the 10-bit channel width SDI configuration, the transmitter data path is made up of the transmitter phase compensation FIFO and the 10:1 serializer. In the 20-bit channel width SDI configuration, the transmitter data path also includes the byte serializer.

 In SDI mode, the transmitter is purely a parallel-to-serial converter. SDI transmitter functions, such as scrambling and cyclic redundancy check (CRC) code generation, must be implemented in the FPGA logic array.


Receiver Data Path

In the 10-bit channel width SDI configuration, the receiver data path comprises of the clock recovery unit (CRU), the 1:10 deserializer, the word aligner in bit-slip mode, and the receiver phase compensation FIFO. In the 20-bit channel width SDI configuration, the receiver data path also includes the byte deserializer.

 SDI receiver functions, such as de-scrambling, framing, and CRC checker, must be implemented in the FPGA logic array.

Receiver Word Alignment/Framing

In SDI systems, since the word alignment and framing happens after de-scrambling, the word aligner in the receiver data path is not useful. Altera recommends driving the ALT2GXB `rx_bitslip` signal low to avoid the word aligner from inserting bits in the received data stream.

 Altera offers SDI MegaCore function that can be configured at SD-SDI, HD-SDI, and 3G-SDI data rates. The SDI MegaCore function implements system-level functions like scrambling and de-scrambling and CRC generation and checking. It also offers the capability of configuring the three SDI data rates (SD, HD, and 3G) dynamically on the same transceiver channel. For more details, refer the [SDI MegaCore Function User Guide](#).

Serial RapidIO Mode

The RapidIO™ Trade Association defines a high-performance, packet-switched interconnect standard to pass data and control information between microprocessors, digital signal, communications, and network processors, system memories, and peripheral devices.

Serial RapidIO physical layer specification defines three line rates:

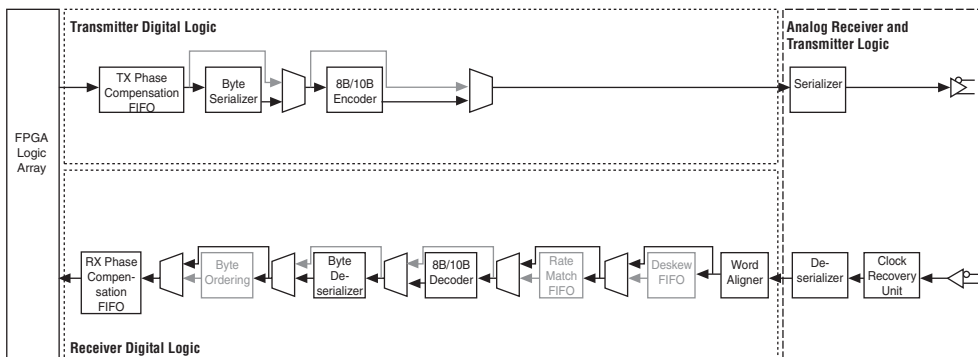
- 1.25 Gbps
- 2.5 Gbps
- 3.125 Gbps

It also defines two link widths—single-lane (1×) and bonded four-lane (4×) at each line rate.

Stratix II GX transceivers support only single-lane (1×) configuration at all three line rates. Four 1× channels configured in Serial RapidIO mode can be instantiated to achieve a 4× Serial RapidIO link. The four transmitter channels in this 4× Serial RapidIO link are not bonded. The four receiver channels in this 4× Serial RapidIO link do not have lane alignment or deskew capability.

Figure 2–147 shows the ALT2GXB transceiver data path when configured in Serial RapidIO mode.

Figure 2–147. Serial RapidIO Mode



Stratix II GX transceivers, when configured in Serial RapidIO functional mode, provide the following PCS and PMA functions:

- 8B/10B encoding/decoding
- Word alignment
- Lane Synchronization State Machine
- Clock recovery from the encoded data
- Serialization/deserialization



Stratix II GX transceivers do not have built-in support for other PCS functions; for example, clock frequency compensation between upstream transmitter clock and local receiver clock (rate matcher), pseudo-random idle sequence generation, and lane alignment in 4× mode. Depending on your system requirements, you must implement these functions in the logic array or external circuits.

Synchronization State Machine

In Serial RapidIO mode, the ALT2GXB MegaWizard Plug-In Manager defaults the word alignment pattern to K28.5. The word aligner has a Synchronization State Machine that handles the receiver lane synchronization.

The ALT2GXB MegaWizard Plug-In Manager automatically defaults the synchronization state machine to indicate synchronization when the receiver receives 127 K28.5 (10'b0101111100 or 10'b1010000011) synchronization code groups without receiving an intermediate invalid code group. Once synchronized, the state machine indicates loss of synchronization when it detects three invalid code groups separated by less than 255 valid code groups or when it is reset.

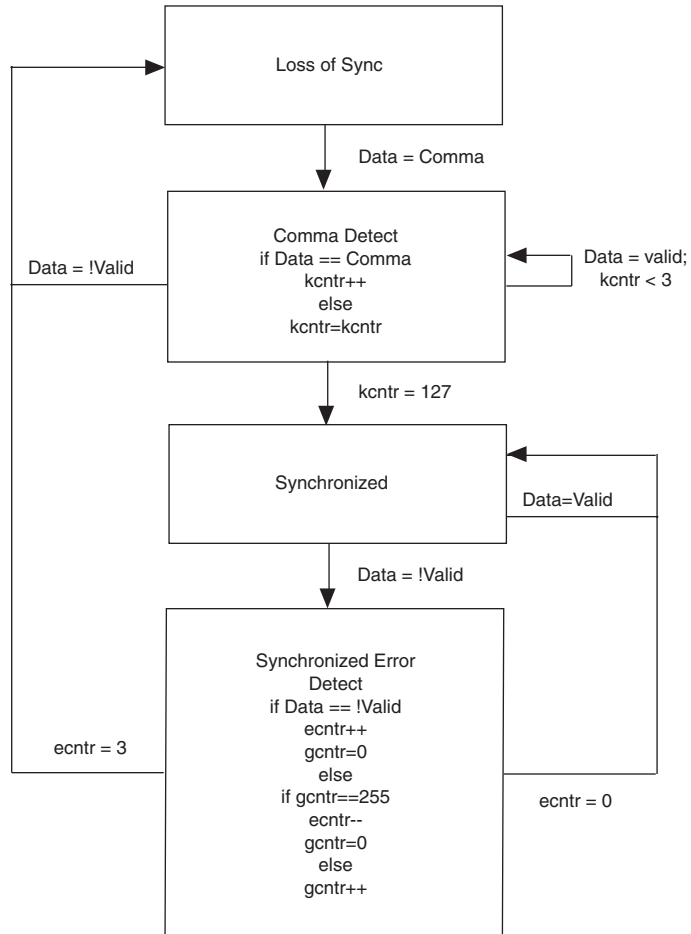
Receiver synchronization is indicated on the `rx_syncstatus` port of each channel. A high on the `rx_syncstatus` port indicates that the lane is synchronized and a low indicates that it has fallen out of synchronization.

Table 2–46 lists the ALT2GXB synchronization state machine parameters when configured in Serial RapidIO mode.

Parameters	Number
Number of valid K28.5 code groups received to achieve synchronization.	127
Number of errors received to lose synchronization.	3
Number of continuous good code groups received to reduce the error count by one.	255

Figure 2–148 gives a conceptual view of the synchronization state machine implemented in Serial RapidIO functional mode.

Figure 2–148. Synchronization State Machine in Serial RapidIO Mode



CPRI Mode

The common public radio interface (CPRI) specification defines a radio base station interface standard between the radio equipment control (REC) and the radio equipment (RE).

CPRI Specification V2.1 defines the following three line rates for deployment flexibility:

- CPRI line bit rate option 1: 614.4 Mbps
- CPRI line bit rate option 2: 1228.8 Mbps (2 x 614.4 Mbps)
- CPRI line bit rate option 3: 2457.6 Mbps (4 x 614.4 Mbps)

Stratix II GX transceivers support all three line bit rate options.

Figure 2–149 shows the ALT2GXB transceiver data path when configured in CPRI mode.

Figure 2–149. ALT2GXB Transceiver Data Path in CPRI Mode

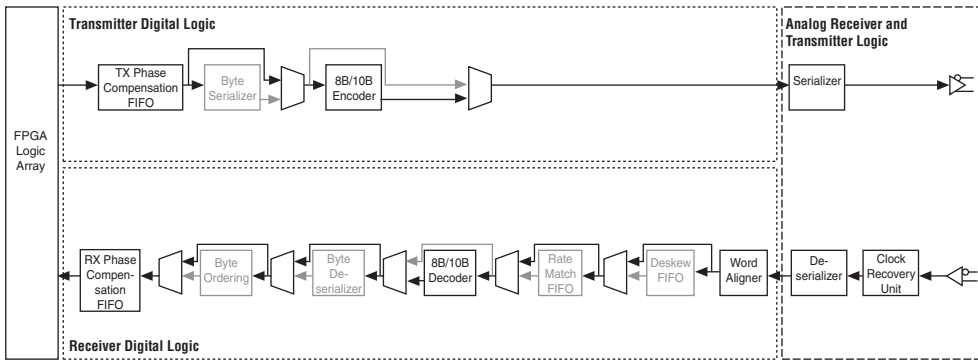


Table 2–47 shows ALT2GXB configurations supported by the Stratix II GX transceivers in CPRI mode.

Data Rate (Mbps)	REFCLK Frequencies (MHz)	Channel Width
614	61.4, 76.675, 122.8, 153.5	8 bit
1228	61.4, 76.675, 122.8, 153.5, 245.6, 307	8 bit
2456	61.4, 76.675, 122.8, 153.5, 245.6, 307, 491.2, 614	8 bit

Transmitter Data Path

The CPRI transmitter data path includes the transmitter phase compensation FIFO, the 8B/10B encoder, and the 10:1 serializer. Layer 1 functions like Hyperframe framing that includes interleaving IQ data, sync data, L1 inband protocol data, and so forth, that must be performed in the FPGA logic array or external circuitry.

Receiver Data Path

The receiver data path includes the clock recovery unit (CRU), 1:10 deserializer, synchronization-state-machine-based word aligner, 8B/10B decoder, and receiver phase compensation FIFO.

The synchronization-state-machine-based word aligner is programmable. You can select the number of bad code groups detected to fall out of synchronization, the number of valid synchronization code groups to acquire synchronization, and the number of good code groups received to reduce the error count by 1.

You can use this programmability to implement the Loss of Signal (LOS), Loss of Frame Synchronization (LOF), and the Remote Alarm Indication (RAI) features in the FPGA logic array. The synchronization status is reported on the `rx_syncstatus` port from the ALT2GXB. The `rx_syncstatus` signal is driven high when the programmed conditions for synchronization state machine are met. Otherwise, it is driven low to indicate loss of synchronization.

Link Delay Accuracy

Requirement R-19 in CPRI Specification V2.1 requires the CPRI link delay accuracy (excluding the transmission medium delay) to be $\pm T_c/32$, where T_c is the length of a basic frame (260.42 ns). This requirement mandates the total uncertainty in CPRI link latency to be less than ~16.3 ns.

To meet the strict CPRI link delay accuracy requirements, the Quartus II software automatically adjusts the routing delays on the transmitter and receiver phase compensation FIFO clocks. As a result of this delay adjustment, the transmitter and receiver phase compensation FIFO latency becomes constant.



The Quartus II software performs the delay adjustment to minimize the uncertainty in link latency only in CPRI mode.

Table 2–48 shows the uncertainty in the CPRI transceiver data path after delay adjustment.

Data Rate (Mbps)	Receiver Deserializer Uncertainty (Parallel Clock Cycles)	Receiver Phase Comp FIFO Uncertainty (Parallel Clock Cycles)	Transmitter Phase Comp FIFO Uncertainty (Parallel Clock Cycles)	Parallel Clock Period (ns)	Total Uncertainty (ns)
614	0.9	0	0	16.3	14.67
1228	0.9	0	0	8.15	7.34

Note to Table 2–48:

- (1) The delay adjustment is not made for CPRI 2456 Mbps line rate configuration since it meets the 16.3 ns link delay requirement without these adjustments.

Table 2–48 shows that the CPRI link delay accuracy requirements are met within the transceiver data path.

Transceiver Limitations in CPRI Mode

To meet the CPRI link delay accuracy requirements, the Quartus II software adjusts delays on the clock routes from the `tx_clkout` and `rx_clkout` ports to the write and read ports of the transmitter and receiver phase compensation FIFOs, respectively, for each transceiver channel. Due to this requirement, the Quartus II software only allows the `tx_clkout` signal from each channel to clock the write port of its transmitter phase compensation FIFO. Similarly, it allows the `rx_clkout` signal from each channel to clock the read port of its receiver phase compensation FIFO. If your design requires dynamic reconfiguration between CPRI mode and other modes, each channel's phase compensation FIFOs must be clocked by its own `tx_clkout` and `rx_clkout` for other modes as well. The default transceiver configuration used to create the programming file (`.sof` or `.pof`) must be CPRI for the delay algorithm to take effect.

In CPRI mode, you cannot group the `tx_coreclk` and/or `rx_coreclk` ports of multiple channels and drive them using a common clock driver using the shared clock or 0 PPM clock group assignments. Each CPRI channel will utilize at least two global and/or regional clock resources. Since a maximum of 32 global and/or regional clock resources are available for transceivers in the Stratix II GX device, the clock resource availability governs the maximum number of CPRI channels that you can instantiate per device.



The maximum number of CPRI channels per device also depends on the number of transceiver channels available in that device and the LRIO clock resource limitations.

CPRI mode is supported only in C3, C4, and I4 speed-grade devices.

PLD-Transceiver Interface Clock Duty Cycle

Due to delay adjustments made to the `tx_clkout` and `rx_clkout` routes in the FPGA clock network, the worst case duty cycle on these PLD-transceiver interface clocks can be 60-40%. If these clocks are used to clock FPGA logic array, you must set proper timing analyzer assignments to account for the 60-40% duty cycle on these clocks.

Figure 2-150 shows how to set 60-40% duty cycle constraints in TimeQuest Timing Analyzer for a CPRI 614.4 Mbps line rate configuration. In the TimeQuest Analyzer window, selecting the **Report Clocks** option lists all the `tx_clkout` and `rx_clkout` clocks in your design. Select all clocks and adjust the falling edge timing appropriately for 60-40% duty cycle.

Figure 2-150. Setting Duty Cycle in TimeQuest Timing Analyzer

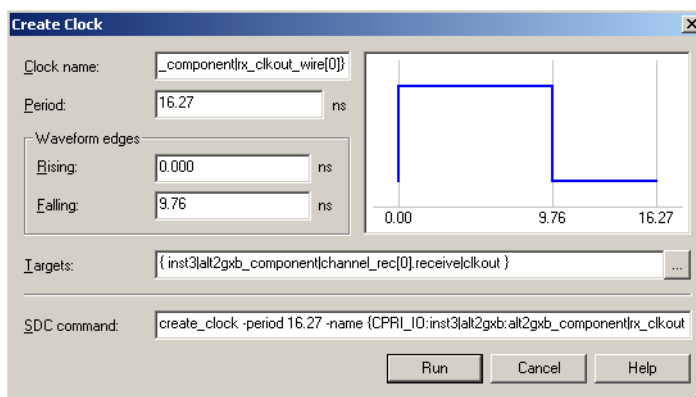
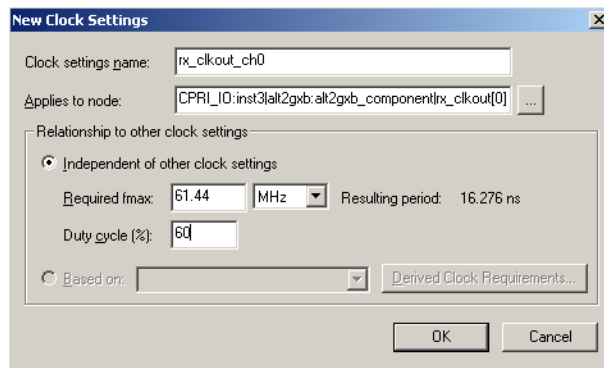


Figure 2-151 shows how to set 60-40% duty cycle constraints in Classic Timing Analyzer. To set this option, go to the Assignments menu and select **Settings**, then select **Timing Analysis Settings** and **Classic Timing Analyzer Settings**, and then click on the **Individual Clocks** tab. In the **Individual Clocks** window, select **New**. In the **New Clock Settings** window, browse to all `rx_clkout` and `tx_clkout` nodes and assign 60% in the **Duty Cycle** option for each of these phase compensation FIFO clocks.

Figure 2–151. Setting Duty Cycle in Classic Timing Analyzer

If the PLD-transceiver interface clocks are fed to an off-chip PLL (for example, VCXO-based PLL for de-jittering purposes), you must make sure that the PLL can tolerate the 60-40% duty cycle on its input reference clock.

Loopback Modes

There are several loopback modes available on the Stratix II GX transceiver block that allow you to isolate portions of the circuit. All paths are designed to run up to full speed. The available loopback paths are:

- Serial loopback available in all functional modes except PCI Express (PIPE)
- Reverse serial loopback available in Basic mode with 8B/10B
- PCI Express PIPE reverse parallel loopback available in PCI Express protocol
- Reverse serial pre-CDR loopback available in Basic mode with 8B/10B
- Reverse serial loopback available in Basic mode with 8B/10B
- Parallel loopback available in Basic mode for BIST testing only

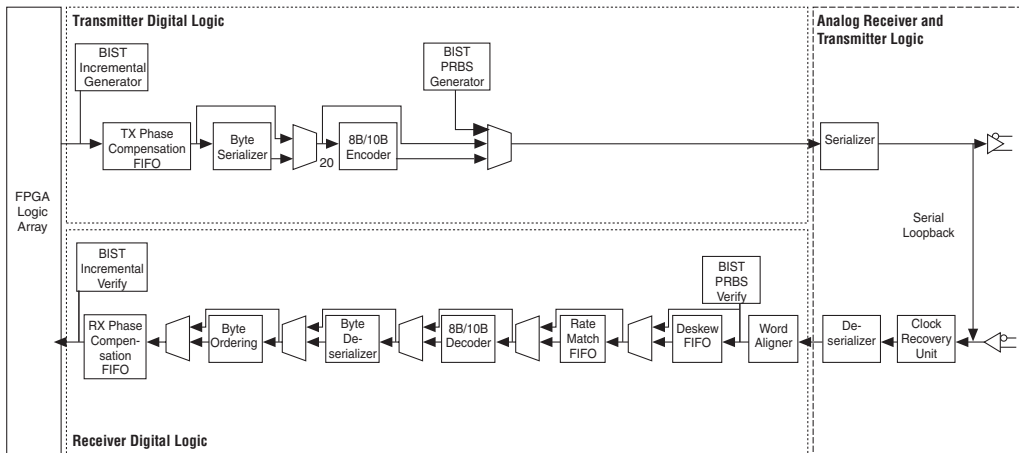
Serial Loopback

Figure 2–152 shows the data path for serial loopback. A data stream is fed to the transmitter from the FPGA logic array and has the option of utilizing all the blocks in the transmitter. The data, in serial form, then traverses from the transmitter to the receiver. The serial data is the data that is transmitted from the Stratix II GX device. Once the data enters the receiver in serial form, it can utilize any of the receiver blocks and is then fed into the FPGA logic array.

Use the `rx_serialpbken` port to dynamically enable serial loopback on a channel by channel basis. When `rx_serialpbken` is high, all blocks that are active when the signal is low are still active. When the serial loopback is enabled, the `tx_dataout` port is still active and drives out the output pins.

Serial loopback is often used to check the entire path of the transceiver. The data is retimed through different clock domains and an alignment pattern is still necessary for the word aligner.

Figure 2–152. Stratix II GX Block in Serial Loopback Mode

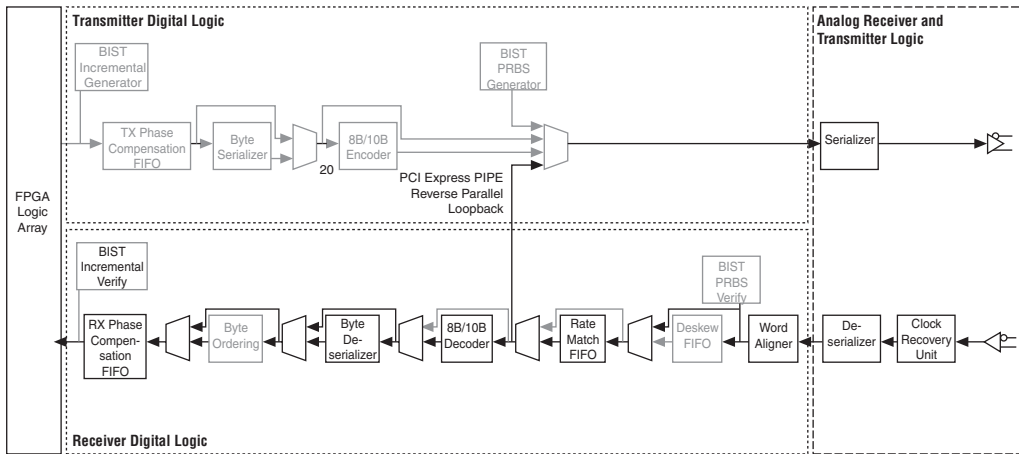


PCI Express PIPE Reverse Parallel Loopback

Figure 2–153 shows the data path for the PCI Express PIPE reverse parallel loopback. This data path is not flexible because it must be compliant with the PCI Express PIPE specification. The data comes in from the `rx_datain` ports. The receiver uses the CRU, deserializer, word aligner, and rate matching FIFO buffer, loops back to the transmitter serializer, and then goes out the transmitter `tx_dataout` ports. The data also goes to the PLD fabric on the receiver side to the `tx_dataout` port. The deskew FIFO buffer is not enabled in this loopback mode. This loopback mode is optionally controlled dynamically through the `tx_detectrxloopback` port.



This is the only loopback allowed in the PIPE mode.

Figure 2–153. Stratix II GX Block in PCI Express PIPE Reverse Parallel Loopback Mode

Reverse Serial Loopback

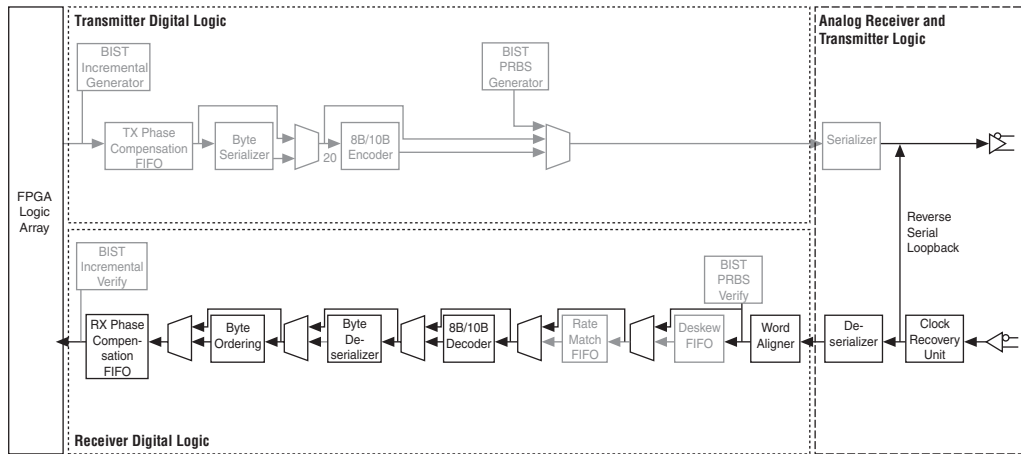
Reverse serial loopback is a subprotocol in Basic mode. It requires 8B/10B, and the word aligner pattern of K28.5. No dynamic pin control is available to select or deselect reverse serial loopback. The active block of the transmitter is only the buffer. The data sent to the receiver is retimed with the recovered clock and sent out to the transmitter.

The data path for reverse serial loopback is shown in [Figure 2–154](#). Data comes in from the `rx_datain` ports in the receiver. The data is then fed through the CDR block in serial form directly to the `tx_dataout` ports in the transmitter block.

You can enable reverse serial loopback for all channels through the MegaWizard. Any pre-emphasis setting on the transmitter buffer is ignored in reverse serial loopback. The data flows through the active blocks of the receiver and into the logic array.

Reverse serial loopback is often implemented when using a bit error rate tester (BERT).

Figure 2–154. Stratix II GX Block in Reverse Serial Loopback Mode

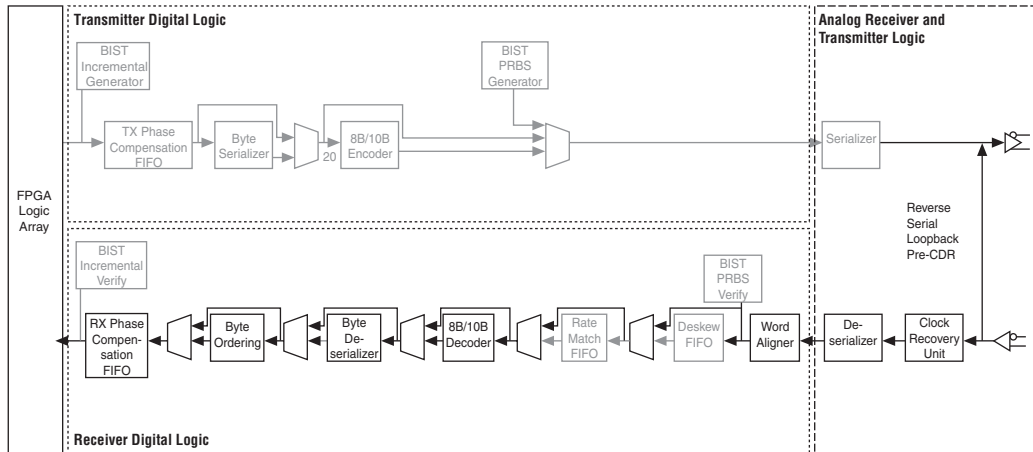


Reverse Serial Pre-CDR Loopback

The reverse serial pre-CDR loopback uses the analog portion of the transceiver. An external source (pattern generator or transceiver) generates the source data. The high-speed serial source data arrives at the high-speed differential receiver input buffer, loops back before the CRU unit, and is transmitted through the high-speed differential transmitter output buffer. It is for test or verification use only to verify the signal being received after the gain and equalization improvements of the input buffer. The signal at the output is not exactly what is received since the signal goes through the output buffer and the VOD is changed to the VOD setting level. The pre-emphasis settings have no effect.

Figure 2–155 show the Stratix II GX block in reverse serial pre-CDR loopback mode.

Figure 2–155. Stratix II GX Block in Reverse Serial Pre-CDR Loopback Mode



Parallel Loopback

The data path for parallel loopback is shown in Figure 2–156. The forward parallel loopback allows a test flow check of the PCS using either the built-in test incremental pattern. This is available only as a subprotocol in Basic Double-Width mode.

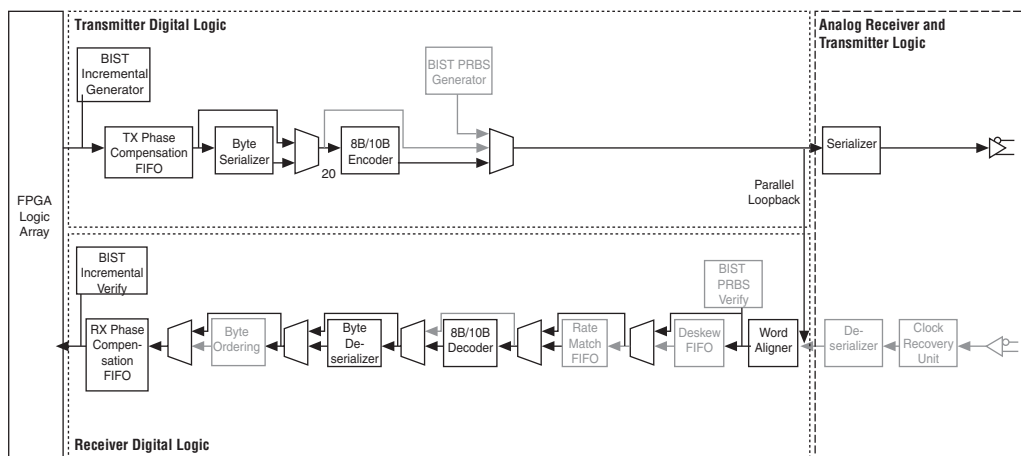
When using the BIST incremental parallel loopback, the deskew and the rate matching FIFO buffer are not available. The 8B/10B encoder and decoder are used. No dynamic control pin is available to enable or disable the loopback. Test result pins, `rx_bistdone` and `rx_bisterr`, are available in this loopback mode.

When using parallel loopback, the `tx_dataout` ports are active and the differential output voltage on the `tx_dataout` ports is based on the V_{OD} settings.

Table 2–49 shows the available BIST patterns in double-width mode.

PATTERN	Word Aligner Alignment Pattern	Byte Order Align Pattern	Description	Double-Width Mode	
				16 Bit	20 Bit
Incremental with 8B/10B	20'h16E83	N/A	All 8B/10B valid code groups		✓

Figure 2–156. Stratix II GX Block in Parallel Loopback Mode



Incremental Pattern Generator

The incremental data generator sweeps through all the valid 8B/10B data and control characters. This mode is only available in Basic mode with the BIST/parallel loopback subprotocol in the Quartus II software. You can also enable the incremental BIST verifier to perform a quick verification of the 8B/10B encoder/decoder paths.

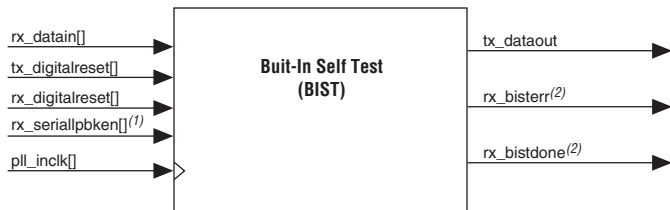
In incremental mode, the BIST generator sends out the data pattern in the following sequence: K28.5 (comma), K27.7 (start of frame, SOF), Data (00-FF incremental), K28.0, K28.1, K28.2, K28.3, K28.4, K28.6, K28.7, K23.7, K30.7, K29.7 (end of frame, EOF), and then repeats. You must enable the 8B/10B encoder for proper operation. No dynamic control pin is available to enable or disable the loopback. Test result pins are

`rx_bistdone` and `rx_bisterr`. The `rx_bistdone` signal goes high at the end of the sequence. If the verifier detects an error before it is finished, `rx_bisterr` pulses high as long as the data is in error.

Built-In Self-Test Modes

Besides the regular data flow blocks, each transceiver channel contains an embedded built-in self test (BIST) generator and corresponding verifier block that you can use for quick device and setup verification (refer to [Figure 2-157](#)). The generators reside in the transmitter block and the verifier in the receiver block. The generators can generate PRBS and incremental patterns. The incremental pattern is available only in Parallel loopback mode. The verifiers are only available for these data patterns. The BIST blocks operate differently when in the single-width mode and the double-width mode. The BIST modes are only available as subprotocols under Basic mode.

Figure 2-157. Built-In Self Test Mode

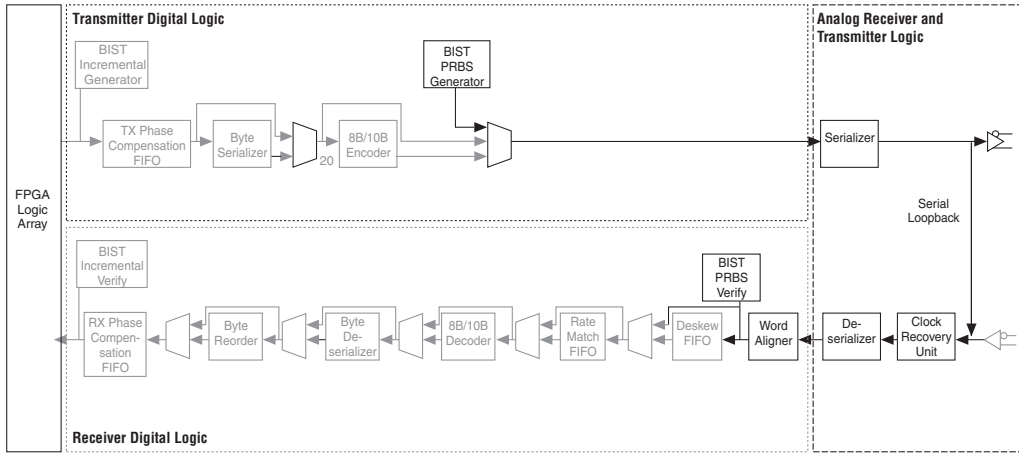


Notes to [Figure 2-157](#):

- (1) `rx_serialpbken []` is required in PRBS.
- (2) `rx_bisterr []` and `rx_bistdone []` are only available in PRBS and BIST modes.

Figure 2–158 shows the PRBS blocks with loopback used in the transceiver channel.

Figure 2–158. PRBS Blocks With Loopback in Transceiver Channel



BIST in Single-Width Mode

Single-width mode supports PRBS10 pattern generation and verification. PRBS10 in Basic mode is supported with or without serial loopback



The PRBS10 pattern is only available when the SERDES factor is 10 bits.

Table 2–50 shows the BIST patterns for single-width mode.

Pattern	Word Aligner Alignment Pattern	Byte Order Align Pattern	Description	Single-Width Mode	
				8 Bit	10 Bit
PRBS10	10'h3FF	N/A	$X^{10} + X^7 + 1$		✓

PRBS10

Pseudo-Random Bit Sequences (PRBS) are commonly used in systems to verify the integrity and robustness of the data transmission paths. When the SERDES factor is 10, use the PRBS10 pattern. The PRBS generator yields $2^{10}-1$ unique serial patterns. You can use PRBS with or without serial loopback. In PRBS/ serial loopback mode, the rx_serialpbken signal is available. In the PRBS/no loopback mode, this control signal is not available.

You enable PRBS mode in the Quartus II ALT2GXB MegaWizard Plug-In Manager. PRBS10 does not use the 8B/10B encoder and decoder. The 8B/10B encoder and decoder are bypassed automatically in the PRBS mode.

The advantage of using a PRBS data stream is that the randomness yields an environment that stresses the transmission medium. In the data stream, you can observe both random jitter and deterministic jitter using a time interval analyzer, bit error rate tester, or oscilloscope.

The PRBS verifier can provide a quick check through the non-8B/10B path of the transceiver block. The PRBS verifier is active once the receiver channel is synchronized. Set the alignment pattern to 10'h3FF for the 10-bit SERDES modes.

The verifier stops checking the patterns after receiving all the PRBS patterns (1023 patterns for 10-bit mode). The rx_bistdone signal goes high, indicating that the verifier has completed. If the verifier detects an error before it is finished, rx_bisterr pulses high for the time the data is incorrect. Use the rx_digitalreset signal to re-start the PRBS verification.

The 8B/10B encoder is enabled, so the data stream is DC balanced. 8B/10B encoding guarantees a run length of less than 5 UI, which yields a less stressful pattern versus the PRBS data. However, since the PRBS generator bypasses the 8B/10B paths, the incremental BIST can test this path.

BIST in Double-Width Mode

Double-width mode supports only PRBS7 pattern generation and verification.



The PRBS7 pattern is only available when the SERDES factor is 20 bits.

Table 2–51 shows the BIST patterns for double-width mode.

PATTERN	Word Aligner Alignment Pattern	Byte Order Align Pattern	Description	Double-Width Mode	
				16 Bit	20 Bit
PRBS7	20'h43040	N/A	$X^7 + X^6 + 1$		✓

PRBS7

Pseudo-Random Bit Sequences (PRBS) are commonly used in systems to verify the integrity and robustness of the data transmission paths. The PRBS7 generator generates 2^7-1 unique patterns. PRBS can be used with or without serial loopback. In PRBS/ serial loopback mode, the `rx_serialpbken` signal is available. In the PRBS/no loopback mode, this control signal is not available.

You enable PRBS mode in the MegaWizard. PRBS7 does not use the 8B/10B encoder and decoder. The 8B/10B encoder and decoder are bypassed automatically in the PRBS mode.

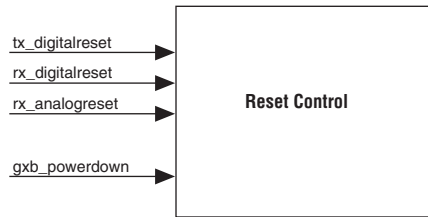
The advantage of using a PRBS data stream is that the randomness yields an environment that stresses the transmission medium. In the data stream you can observe both random jitter and deterministic jitter using a time interval analyzer, bit error rate tester, or oscilloscope.

The PRBS verifier provides a quick check through the non-8B/10B path of the transceiver block. The PRBS verifier is active once the receiver channel is synchronized. Set the alignment pattern to 20'h43040 for the 20-bit SERDES modes. The PRBS verifier prevents the word aligner from aligning to a new pattern after the first five successfully verified words.

The verifier stops checking the patterns after receiving all the PRBS patterns (127 patterns for PRBS7). The `rx_bistdone` signal goes high, indicating that the verifier has completed. If the verifier detects an error before it is finished, `rx_bisterr` pulses high for as long as the data is in error. Use the `rx_digitalreset` signal to re-start the PRBS verification.

Reset Control and Power Down

Stratix II GX transceivers offer multiple reset signals to control separate ports of the transceiver channels and blocks (Figure 2–159). You can set each unused channel to a power-down mode to reduce power consumption.

Figure 2–159. Reset Control and Power Down

User Reset and Enable Signals

Each transceiver block and each channel in the transceiver block of the Stratix II GX device has individual reset signals to reset the digital and analog portions of the channel. The analog resets are power-down signals, which require a longer pulse width for the circuits to power down. The `tx_digitalreset`, `rx_digitalreset`, and `rx_analogreset` signals affect the channels individually. The `gxb_powerdown` signal affects the entire transceiver block.



All the reset and enable signals are not required. If not used, the signals are defaulted to not reset for all reset signals and enabled for the PLL enable signal. All reset and enable signals are asynchronous.

- `tx_digitalreset`. The `tx_digitalreset` signal resets all digital logic in the transmitter, including the XAUI transmit state machine, the BIST-PRBS generator, and the BIST pattern generator. This signal operates independently from the other reset signals. The minimum pulse width is two parallel cycles.
- `rx_digitalreset`. The `rx_digitalreset` signal resets all digital logic in the receiver, including the XAUI and GIGE receiver state machine, the XAUI channel alignment state machine, the BIST-PRBS verifier, and the BIST-incremental verifier. This signal operates independently from the other reset signals. The minimum pulse width is two parallel cycles.
- `rx_analogreset`. The `rx_analogreset` signal resets part of the analog portion of the receiver CDR. This signal operates independently from the other reset signals. The minimum pulse width is two parallel clock cycles.

- `gxb_powerdown`. The `gxb_powerdown` signal powers down the entire transceiver block. All digital and analog circuits are also reset. This signal operates independently from the other reset signals. The minimum pulse width for `gxb_powerdown` signal is 100 ns

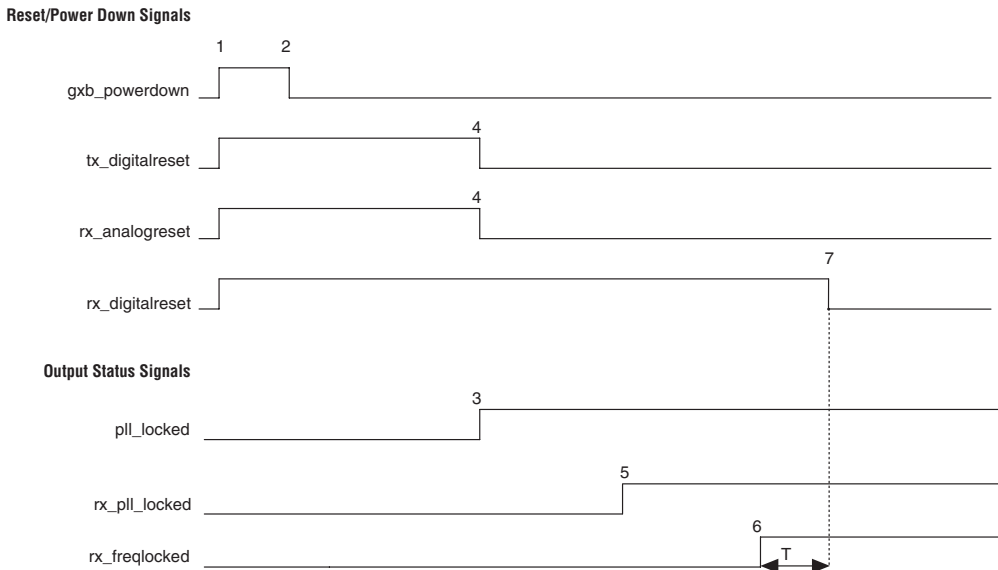
Synchronization is performed after any reset condition. You must determine when the data is valid after reset (for example, by using the `rx_syncstatus` signal). [Table 2-52](#) shows the blocks affected by each reset and power-down signal.

Transceiver Blocks	<code>rx_digitalreset</code>	<code>rx_analogreset</code>	<code>tx_digitalreset</code>	<code>gxb_powerdown</code>
Transmitter phase compensation FIFO buffer and byte serializer			✓	✓
Transmitter 8B/10B encoder			✓	✓
Transmitter serializer				✓
Transmitter analog circuits				✓
Transmitter PLLs				✓
Transmitter XAUI state machine			✓	✓
Transmitter analog circuits				✓
BIST generators			✓	✓
Receiver deserializer				✓
Receiver word aligner	✓			✓
Receiver deskew FIFO buffer	✓			✓
Receiver rate matcher	✓			✓
Receiver 8B/10B encoder	✓			✓
Receiver phase compensation FIFO buffer and byte deserializer	✓			✓
Receiver PLL and CRU		✓		✓

Transceiver Blocks	rx_digitalreset	rx_analogreset	tx_digitalreset	gxb_powerdown
Receiver XAUI state machine	✓			✓
Receiver byte ordering block	✓			✓
BIST verifiers	✓			✓
Receiver analog circuits				✓

Figure 2–160 shows a sample reset cycle.

Figure 2–160. Reset Power Signal Timing Waveform



Notes to Figure 2–160:

- (1) tx_digitalreset is valid in transmitter only and duplex configuration.
- (2) rx_analogreset and rx_digitalreset are valid in receiver only and duplex configuration.

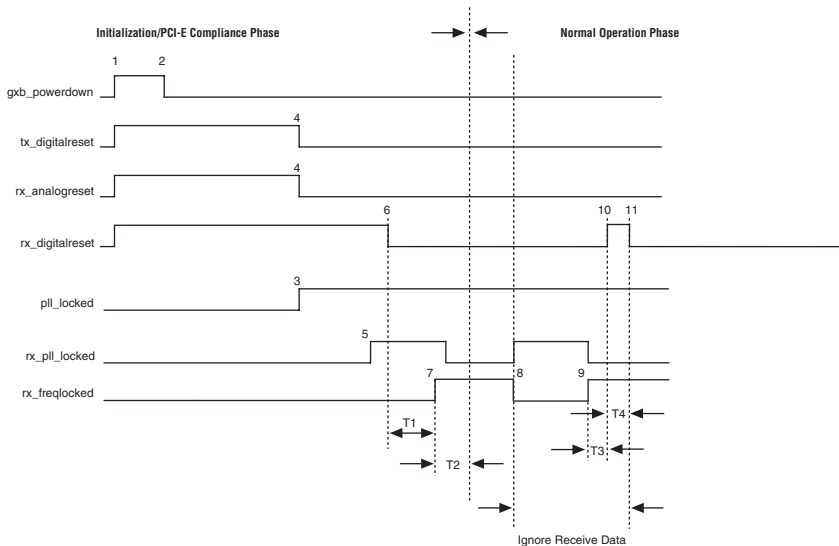
The minimum pulse width for the gxb_powerdown port (between time marker 1 and 2) is 100 ns. The tx_digitalreset and rx_analogreset signals can be deasserted after the driving PLL asserts its associated pll_locked signal. The rx_digitalreset signal can be deasserted 4 μ s after the rx_freqlocked signal goes high (time between markers 6 and 7).

In a transmitter only configuration, only the `pll_locked`, `gxb_powerdown`, and `tx_digitalreset` signals are used. In a receiver only configuration, only the `rx_analogreset`, `rx_digitalreset`, and `rx_freqlocked` signals are used.

Reset Sequence for PIPE Mode

The reset sequence used for the other modes looks for `rx_freqlocked` signal to deassert `rx_digitalreset`. In PIPE mode, the `rx_freqlocked` signal does not go high during the PCI-E compliance testing phase because of receiving Electrical Idle. Figure 2-161 shows the reset sequence for PIPE mode.

Figure 2-161. PIPE Mode Reset Sequence



Initialization and PCI-E Compliance Phase

After the device is powered up, any PCI-E compliant device performs compliance testing. During this phase, all the transceiver reset signals (`gxb_powerdown`, `tx_digitalreset`, `rx_analogreset`, and `rx_digitalreset`) are asserted.



The minimum time period between markers 1 and 2 for the `gxb_powerdown` signal is 100 ns (Figure 2-161).

The `tx_digitalreset` and `rx_analogreset` signals can be deasserted after the `p11_locked` signal goes high. The reset controller should deassert the `rx_digitalreset` when the `rx_pll_locked` signal goes high.

The parallel data sent to the PLD logic array in the receive side may not be valid until 4 us (T2) after `rx_freqlocked` goes high.

Normal Operation Phase

During normal operations, the receive data is valid and the `rx_freqlocked` signal is high. In this situation, when `rx_freqlocked` is deasserted, (marker 8 in Figure 2–161), the reset controller should wait for the `rx_freqlocked` to go high again and assert `rx_digitalreset` (marker 10 in Figure 2–161) for two parallel receive clock cycles.

The data from the gigabit transceiver block is not valid between the time when `rx_freqlocked` goes low until `rx_digitalreset` is deasserted. The PLD logic should ignore the data during this time period (the time period between markers 8 and 11 in Figure 2–161).

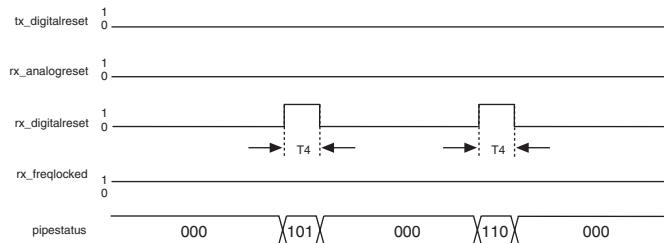


Minimum T1 period is 100 ns. Minimum T2 and T3 periods are 4 us. T4 indicates two parallel receive clock cycles.

Rate Matcher FIFO Buffer Overflow and Underflow Condition

During the normal operating phase, the reset controller monitors the overflow and underflow status of the rate matcher FIFO buffer. If there is overflow and underflow on the rate matcher FIFO buffer, the reset controller asserts `rx_digitalreset` for two receive parallel clock cycles. You can monitor the rate matcher FIFO buffer status through the `pipestatus[2:0]` signal from the PIPE interface. This condition is shown in Figure 2–162.

Figure 2–162. PIPE Mode Reset During Rate Matcher FIFO Buffer Overflow and Underflow Condition




Notes to Figure 2–162:

- (1) Pipestatus = 101 represents elastic overflow.
- (2) Pipestatus = 110 represents elastic overflow.

Power Down

The Quartus II software automatically selects the power-down channel feature, which takes affect when you configure the Stratix II GX device. All unused transceiver and blocks in a design are powered down to reduce the overall power consumption. You cannot use the power-down feature on the fly to turn the transceiver channels and transceiver blocks on and off without reconfiguration.

You can set the transceiver block to power down automatically in the Quartus II software or to power down dynamically in the PLD fabric through the `gxb_powerdown` port. Assertion of this port does not power down the `refclk` reference clock buffer.

 The `gxb_powerdown` port is optional. In simulation, if the `gxb_powerdown` port is not instantiated, you must assert `tx_digitalreset`, `rx_digitalreset` and `rx_analogreset` signals appropriately for correct simulation behavior. If the `gxb_powerdown` port is instantiated and other reset signals are not used, you must assert the `gxb_powerdown` signal for at least one parallel clock cycle for correct simulation behavior. In simulation, you can de-assert the `rx_digitalreset` immediately after `rx_freqlocked` signal goes high to reduce the simulation run time. It is not necessary to wait for 4 us as suggested in the actual reset sequence.


 In PIPE mode simulation, you must assert the `tx_forceelecidle` signal for at least one parallel clock cycle before transmitting normal data for correct simulation behavior.

Table 2–53 lists the I/O pin states during power down for normal operation, power down, and PMA lookback.

Operation	Transmitter Pins	Receiver Pins	REFCLK Pins	Rref Pins
Normal operation	Transmitter	Receiver	Clk input	Ext. reference R
Power down	Tri-state (1)	Tri-state (1)	Tri-state (2)	Low (3)

Table 2–53. I/O Pin States During Power-Down (Part 2 of 2)

Operation		Transmitter Pins	Receiver Pins	REFCLK Pins	Ref Pins
PMA loopback	Serial loopback	Tri-state (4) toggle	Receiver (5)	—	—
	Reverse serial loopback	Transmitter (4)	Receiver	—	—

Notes to Table 2–53:

- (1) Either leave these pins floating or connect `n_leg` to GND through a 10-k Ω resistor and connect `p_leg` to `GXB_VCC` through a 10-k Ω resistor to improve the device's immunity to noise.
- (2) Either leave these pins floating or connect `refclk(n)` to GND through a 10-k Ω resistor and connect `refclk(p)` to `GXB_VCC` through a 10-k Ω resistor to improve the device's immunity to noise.
- (3) Altera recommends driving the reference resistor pin low for the powered down transceiver block.
- (4) All supported VODs.
- (5) It must be left floating or driven to a constant value.

TimeQuest Timing Analyzer

For Stratix II GX designs, you can either use the Classic Timing analyzer or TimeQuest for static timing analysis. TimeQuest does not automatically constrain the transceiver reset ports and asynchronous input/output ports. As a result, TimeQuest does not perform timing analysis on these paths.

TimeQuest reports these unconstrained paths in RED in the Timing Analyzer report. You must manually add the constraints in the Synopsys Design Constraints (`.sdc`) file for TimeQuest to analyze these paths.

Unconstrained Reset Ports

In the Quartus II software versions 7.1 and 7.1 sp1, TimeQuest does not constrain the following transceiver reset ports:

- `gxb_powerdown`
- `tx_digitalreset`
- `rx_digitalreset`
- `rx_analogreset`

Identifying Unconstrained Reset Ports

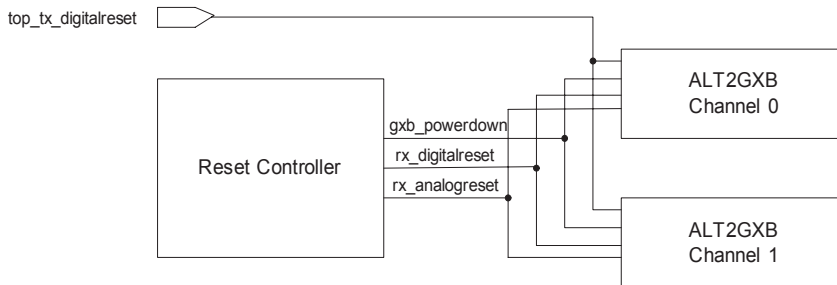
To identify the unconstrained reset/powerdown ports, follow these steps:

1. After compiling your design, select the **TimeQuest Timing Analyzer** in the **Tools** drop-down menu. This opens up the Quartus II TimeQuest Timing Analyzer window.

2. In the **Tasks** pane, execute **Report Unconstrained Paths**. This will report all unconstrained paths in RED in the **Report** pane.
3. Expand the **Unconstrained Paths** option in the **Report** pane and further expand the **Setup Analysis** or **Hold Analysis** option.
4. Under **Setup Analysis** or **Hold Analysis**, you will see **Unconstrained Input Port Paths**, **Unconstrained Output Port Paths**, or both, depending on how the reset/powerdown ports are driven.
 - a. If a reset/powerdown port is driven by an input pin, it will be listed in the **Unconstrained Input Port Paths** report.
 - b. If a reset/powerdown port is driven by synchronous logic, it will be listed in the **Unconstrained Output Port Paths** report.
5. In the **Unconstrained Input Port Paths** and **Unconstrained Output Port Paths** reports, the unconstrained reset/powerdown ports of your ALT2GXB instances are listed under the **To** column.

Consider the design example in [Figure 2–163](#).

Figure 2–163. Example Design for TimeQuest Timing Analyzer Constraints



In the design example in [Figure 2–163](#), all reset/powerdown ports except the `tx_digitalreset` port for the two channels are driven by the reset controller. The `tx_digitalreset` port is driven from an input pin.

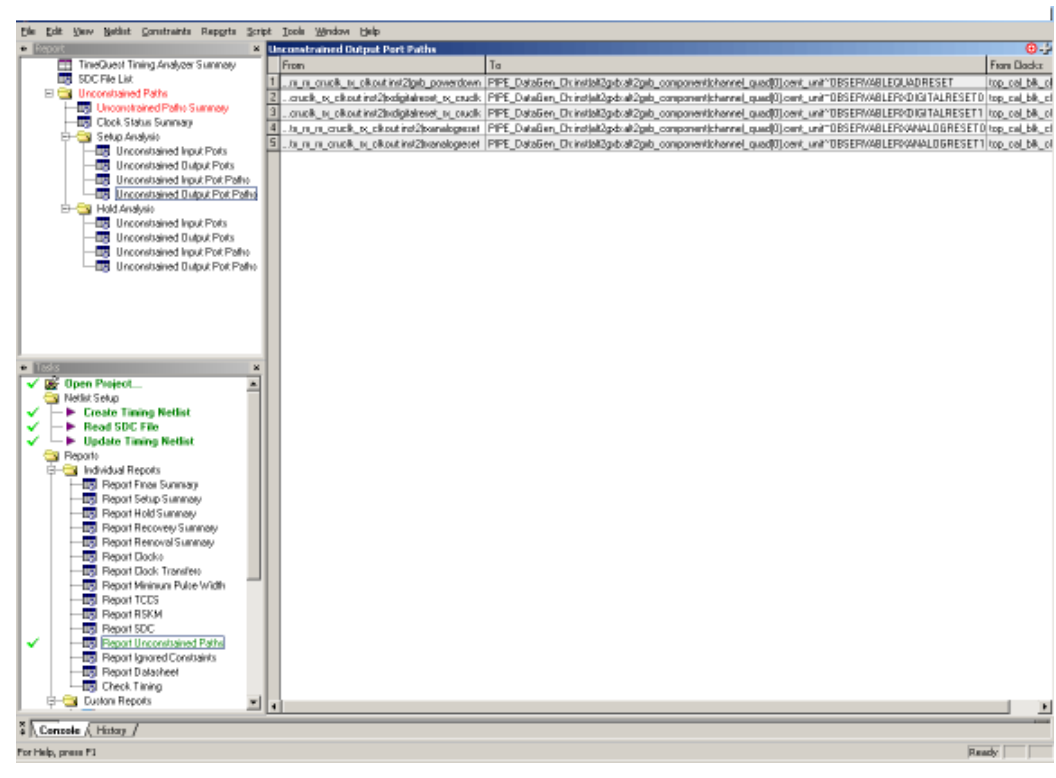
[Figures 2–164](#) and [2–165](#) show the TimeQuest Timing Analyzer Report for **Unconstrained Input Port Paths** and **Unconstrained Output Port Paths**, respectively.

Figure 2–164. Unconstrained Input Port Paths

The screenshot shows the 'Unconstrained Input Port Paths' report window. The report is organized into a table with columns for 'From' and 'To'. The 'From' column lists components like 'inst1k03pba02pab_componentbx_trelockeded0' and 'inst1k03pba02pab_componentbx_locked_out0'. The 'To' column lists signals such as 'reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[7]', 'clk_signalap:auto_signalap_0b4d_signalap_inst_0b4d_signalap_bodylocq_trigger_in_reg[56]', and 'clk_signalap:auto_signalap_0b4d_signalap_inst_0b4d_signalap_bodylocq_trigger_in_reg[57]'. The 'To' column also includes 'Tap_Cr' and 'Tap_Cs' labels.

From	To	Tap
41 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[7]	Tap_Cs
42 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_tx_cruck	Tap_Cr
43 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[13]	Tap_Cs
44 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[3]	Tap_Cr
45 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[16]	Tap_Cs
46 ..inst1k03pba02pab_componentbx_locked_out0	clk_signalap:auto_signalap_0b4d_signalap_inst_0b4d_signalap_bodylocq_trigger_in_reg[56]	Tap_Cr
47 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[11]	Tap_Cs
48 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[1]	Tap_Cr
49 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[3]	Tap_Cs
50 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[12]	Tap_Cr
51 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[8]	Tap_Cs
52 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[13]	Tap_Cr
53 ..inst1k03pba02pab_componentbx_trelockeded0	clk_signalap:auto_signalap_0b4d_signalap_inst_0b4d_signalap_bodylocq_trigger_in_reg[57]	Tap_Cr
54 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[10]	Tap_Cs
55 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[14]	Tap_Cr
56 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[15]	Tap_Cs
57 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[18]	Tap_Cr
58 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[2]	Tap_Cs
59 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[4]	Tap_Cr
60 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[7]	Tap_Cs
61 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[6]	Tap_Cr
62 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[9]	Tap_Cs
63 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[5]	Tap_Cr
64 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[7]	Tap_Cs
65 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[13]	Tap_Cr
66 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[13]	Tap_Cs
67 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[8]	Tap_Cr
68 ..inst1k03pba02pab_componentbx_trelockeded0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[16]	Tap_Cs
69 ..inst1k03pba02pab_componentbx_locked_out0	clk_signalap:auto_signalap_0b4d_signalap_inst_0b4d_signalap_bodylocq_trigger_in_reg[57]	Tap_Cr
70 ..inst1k03pba02pab_componentbx_locked_out0	reset_soa_h_u_rs_cruck_tx_clock_inst2iwaltstst_line[STRIBE_TX_PL1_LOCKED	Tap_Cr
71 ..inst1k03pba02pab_componentbx_locked_out0	clk_signalap:auto_signalap_0b4d_signalap_inst_0b4d_signalap_bodylocq_trigger_in_reg[5]	Tap_Cr
72 ..inst1k03pba02pab_componentbx_locked_out0	clk_signalap:auto_signalap_0b4d_signalap_inst_0b4d_signalap_bodylocq_trigger_in_reg[5]	Tap_Cr
73 ..inst1k03pba02pab_componentbx_locked_out0	clk_signalap:auto_signalap_0b4d_signalap_inst_0b4d_signalap_bodylocq_trigger_in_reg[5]	Tap_Cr
74 ..inst1k03pba02pab_componentbx_locked_out0	clk_signalap:auto_signalap_0b4d_signalap_inst_0b4d_signalap_bodylocq_trigger_in_reg[5]	Tap_Cr
75 ..inst1k03pba02pab_componentbx_locked_out0	clk_signalap:auto_signalap_0b4d_signalap_inst_0b4d_signalap_bodylocq_trigger_in_reg[5]	Tap_Cr
76 ..inst1k03pba02pab_componentbx_locked_out0	clk_signalap:auto_signalap_0b4d_signalap_inst_0b4d_signalap_bodylocq_trigger_in_reg[5]	Tap_Cr

Figure 2–165. Unconstrained Output Port Paths



Having identified the unconstrained reset/powerdown ports in the design, the next step is to constrain these ports.

Setting Reset/Powerdown Port Timing Constraints

You must add the reset/powerdown port timing constraints either directly in the SDC file or through the TimeQuest Timing Analyzer GUI.

To add the timing constraints using the TimeQuest GUI, follow these steps:

1. Locate the reset/powerdown ports in either the **Unconstrained Input Port Paths** or **Unconstrained Output Port Paths** report.
2. Right click on the reset/powerdown port in the **To** column and select **Set Max Delay**. On the resulting window, enter an initial **Delay Value** of 4 ns.

3. Right click on the reset/powerdown port in the **To** column again and select **Set Min Delay**. On the resulting window, enter an initial **Delay Value** of 1.2 ns.



The difference between the maximum delay and minimum delay is set to 2.8 ns which is the maximum skew allowed on reset/powerdown ports.

4. Similarly, set the maximum and minimum delay for all transceiver reset/powerdown ports in your design.
5. Execute **Update Timing Netlist** and **Write SDC File** by double-clicking these options in the **Tasks** pane of the TimeQuest Timing Analyzer window. Confirm that the above timing constraints were added to the SDC file linked with your design.
6. Run the Quartus II Fitter.
7. After the Quartus II Fitter operation completes, execute **Update Timing Netlist** by double-clicking this option in the **Tasks** pane of TimeQuest Timing Analyzer window.
8. Execute **Report Top Failing Paths** by double-clicking this option in the **Tasks** pane of the TimeQuest Timing Analyzer window.
9. Assuming all other paths in your design meet timing, one or more of the paths involving reset/powerdown ports might report timing violations. This is because the design is not able to meet the preliminary timing constraints of 4 ns (maximum delay) and 1.2 ns (minimum delay).
10. Note the slack in the timing report for all failing paths and adjust the maximum delay and the minimum delay values in the SDC file. Maintain a difference of 2.8 ns between the maximum delay and the minimum delay for each reset/powerdown port.
11. After adjusting the delay values, execute **Update Timing Netlist** and run the Quartus II Fitter again.
12. After the Quartus II Fitter operation completes, execute **Update Timing Netlist**.
13. Execute **Report Top Failing Paths** once again. If there are any failing paths involving the reset/powerdown ports, adjust the delay values in the SDC file and repeat the procedure until no failing paths are reported.

Consider the previous design example in which all unconstrained ports were identified. The following example shows how to set the constraints for the `gxb_powerdown` port. The same procedure must be followed for all other reset ports.

After setting the maximum and minimum delay for the `gxb_powerdown` port, the SDC file should have the following constraints:

```

*****
# Set Maximum Delay
*****

set_max_delay -from [get_keepers
{reset_seq_tx_rx_rx_cruclk_rx_clkout:inst2|gxb_powerd
own}] -to [get_ports
{PIPE_DataGen_Ch:inst|alt2gxb:alt2gxb_component|chann
el_quad[0].cent_unit~OBSERVABLEQUADRESET}] 4.000

*****
# Set Minimum Delay
*****

set_min_delay -from [get_keepers
{reset_seq_tx_rx_rx_cruclk_rx_clkout:inst2|gxb_powerd
own}] -to [get_ports
{PIPE_DataGen_Ch:inst|alt2gxb:alt2gxb_component|chann
el_quad[0].cent_unit~OBSERVABLEQUADRESET}] 1.200
    
```

After running the Quartus II fitter with the above timing constraints for the `gxb_powerdown` port, the following slack is reported on this path after executing **Report Top Failing Paths** (Figure 2-166).

Figure 2-166. Slack Reported for the `gxb_powerdown` Port

SLACK: -0.798 ns (VIOLATED)

Path Summary	
Property ▾	Value
1 To Node	PIPE_DataGen_Ch:inst alt2gxb:alt2gxb_component channel_quad[0].cent_unit~OBSERVABLEQUADRESET
2 Slack	-0.798 (VIOLATED)
3 Launch Clock	cal_blk_clk
4 Latch Clock	n/a
5 From Node	reset_seq_tx_rx_rx_cruclk_rx_clkout:inst2 gxb_powerdown
6 Data Required Time	4.000
7 Data Arrival Time	4.798

Since the data arrival time is later than the data required time by 0.798 ns, the maximum delay and minimum delay should both be incremented by 0.8 ns in the SDC file. The new SDC file should have the following modified constraints for the `gxb_powerdown` port.

```

*****
# Set Maximum Delay
*****

set_max_delay -from [get_keepers
{reset_seq_tx_rx_rx_cruclk_rx_clkout:inst2|gxb_powerd
own}] -to [get_ports
{PIPE_DataGen_Ch:inst|alt2gxb:alt2gxb_component|chann
el_quad[0].cent_unit~OBSERVABLEQUADRESET}] 4.8

*****
# Set Minimum Delay
*****

set_min_delay -from [get_keepers
{reset_seq_tx_rx_rx_cruclk_rx_clkout:inst2|gxb_powerd
own}] -to [get_ports
{PIPE_DataGen_Ch:inst|alt2gxb:alt2gxb_component|chann
el_quad[0].cent_unit~OBSERVABLEQUADRESET}] 2.000

```

After modifying the SDC file and running the Quartus II Fitter, the **Update Timing Netlist** option should be executed, followed by **Report Top Failing Paths**. If the `gxb_powerdown` port still shows in the failing paths, modify the slack appropriately in the SDC file and repeat the procedure until timing is met on this path.

Follow the same procedure to set timing constraints on all transceiver reset/powerdown ports in your design.



You should set constraints and meet timing for both fast and slow timing models. The same maximum and minimum delay constraints might not be able to meet timing for both timing models. This is acceptable as long as the skew is within the specified period (2.8 ns) for each path in the SDC file for each timing model.

Unconstrained Asynchronous ALT2GXB Ports

In the Quartus II software versions 7.1 and 7.1 sp1, TimeQuest does not automatically constrain transceiver asynchronous input/output ports. These ports are listed in [Table 2-54](#).

TimeQuest Port Name	ALT2GXB Port Name
ala2size	rx_ala2size
enapatternalign	rx_enapatternalign
bitslip	rx_bitslip
rlv	rx_rlv
invpol	rx_invpolarity
enabyteord	rx_enabyteord
pipe8b10binvpolarity	pipe8b10binvpolarity
revbitorderwa	rx_revbitorderwa
bisterr	rx_bisterr
bistdone	rx_bitstdone
phaselockloss	rx_pll_locked
freglock	rx_freqlocked
serialpbkben	rx_serialpbken

You must manually add the timing constraints in the SDC file for TimeQuest to analyze these paths. For these asynchronous ports, you only need to set a maximum delay constraint of 10 ns in the SDC file.

To identify all unconstrained ALT2GXB asynchronous ports, execute **Report Unconstrained Paths** in TimeQuest Timing Analyzer after running the Quartus II Fitter. Set a maximum delay of 10 ns for all such ports in the SDC file.

For example, if the rx_invpolarity signal is driven by the signal top_rx_invpolarity on an input pin, the SDC file constraint for this port should be set as:

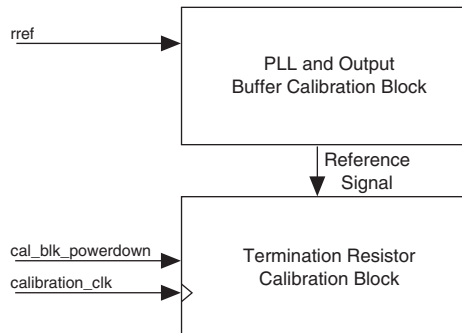
```
set_max_delay -from [get_ports {top_rx_invpolarity}]
-to [get_keepers
{xcvr_inst.receive~OBSERVABLEINVPOL}] 10.000
```

Follow the same procedure to constrain all asynchronous ALT2GXB ports in your design before closing timing analysis for your design.

Calibration Blocks

The Stratix II GX gigabit transceiver block contains calibration circuits to calibrate the on-chip termination, the PLLs, and the output buffers. The calibration circuits are divided into two main blocks: the PLL and output buffer calibration block and the termination resistor calibration block (refer to [Figure 2-167](#)). Each transceiver block contains a PLL and output buffer calibration block that calibrates the PLLs and output buffers within that particular transceiver block. Each device contains one termination resistor calibration block that calibrates all the termination resistors in the transceiver channels of the entire device.

Figure 2-167. Calibration Block



PLL and Output Buffer Calibration Block

Each Stratix II GX transceiver block contains a PLL and output buffer calibration circuit to counter the effects of PVT (process, voltage, and temperature) on the PLL and output buffer. Each transceiver block's calibration circuit uses a voltage reference derived from an external reference resistor. There is one reference resistor required for each active transceiver block in Stratix II GX devices. Unused transceiver block's (except the transceiver blocks feeding the termination resistor calibration block) can be left unconnected or be tied to the 3.3-V transceiver analog VCC (if the transceiver block's 3.3-V analog supply is connected to 3.3 V).

Termination Resistor Calibration Block

The Stratix II GX transceiver's on-chip termination resistors in the transceiver channels of the entire device are calibrated by a single calibration block. This block ensures that process, voltage, and temperature variations do not have an impact on the termination resistor value. There is only one termination resistor calibration block per device.

The calibration block uses the reference resistor of transceiver block 0 or transceiver block 1, depending on the device. The calibration block uses the reference resistor in transceiver block 0 for EP2SGX30 and EP2SGX60 devices and the reference resistor in transceiver block 1 for EP2SGX90 and EP2SGX130 devices. A reference resistor must be connected to either transceiver block 0 or transceiver block 1 to ensure proper operation of the calibration block, whether or not the transceiver block is in use. Failing to connect the reference resistor of the transceiver block feeding the calibration block results in incorrect termination values for all the termination resistors in the transceivers of the entire device.

The termination resistor calibration circuit requires a calibration clock. You can use a global clock line if the REFCLK pins are used for the reference clock. You can instantiate a calibration clock port in the MegaWizard to supply your own clock through the `cal_blk_clk` port.

The frequency range of the `cal_blk_clk` is 10 MHz to 125 MHz. If there are no slow-speed clocks available, use a divide-down circuit (for example, a ripple counter) to divide the available clock to a frequency in that range. The quality of the calibration clock is not an issue, so PLD local routing is sufficient to route the calibration clock.

For multiple ALT2GXB instances in the same device, if all the instances are the same, the calibration block must be active and the `cal_blk_clk` port of all instances must be tied to a common clock. Physically, there is one `cal_blk_clk` port per device. The Quartus II software provides an error message if the `cal_blk_clk` port is tied to different clock sources, because this would be impossible to fit into a device. If there are different configurations of the ALT2GXB instance, only one must have the calibration block instantiated. If multiple instances of the ALT2GXB custom megafunction variation have the calibration block instantiated, then all the `cal_blk_clk` ports must be tied to the same clock source.

The calibration block can be powered down through the optional `cal_blk_powerdown` port (this is an active low input). Powering down the calibration block during operations may yield transmit and receive data errors. Only use this port to reset the calibration block to initiate a recalibration of the termination resistors to account for variations in

temperature or voltage. The minimum pulse duration for this port is to be determined by characterization. If external termination is used on all signals, the calibration block in ALT2GXB need not be used.

Referenced Documents

This chapter references the following documents:

- *ALT2GXB Megafunction User Guide* chapter in volume 2 of the *Stratix II GX Device Handbook*
- *SDI MegaCore Function User Guide*
- *Specifications & Additional Information* chapter in volume 2 of the *Stratix II GX Handbook*.
- *Stratix II GX Dynamic Reconfiguration* chapter in volume 2 of the *Stratix II GX Device Handbook*

Document Revision History

Table 2–55 shows the revision history for this chapter.

<i>Table 2–55. Document Revision History (Part 1 of 6)</i>		
Date and Document Version	Changes Made	Summary of Changes
October 2007, v4.2	Updated: <ul style="list-style-type: none"> ● Figure 2–1 ● Figure 2–9 ● Figure 2–12 ● Figure 2–13 ● Figure 2–14 ● Figure 2–15 ● Figure 2–38 ● Figure 2–39 ● Figure 2–44 ● Figure 2–51 ● Figure 2–70 ● Figure 2–118 	—
	Updated: <ul style="list-style-type: none"> ● Table 2–1 ● Table 2–3 ● Table 2–6 ● Table 2–8 ● Table 2–18 ● Table 2–19 ● Table 2–24 	—

Table 2–55. Document Revision History (Part 2 of 6)		
Date and Document Version	Changes Made	Summary of Changes
	Updated: <ul style="list-style-type: none"> ● “Reverse Serial Pre-CDR Loopback” ● “(OIF) CEI-PHY Interface Mode” ● “Clock Synthesis” ● “Clock Multiplier Unit” ● “Receiver PLL” ● “Rate Matcher” ● “Transmitter Bit Reversal” ● “Receiver Common Mode” ● “Native Modes” ● “Basic Single-Width Mode” ● “Basic Double-Width Mode” ● “SONET/SDH Mode” ● “Receiver Bit Reversal” ● “Pattern Detector Module” ● “Channel Clock Distribution” ● “Low-Latency PIPE mode” ● “Synchronization (Word Aligner)” ● “TimeQuest Timing Analyzer” 	—
	Added: <ul style="list-style-type: none"> ● “Referenced Documents” ● “Serial Digital Interface (SDI) Mode” ● “Serial RapidIO Mode” ● “CPRI Mode” ● “DC Coupling” ● “Basic Single-Width Mode with x4 Clocking” 	—
	Added Table 2–2.	—
	Minor text edits.	—

Table 2–55. Document Revision History (Part 3 of 6)

Date and Document Version	Changes Made	Summary of Changes
August 2007, v4.1	Moved the Dynamic Reconfiguration section.	The Dynamic Reconfiguration section was moved to the <i>Stratix II GX Dynamic Reconfiguration</i> chapter in this handbook.
	Added Table 2–45.	—
	Added note to “Serializer” and “Deserializer”.	—
	Updated the “NTFS Fast Recovery IP (NFRI)” section.	—
	Updated: <ul style="list-style-type: none"> ● “Double-Width General Rate Matching” and Figure 2–73 in the same section ● “PCI Express Receiver Detect” ● “Receiver Detect” ● “Native Modes” (introduction) ● “XAUI Mode” ● “Manual 7-bit Alignment Mode” ● “Manual 8-bit Alignment Mode” ● “Manual 10-Bit Alignment Mode” ● “Manual 16-Bit Alignment Mode” ● “Manual 20-bit Alignment Mode” ● “Manual 32-Bit Alignment Mode” ● Figure 2–72 ● Figure 2–73 	—
	Updated Table 2–6 and Figure 2–62.	—
	Added the “TimeQuest Timing Analyzer” section.	—
	Added “Reverse Serial Pre-CDR Loopback” section.	—
February 2007, v4.0	Replaced old Dynamic Reconfiguration section with a new “Dynamic Reconfiguration” section.	—
	Added the “Document Revision History” section to this chapter.	—
	Updated Figures 2–1, 2–2, 2–4, 2–8, 2–9, 2–11, 2–12, 2–20, 2–118, 2–119, 2–120	—

Table 2–55. Document Revision History (Part 4 of 6)

Date and Document Version	Changes Made	Summary of Changes
	Added the following sections: <ul style="list-style-type: none"> ● “Transmitter Phase Compensation FIFO Error Flag” ● “Transmitter Force Disparity” ● “Transmitter Polarity Inversion” ● “Transmitter Bit Reversal” ● “Generic Receiver Polarity Inversion” ● “Receiver Bit Reversal” ● “Receiver Byte Reversal” ● “PLD-Controlled Byte Ordering” ● “Receiver Phase Compensation FIFO Error Flag” ● “Multiple Protocols and Data Rates in a Transceiver Block” ● “Low-Latency PIPE mode” ● “Design Flow” 	—
	Added PLD Interface Clock Resources section, including: <ul style="list-style-type: none"> ● Tables 2–21 through 2–26. 	—
	Updated Tables 2–1, 2–9, 2–45.	—
	Updated content from “Transmitter Modules” through “Byte Serializer”.	—
	Updated and moved PLD transceiver information to “Receiver Phase Compensation FIFO Error Flag”.	—
	Sections modified: <ul style="list-style-type: none"> ● “Receiver Detect” ● “Reverse Serial Loopback” ● “Signal Threshold Detection Circuit” ● “Parallel Loopback” ● “Clock Synthesis” ● “Double-Width Mode” ● “Inter-Transceiver Line Routing” ● “Serial Loopback” ● “Loopback Modes” ● “BIST in Single-Width Mode” ● “BIST in Double-Width Mode” ● “Transmitter Buffer” ● “Transmitter PLL Block” ● “Transmitter PLL Bandwidth Setting” ● “Transmitter Polarity Inversion” 	—

Table 2–55. Document Revision History (Part 5 of 6)

Date and Document Version	Changes Made	Summary of Changes
	622 Mbps was changed to 600 Mbps in: <ul style="list-style-type: none"> ● “Transmitter PLL Block” ● “Transmitter PLLs” ● “Serializer” ● “Receiver Buffer” ● “Receiver Common Mode” ● “Deserializer” 	—
	3.125 to 6.375 Gbps was changed to 1 to 6.375 Gbps in: <ul style="list-style-type: none"> ● “Serializer” ● “Deserializer” ● “Native Modes” 	—
	Updated note in “Normal Operation Phase” section.	—
	The Automatic Mode section was updated and changed to “Word Alignment Based on Byte Ordering”.	—
	Changed “bits[15..8]” to “bits[31..24]” in the “32-Bit Pattern Mode” section	—
	Changed “rx_outrx_dataout” to “rx_dataout” in the “GIGE Receiver Synchronization” section.	—
	Added new note to the “Dynamic Transmit Rate Switch” and “Dedicated Reference Clock Pin Specifications” sections.	—
	<ul style="list-style-type: none"> ● Changed V_{CCHTX} to V_{CCH} throughout the chapter. ● Changed TX V_{CM} to V_{CM} throughout the chapter. ● Changed VCC_H to VCCH throughout the chapter. 	—
June 2006, v3.2	<ul style="list-style-type: none"> ● Minor change to Figure 2–1. ● Updated Table 2–1. ● Updated Figures 2–90 and 2–96. ● Added “NTFS Fast Recovery IP (NFRI)” section. 	<ul style="list-style-type: none"> ● Updated descriptions for rx_errdetect and cal_blk_powerdown in Table 2–1.

Table 2–55. Document Revision History (Part 6 of 6)

Date and Document Version	Changes Made	Summary of Changes
April 2006, v3.1	<ul style="list-style-type: none"> ● Added “Dedicated Reference Clock Pin Specifications” section, including Table 2–3 and Figure 2–3. ● Updated Figures 2–9, 2–16, 2–21, 2–22, 2–30, 2–35, 2–52, 2–53, 2–57, 2–68, 2–69, 2–73, 2–83, 2–84, 2–109, 2–110, and 2–118. ● Updated data rate in “Deserializer” section. ● Added “7-bit Alignment Mode” section. ● Removed references to the <code>rx_runningdisp</code> port. ● Updated “Manual SONET Alignment Mode (Two Consecutive 8-bit Characters (A1A2) or Four Consecutive 8-bit Characters (A1A1A2A2))” section. ● Updated “Manual Alignment Modes” section. ● Updated “Code Error Detect” section. ● Updated “Disparity Error Detector” section. ● Added “Reset Sequence for PIPE Mode” section, including Figures 2–119 and 2–120. ● Updated “Dynamic Reconfiguration Setup for alt2gxb Instance” section. ● Updated Table 2–34. 	<ul style="list-style-type: none"> ● Updated <code>tx_preemp_2t</code> port description in Table 2–34.
February 2006, v3.0	<ul style="list-style-type: none"> ● Updated technical content throughout chapter. ● Added “Dynamic Reconfiguration” section. 	—
December 2005, v2.0	Updated technical content throughout chapter.	—
October 2005 v1.0	Added chapter to the <i>Stratix II GX Device Handbook</i> .	—