Chapter Revision Dates .......................................................... vii

Section 1. Transceiver Architecture for Arria II Devices
Revision History ............................................................... 1–1

Chapter 1. Transceiver Architecture for Arria II Devices
Transceiver Block Overview ............................................... 1–3
Clock Multiplier Units (CMU) ............................................. 1–6
CMU PLL ........................................................................ 1–9
CMU0 Clock Divider ........................................................ 1–10
Transmitter Channel Local Clock Divider Block ................... 1–10
Transceiver Channel Architecture .................................... 1–11
Transmitter Channel Datapath ............................................. 1–12
Transmitter PCS .............................................................. 1–12
TX Phase Compensation FIFO .......................................... 1–12
Byte Serializer ............................................................. 1–14
8B/10B Encoder ............................................................. 1–15
Transmitter PMA ............................................................ 1–18
Serializer ......................................................................... 1–19
Transmitter Output Buffer ................................................. 1–20
Receiver Channel Datapath ............................................... 1–21
Receiver PMA ................................................................. 1–21
Receiver Input Buffer ...................................................... 1–22
CDR .............................................................................. 1–26
Deserializer ................................................................. 1–29
Receiver PCS ................................................................. 1–30
Word Aligner ................................................................. 1–30
Deskew FIFO ................................................................. 1–41
Rate-Match FIFO ........................................................... 1–41
8B/10B Decoder ............................................................... 1–42
Byte Deserializer ............................................................ 1–43
Byte Ordering ................................................................. 1–44
RX Phase Compensation FIFO ........................................ 1–46
Calibration Block ............................................................ 1–47
PCle Hard IP Block .......................................................... 1–50
Functional Modes ........................................................... 1–50
Basic Mode ................................................................. 1–51
Deterministic Latency ...................................................... 1–56
GbE .............................................................................. 1–57
Idle Ordered-Set Generation in GbE Mode ......................... 1–59
GbE Mode Reset Condition ............................................ 1–60
Word Aligner in GbE Mode .............................................. 1–60
Rate Match FIFO in GbE Mode .......................................... 1–61
PCle Mode .................................................................... 1–62
Power State Management ................................................. 1–65
Transmitter Buffer Electrical Idle .................................... 1–66
Receiver Detection ......................................................... 1–67
Compliance Pattern Transmission Support ....................... 1–68
# Chapter 3. Configuring Multiple Protocols and Data Rates in Arria II Devices

- Transceiver PLL Configurations .................................................. 3–1
- Creating Transceiver Channel Instances ...................................... 3–2
- General Requirements to Combine Channels ................................. 3–2
  - Control Signals ........................................................................ 3–2
  - Calibration Clock and Power Down ........................................... 3–2
- Sharing CMU PLLs ....................................................................... 3–3
  - Multiple Channels Sharing a CMU PLL ..................................... 3–3
    - Example 1 ............................................................................ 3–3
  - Multiple Channels Sharing Two CMU PLLs ............................... 3–6
    - Example 2 ............................................................................ 3–6
- Combining Receiver Only Channels .............................................. 3–8
- Combining Transmitter Channel and Receiver Channel Instances ........ 3–9
  - Multiple Transmitter Channel and Receiver Channel Instances ... 3–9
    - Example 3 ............................................................................ 3–9
- Combining Channels Configured in Protocol Functional Modes .......... 3–10
  - Basic ×4 Mode ....................................................................... 3–10
  - Combining Channels Using the PCIe hard IP Block with Other Channels 3–12
- Combining Transceiver Instances Using PLL Cascade Clocks .......... 3–12
- Combining Transceiver Instances in Multiple Transceiver Blocks ........ 3–13
  - Example 4 ............................................................................ 3–13
- Summary .................................................................................. 3–15
- Document Revision History ....................................................... 3–16

# Chapter 4. Reset Control and Power Down in Arria II Devices

- User Reset and Power-Down Signals ........................................... 4–1
  - Blocks Affected by Reset and Power-Down Signals .................. 4–3
- Transceiver Reset Sequences ..................................................... 4–4
  - All Supported Functional Modes Except PCIe Functional Mode ... 4–6
  - Bonded Channel Configuration ................................................ 4–6
  - Non-Bonded Channel Configuration .......................................... 4–10
  - PCIe Functional Mode ............................................................ 4–15
  - PCIe Reset Sequence ............................................................. 4–15
- Dynamic Reconfiguration Reset Sequences .................................. 4–17
  - Reset Sequence with Data Rate Division in the TX Option ........... 4–17
  - Reset Sequence with the Channel and TX PLL Select/Reconfig Option 4–18
- Hot Socketing Reset Sequence .................................................. 4–19
- Power Down ............................................................................ 4–20
- Simulation Requirements ......................................................... 4–21
- Document Revision History ....................................................... 4–21

# Additional Information

- How to Contact Altera .............................................................. Info–1
- Typographic Conventions ......................................................... Info–1
Chapter 1. Transceiver Architecture in Arria II Devices
Revised: February 2015
Part Number: AIIGX52001-4.6

Chapter 2. Transceiver Clocking in Arria II Devices
Revised: June 2011
Part Number: AIIGX52002-3.1

Chapter 3. Configuring Multiple Protocols and Data Rates in Arria II Devices
Revised: December 2010
Part Number: AIIGX52003-3.0

Chapter 4. Reset Control and Power Down in Arria II Devices
Revised: July 2013
Part Number: AIIGX52004-3.2
This section provides information about Arria® II device family transceiver architecture and clocking. It also describes configuring multiple protocols, data rates, and reset control and power down in the Arria II device family. This section includes the following chapters:

- Chapter 1, Transceiver Architecture in Arria II Devices
- Chapter 2, Transceiver Clocking in Arria II Devices
- Chapter 3, Configuring Multiple Protocols and Data Rates in Arria II Devices
- Chapter 4, Reset Control and Power Down in Arria II Devices

Revision History

Refer to each chapter for its own specific revision history. For information about when each chapter was updated, refer to the Chapter Revision Dates section, which appears in this volume.
This chapter describes all available modules in the Arria® II GX and GZ transceiver architecture and describes how these modules are used in the protocols shown in Table 1–1. In addition, this chapter lists the available test modes, dynamic reconfiguration, and ALTGX port names.

Arria II GX and GZ devices provide up to 24 full-duplex clock data recovery-based (CDR) transceivers with physical coding sublayer (PCS) and physical medium attachment (PMA), and support the serial protocols listed in Table 1–1 and Table 1–2. Table 1–1 lists the serial protocols for Arria II GX devices.

### Table 1–1. Serial Protocols for Arria II GX Devices

<table>
<thead>
<tr>
<th>Protocol Description</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCI Express® (PIPE) (PCIe®)</td>
<td>Gen1, 2.5 Gbps</td>
</tr>
<tr>
<td>Serial RapidIO® (SRIO)</td>
<td>1.25 Gbps, 2.5 Gbps, and 3.125 Gbps</td>
</tr>
</tbody>
</table>
| Serial ATA (SATA)/ Serial Attached SCSI (SAS) | ■ SATA I, 1.5 Gbps  
■ SATA II, 3.0 Gbps  
■ SATA III, 6.0 Gbps  
■ SAS, 1.5 Gbps and 3.0 Gbps |
| Serial Digital Interface (SDI) | ■ HD-SDI, 1.485 Gbps and 1.4835 Gbps  
■ 3G-SDI, 2.97 Gbps and 2.967 Gbps |
| ASI | 270 Mbps |
| Common Public Radio Interface (CPRI) | 614.4 Mbps, 1228.8 Mbps, 2457.6 Mbps, 3072 Mbps, 4915.2 Mbps, and 6144 Mbps |
| OBSAI | 768 Mbps, 1536 Mbps, 3072 Mbps, and 6144 Mbps |
| Gigabit Ethernet (GbE) | 1.25 Gbps |
| XAUI | 3.125 Gbps to 3.75 Gbps for HiGig/HiGig+ support |
| SONET/SDH | ■ OC-12 (622 Mbps)  
■ OC-48 (2.488 Gbps) |
| GPON | 1.244 uplink and 2.488 downlink |
| SerialLite II | 0.6 Gbps to 3.75 Gbps |
| Interlaken | — |
| CEI | — |
Table 1–2 lists the serial protocols for Arria II GZ devices.

### Table 1–2. Serial Protocols for Arria II GZ Devices

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIe</td>
<td>Gen2, 5.0 Gbps</td>
</tr>
<tr>
<td>SRIO</td>
<td>1.25 Gbps, 2.5 Gbps, and 3.125 Gbps</td>
</tr>
<tr>
<td>Serial ATA (SATA)/ Serial Attached SCSI (SAS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SATA I, 1.5 Gbps</td>
</tr>
<tr>
<td></td>
<td>SATA II, 3.0 Gbps</td>
</tr>
<tr>
<td></td>
<td>SATA III, 6.0 Gbps</td>
</tr>
<tr>
<td></td>
<td>SAS, 1.5 Gbps and 3.0 Gbps</td>
</tr>
<tr>
<td>Serial Digital Interface (SDI)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HD-SDI, 1.485 Gbps and 1.4835 Gbps</td>
</tr>
<tr>
<td></td>
<td>3G-SDI, 2.97 Gbps and 2.967 Gbps</td>
</tr>
<tr>
<td>ASI</td>
<td>270 Mbps</td>
</tr>
<tr>
<td>Common Public Radio Interface (CPRI)</td>
<td>614.4 Mbps, 1228.8 Mbps, 2457.6 Mbps, 3072 Mbps, 4915.2 Mbps, and 6144 Mbps</td>
</tr>
<tr>
<td>OBSAI</td>
<td>768 Mbps, 1536 Mbps, 3072 Mbps, and 6144 Mbps</td>
</tr>
<tr>
<td>Gigabit Ethernet (GbE)</td>
<td>1.25 Gbps</td>
</tr>
<tr>
<td>XAUI</td>
<td>3.125 Gbps to 3.75 Gbps for HiGig/HiGig+ support</td>
</tr>
<tr>
<td>SONET/SDH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OC-12 (622 Mbps)</td>
</tr>
<tr>
<td></td>
<td>OC-48 (2.488 Gbps)</td>
</tr>
<tr>
<td></td>
<td>OC-96 (4.976 Gbps)</td>
</tr>
<tr>
<td>GPON</td>
<td>1.244 uplink and 2.488 downlink</td>
</tr>
<tr>
<td>Interlaken</td>
<td>40G with 10 channels at 6.375 Gbps</td>
</tr>
<tr>
<td>CEI</td>
<td>6.375 Gbps</td>
</tr>
</tbody>
</table>

You can implement these protocols through the ALTGX MegaWizard™ Plug-In Manager, which also offers the highly flexible Basic functional mode to implement proprietary serial protocols.
Transceiver Block Overview

Arria II GX devices offer two to four transceiver blocks per device while Arria II GZ devices offer up to six transceiver blocks. Each block consists of four fully-duplex (transmitter and receiver) channels, located on the left side of the device (in a die-top view).

Figure 1–1 shows the die-top view of the transceiver block locations in Arria II GX devices.

Figure 1–1. Transceiver Channels for Arria II GX Devices
Figure 1–2 shows the die top view of the transceiver block locations in Arria II GZ devices.

Figure 1–2. Transceiver Channels for Arria II GZ Devices
Figure 1–3 shows the block diagram of the transceiver block architecture for Arria II GX and GZ devices.

The following sections describe all the modules of the transceiver blocks. The input and output ports of these modules are described in the module sections, and are listed in the “Transceiver Port List” on page 1–94.
Clock Multiplier Units (CMU)

Each transceiver block contains two CMU blocks, which contain a CMU phase-locked loop (PLL) that provides clocks to all the transmitter channels in the same transceiver block. These two CMU blocks can provide two independent high-speed clocks per transceiver block.

The CMU PLL is also known as the TX PLL.

The CMU PLLs in CMU0 and CMU1 are identical and each transmitter channel in the transceiver block can receive a high-speed clock from either of the two CMU PLLs. However, the CMU0 block has an additional clock divider after the CMU0 PLL to support bonded functional modes where multiple channels share a common clock to reduce skew between the channels. With the ALTGX MegaWizard Plug-In Manager, you can select the bonded functional modes used in ×4 Basic, PCIe, and XAUI.
Figure 1–4 shows a top-level block diagram of the connections between the CMU blocks and the transceiver channels.

**Figure 1–4. Top-Level Diagram of CMU Block Connections in a Transceiver Block**

Notes to Figure 1–4:

1. Clocks provided to support bonded channel functional mode.
2. For more information, refer to the Transceiver Clocking for Arria II Devices chapter.
Figure 1–5 and Figure 1–6 show the top-level block diagram of CMU0 and CMU1 blocks, respectively.

Figure 1–5. CMU0 Block Diagram

Notes to Figure 1–5:
(1) Although each CMU PLL has its own \texttt{pll\_powerdown} port, the ALTGX MegaWizard Plug-In Manager instantiation provides only one port per transceiver block. This port power downs one or both CMU PLLs (if used).
(2) The inter-transceiver block (ITB) clock lines shown are the maximum value. The actual number of ITB lines in your device depends on the number of transceiver blocks on one side of the device.
(3) There is one \texttt{pll\_locked} signal per CMU PLL.
(4) Used in \times4, \times8, and XAUI functional modes. In \times8 functional mode, only the \texttt{CMU0} channel of the master transceiver block provides clock output to all eight transceiver channels configured in PCIe functional mode.

Figure 1–6. CMU1 Block Diagram

Notes to Figure 1–6:
(1) Although each CMU PLL has its own \texttt{pll\_powerdown} port, the ALTGX MegaWizard Plug-In Manager instantiation provides only one port per transceiver block. This port power downs one or both CMU PLLs (if used).
(2) The ITB clock lines shown are the maximum value. The actual number of ITB lines in your device depends on the number of transceiver blocks on one side of the device.
(3) There is one \texttt{pll\_locked} signal per CMU PLL.
CMU PLL

Figure 1–7 shows the block diagram of the CMU PLL.

Figure 1–7. Diagram of the CMU PLL

Notes to Figure 1–7:
(1) The ITB clock lines shown are the maximum value. The actual number of ITB lines in your device depends on the number of transceiver blocks on one side of the device.
(2) Although each CMU PLL has its own pll_powerdown port, the ALTGX MegaWizard Plug-In Manager instantiation provides only one port per transceiver block. This port power downs one or both CMU PLLs (if used).
(3) There is one pll_locked signal per CMU PLL.

For more information about input reference clocks, refer to the “CMU PLL and Receiver CDR Input Reference Clocks” section of the Transceiver Clocking in Arria II Devices chapter.

The phase frequency detector (PFD) in the CMU PLL tracks the voltage-controlled oscillator (VCO) output with the input reference clock. This VCO runs at half the serial data rate. The CMU PLL generates the high-speed clock from the input reference clock through the two divider blocks (/M and /L) in the feedback path. Table 1–3 lists the available /M and /L settings, which are set automatically in the Quartus® II software, based on the input reference clock frequency and serial data rate.

Table 1–3. Multiplier Block Heading to Clock Divider for Arria II Devices

<table>
<thead>
<tr>
<th>Multiplier Block</th>
<th>Available Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>/M</td>
<td>1, 4, 5, 8, 10, 16, 20, 25</td>
</tr>
<tr>
<td>/L</td>
<td>1, 2, 4</td>
</tr>
</tbody>
</table>

You can set the PLL bandwidth in the ALTGX megafuction.

The high-speed clock output from the CMU PLL is forwarded to the CMU0 clock divider block in bonded functional modes and the transmitter channel local clock divider block in non-bonded functional modes. The output of either clock divider block provides clocks for the PCS and PMA blocks.

For more information about using two CMU PLLs to configure multiple transmitter channels, refer to the Configuring Multiple Protocols and Data Rates in Arria II Devices chapter.
CMU0 Clock Divider

The clock divider is only available only in the CMU0 block and is used in bonded functional modes.

Figure 1–8 shows a diagram of the CMU0 clock divider block.

**Figure 1–8. CMU0 Clock Divider Block** *(Note 1)*

Notes to Figure 1–8:

1. The Quartus II software automatically selects all the divider settings based on the input clock frequency, data rate, deserialization width, and channel width settings.

2. The high-speed serial clock is available to all the transmitter channels in the transceiver block. In a ×8 configuration, only the CMU0 clock divider of the master transceiver block provides the high-speed serial clock to all eight channels.

3. If the byte serializer block is enabled in bonded channel modes, the coreclkout clock output is half the frequency of the low-speed parallel clock. Otherwise, the coreclkout clock output is the same frequency as the low-speed parallel clock.

Transmitter Channel Local Clock Divider Block

Each transmitter channel contains a local clock divider block used automatically by the Quartus II software for non-bonded functional modes (for example, ×1 PCIe, Gbps Ethernet (GbE), SONET/SDH, and SDI mode). This block allows each transmitter channel to run at /1, /2, or /4 of the CMU PLL output data rate.

Figure 1–9 shows the transmitter local clock divider block.

**Figure 1–9. Transmitter Local Clock Divider Block**

For more information about transceiver channel local clock divider block clocking, refer to the “Transceiver Channel Datapath Clocking” section in the Transceiver Clocking in Arria II Devices chapter.
Transceiver Channel Architecture

Each transceiver channel consists of a transmitter channel and a receiver channel. Each transmitter or receiver channel comprises the channel PCS and channel PMA blocks. Figure 1–10 shows the Arria II GX and GZ transceiver channel architecture.

The FPGA fabric-to-transceiver interface and the PMA-to-PCS interface can support an 8, 10, 16, or 20 bit-width data bus.

The transceiver channel is available in two modes:

- Single-width mode—In this mode, the PMA-to-PCS interface uses an 8- or 10-bit wide data bus. The FPGA fabric-to-transceiver interface supports an 8- or 10-bit wide data bus, with the byte serializer/deserializer disabled. When the byte serializer/deserializer is enabled, the FPGA fabric-to-transceiver interface supports a 16 or 20 bit-width data bus.

- Double-width mode—In this mode, both the PMA-to-PCS interface and the FPGA fabric-to-transceiver uses an 16- and 20-bit wide data bus. The byte serializer/deserializer is supported in Arria II GZ devices, but not in Arria II GX devices. This mode is only supported for BASIC or Deterministic Latency protocol, used for CPRI and OBSAI interfaces.
Transmitter Channel Datapath

This section describes the Arria II GX and GZ transmitter channel datapath architecture. The sub-blocks in the transmitter datapath are described in order from the transmitter (TX) phase compensation FIFO buffer at the FPGA fabric-to-transceiver interface to the transmitter input buffer.

Figure 1–11 shows the transmitter channel datapath.

**Transmitter PCS**

This section describes the transmitter PCS modules, which consists of the TX phase compensation FIFO, byte serializer, and 8B/10B encoder.

The tx_digitalreset signal resets all modules in the transmitter PCS block.

For more information about the tx_digitalreset signal, refer to the Reset Control and Power Down in Arria II Devices chapter.

**TX Phase Compensation FIFO**

This FIFO compensates for the phase difference between the low-speed parallel clock and the FPGA fabric interface clock. Table 1–4 lists the available modes for the TX phase compensation FIFO.

<table>
<thead>
<tr>
<th>Mode</th>
<th>FIFO Depth</th>
<th>Latency Through FIFO</th>
<th>Applicable Functional Modes (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Latency</td>
<td>4-words deep</td>
<td>2-to-3 parallel clock cycles</td>
<td>All functional modes except PCIe and Deterministic Latency</td>
</tr>
<tr>
<td>High Latency</td>
<td>8-words deep</td>
<td>4-to-5 parallel clock cycles</td>
<td>PCIe</td>
</tr>
<tr>
<td>Register</td>
<td>—</td>
<td>1</td>
<td>Deterministic Latency</td>
</tr>
</tbody>
</table>

*Note to Table 1–4:*

(1) Automatically set when you select a protocol in the ALTGX MegaWizard Plug-In Manager.
Figure 1–12 shows the datapath and clocking of the TX phase compensation FIFO.

**Figure 1–12. TX Phase Compensation FIFO**

**Notes to Figure 1–12:**
1. The `tx_phase_comp_fifo_error` is optional and available in all functional modes. This signal is asserted high to indicate an overflow or underflow condition.
2. Use this optional clock for the FIFO write clock if you instantiate the `tx_coreclk` port in the ALTGX MegaWizard Plug-In Manager, regardless of the channel configurations. Otherwise, the same clock used for the read clock is also used for the write clock. Ensure that there is 0 parts per million (PPM) frequency difference between the `tx_coreclk` clock and the read clock of the FIFO.
3. The `tx_clkout` low-speed parallel clock is from the local clock divider from the associated transmitter channel and is used in non-bonded configurations.
4. The `coreclkout` clock is from the CMUX block of the associated transceiver block or the master transceiver block for ×4 bonded or ×8 bonded channel configurations, respectively.

For more information about TX phase compensation FIFO clocking, refer to the “Limitation of the Quartus II Software-Selected Transmitter Phase Compensation FIFO Write (or Read) Clocks” section in the Transceiver Clocking in Arria II Devices chapter.

An optional `tx_phase_comp_fifo_error` port is available in all functional modes and is asserted high in an overflow or underflow condition. If this signal is asserted, ensure that there is 0 PPM difference between the TX phase compensation FIFO read and write clocks.

The output of this block can go to any of the following blocks:
- Byte serializer—If you enable this block.
- 8B/10B encoder—If you disable the byte serializer, but enable the 8B/10B encoder and your channel width is either 8 or 16 bits.
- Serializer—If you disable both the byte serializer and the 8B/10B encoder, or if you use low-latency PCS bypass mode.
**Byte Serializer**

In Arria II GX devices, you cannot enable the byte serializer in double-width mode. However, in Arria II GZ devices, you can enable both double-width and the byte serializer to achieve a 32- or 40-bit PCS-FPGA interface.

Figure 1–13 shows the byte serializer datapath for Arria II GX devices.

**Figure 1–13. Byte Serializer Datapath for Arria II GX Devices**

The byte serializer divides the input datapath width by two. This allows you to run the transceiver channel at higher data rates while keeping the FPGA fabric frequency within the maximum limit. This module is required in configurations that exceed the FPGA fabric-to-transceiver interface clock upper frequency limit. It is optional in configurations that do not exceed the FPGA fabric-to-transceiver interface clock upper frequency limit.

For example, if you want to run the transceiver channel at 3.125 Gbps, without the byte serializer, the FPGA fabric interface clock frequency must be 312.5 MHz (3.125 Gbps/10), which violates the FPGA fabric interface frequency limit. When you use the byte serializer, the FPGA fabric interface frequency is 156.25 MHz (3.125 Gbps/20).

For more information about the maximum frequency limit for the FPGA fabric-to-transceiver interface, refer to the Device Datasheet for Arria II Devices.

The byte serializer forwards the data from the TX phase compensation FIFO LSByte first. For example, assuming a channel width of 20 bits, the byte serializer sends out the least significant word `data_in[9:0]` of the parallel data from the FPGA fabric, followed by `data_in[19:10]`.

The data from the byte serializer is forwarded to the 8B/10B encoder if the module is enabled and the input data width is 16 bits. Otherwise, the output is forwarded to the serializer module in the transceiver PMA block.
8B/10B Encoder

Figure 1–14 shows the inputs and outputs of the 8B/10B encoder.

The 8B/10B encoder generates 10-bit code groups from the 8-bit data and 1-bit control identifier. If the tx_ctrlenable input is high, the 8B/10B encoder translates the 8-bit input data to a 10-bit control word (Kx.y). Otherwise, the 8B/10B encoder translates the 8-bit input data to a 10-bit data word (Dx.y). Figure 1–15 shows an example of how the second 8'hBC data is encoded as a control word, while the reset of the data are encoded as a data word.

The IEEE 802.3 8B/10B encoder specification identifies only a set of 8-bit characters for which tx_ctrlenable should be asserted. If you assert tx_ctrlenable for any other set of characters, the 8B/10B encoder might encode the output 10-bit code as an invalid code (it does not map to a valid Dx.y or Kx.y code), or an unintended valid Dx.y code, depending on the value entered. It is possible for a downstream 8B/10B decoder to decode an invalid control word into a valid Dx.y code without asserting any code error flags. Altera recommends not asserting tx_ctrlenable for unsupported 8-bit characters.
Figure 1–16 shows the conversion format. The LSB is transmitted first by default. You can, however, enable the Transmitter Bit Reversal option in the ALTGX MegaWizard Plug-In Manager to allow reversing the transmit bit order (MSB first) before it is forwarded to the serializer.

During reset, the running disparity and data registers are cleared. Also, the 8B/10B encoder outputs a K28.5 pattern from the RD- column continuously until tx_digitalreset is de-asserted. The input data and control code from the FPGA fabric is ignored during the reset state. After power up or reset, the 8B/10B encoder starts with a negative disparity (RD-) and transmits three K28.5 code groups for synchronization before it starts encoding and transmitting data on its output.

While tx_digitalreset is asserted, the downstream 8B/10B decoder that receives the data might observe synchronization or disparity errors.

In Basic functional mode, you can use the tx_forcedisp and tx_dispval ports to control the running disparity of the output from the 8B/10B encoder. Forcing disparity can either maintain the current running disparity calculations if the forced disparity value (on the tx_dispval bit) happens to match the current running disparity, or flip the current running disparity calculations if it does not match. If the forced disparity flips the current running disparity, the downstream 8B/10B decoder might detect a disparity error.
Table 1–5 lists the _tx_forcedisp_ and _tx_dispval_ port values and the effects they have on the data.

**Table 1–5. _tx_forcedisp_ and _tx_dispval_ Port Values for Arria II Devices**

<table>
<thead>
<tr>
<th><em>tx_forcedisp</em></th>
<th><em>tx_dispval</em></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>Current running disparity has no change.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Encoded data has positive disparity.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Encoded data has negative disparity.</td>
</tr>
</tbody>
</table>

Figure 1–18 shows an example of _tx_forcedisp_ and _tx_dispval_ port use, where data is shown in hexadecimal radix.

**Figure 1–18. 8B/10B Encoder Force Running Disparity Operations**

In this example, a series of K28.5 code groups are continuously sent. The stream alternates between a positive running disparity K28.5 (RD+) and a negative running disparity K28.5 (RD-) to maintain a neutral overall disparity. The current running disparity at time _n_ + 3 indicates that the K28.5 in time _n_ + 4 must be encoded with a negative disparity. Because _tx_forcedisp_ is high at time _n_ + 4, and _tx_dispval_ is also high, the K28.5 at time _n_ + 4 is encoded as a positive disparity code group.

The optional _tx_invpolarity_ port is available in all functional modes to dynamically enable the transmitter polarity inversion feature as a workaround to board re-spin or a major update to the FPGA fabric design when the positive and negative signals of a serial differential link are accidentally swapped during board layout.

A high value on the _tx_invpolarity_ port inverts the polarity of every bit of the input data word to the serializer in the transmitter datapath. Correct data is seen by the receiver, because inverting the polarity of each bit has the same effect as swapping the positive and negative signals of the differential link. The _tx_invpolarity_ signal is dynamic and might 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 1–19 shows an example result with the `tx_inv polarity` feature in a 10-bit wide datapath configuration.

**Figure 1–19. Transmitter Polarity Inversion**

<table>
<thead>
<tr>
<th>Output from Transmitter PCS</th>
<th>Converted Data Output to the Transmitter Serializer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 MSB</td>
<td>1 MSB</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 tx_inv polarity = high</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 LSB</td>
<td>1 LSB</td>
</tr>
</tbody>
</table>

**Transmitter PMA**

This section describes the transmitter PMA modules that consist of the serializer and the transmitter output buffer.

The `tx_analogreset` signal resets all modules in the transmitter PMA block. For more information about this signal, refer to the *Reset Control and Power Down in Arria II Devices* chapter.
Serializer

The serializer converts the incoming low-speed parallel signal from the transceiver PCS to the high-speed serial data and sends its LSB first to the transmitter output buffer. Figure 1–20 shows the serializer block diagram in an 8-bit PCS-to-PMA interface.

Figure 1–20. Serializer Block in an 8-Bit PCS-PMA Interface

Note to Figure 1–20:
(1) This clock is provided by the CMU0 clock divider of the master transceiver block and is only used in ×8 mode.

Figure 1–21 shows an example of serialized data with a 8'b01101010 value.

Figure 1–21. Serializer Bit Order (Note 1)

Note to Figure 1–21:
(1) The input data to the serializer is 8 bits (channel width = 8 bits with the 8B/10B encoder disabled).
Transmitter Output Buffer

The Arria II GX and GZ transmitter output buffers support the 1.5-V pseudo current mode logic (PCML) I/O standard and can drive 40 inches of FR4 trace (with 50-Ω impedance) across two connectors.

For data rates > 3.75 Gbps, Altera recommends limiting the FR4 trace length to 15 inches.

The transmitter output buffer power supply (VCCH) only provides voltage to the transmitter output buffers in the transceiver channels. This is set to 1.5 V in the ALTGX MegaWizard Plug-In Manager. The common mode voltage (VCM) for the Arria II GX and GZ transmitter output buffers is 650 mV.

To improve signal integrity, the transmitter output buffer has the following additional circuitry, which you can set in the ALTGX MegaWizard Plug-In Manager:

- Programmable differential output voltage (VOD)—This feature allows you to customize the VOD to handle different trace lengths, various backplanes, and various receiver requirements.

- Programmable pre-emphasis—Pre-emphasis boosts high frequencies in the transmit data signal, which might be attenuated in the transmission media because of data-dependent jitter and other intersymbol interference (ISI) effects. It equalizes the frequency response at the receiver so the differences between the low-frequency and high-frequency components are reduced, minimizing the ISI effects from the transmission medium.

  Pre-emphasis requirements increase as data rates through legacy backplanes increase. Using pre-emphasis can maximize the data eye opening at the far-end receiver.

- Programmable differential on-chip termination (OCT)—The Arria II GX transmitter buffer includes a differential OCT of 100 Ω. The Arria II GZ transmitter buffer includes a differential OCT of 85, 100, 120, and 150 Ω. The resistance is adjusted in the calibration block to compensate for temperature, voltage, and process changes (for more information, refer to “Calibration Block” on page 1-47).

  You can set the transmitter termination setting in the ALTGX MegaWizard Plug-In Manager or through the Quartus II Assignment Editor by setting the assignment output termination to 85, 100, 120, or 150 for Arria II GZ devices or 100 for Arria II GX devices on the transmitter output buffer.

  You can disable OCT and use external termination. In this case, the transmitter common mode is tri-stated.

The Arria II GX and GZ transmitter output buffers in the transceiver block are current-mode drivers. The resulting VOD is a function of the transmitter termination value.
Receiver-detect capability to support PCIe functional 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. For more information, refer to “PCIe Mode” on page 1–62.

Tristate-able transmitter buffer to support PCIe electrical idle—This feature is only active in PCIe mode to work hand-in-hand with the receiver-detect capability. For more information, refer to “PCIe Mode” on page 1–62.

For more information about the available settings in each feature, refer to the Device Datasheet for Arria II Devices.

Figure 1–22 shows the transmitter output buffer block diagram.

**Figure 1–22. Transmitter Output Buffer**

---

**Receiver Channel Datapath**

This section describes the Arria II GX and GZ receiver channel datapath architecture. The sub-blocks in the receiver datapath are described in order from the receiver input buffer to the receiver (RX) phase compensation FIFO buffer at the FPGA fabric-to-transceiver interface.

Figure 1–23 shows the receiver channel datapath in Arria II GX and GZ devices.

**Figure 1–23. Receiver Channel Datapath**

---

**Receiver PMA**

This section describes the receiver PMA modules, which consists of the receiver input buffer, CDR, and deserializer.

The rx_analogreset signal resets all modules in the receiver PMA block. For more information about this signal, refer to the Reset Control and Power Down in Arria II Devices chapter.
Receiver Input Buffer

The receiver input buffer receives serial data from the rx_datain port and feeds it to the CDR unit. Figure 1–24 shows the receiver input buffer.

Figure 1–24. Receiver Input Buffer for Arria II GX Devices

Notes to Figure 1–24:
1. For more information about reverse serial pre-CDR loopback mode, refer to “Test Modes” on page 1–85.
2. For Arria II GZ devices, this is 85, 100, 120, and 150 Ω.

Table 1–6 lists the electrical features supported by the receiver input buffer.

Table 1–6. Electrical Features Supported by the Receiver Input Buffer for Arria II Devices (Note 1)

<table>
<thead>
<tr>
<th>I/O Standard</th>
<th>Programmable Common Mode Voltage (V)</th>
<th>Coupling</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5 V PCML</td>
<td>0.82</td>
<td>AC, DC</td>
</tr>
<tr>
<td>2.5 V PCML</td>
<td>0.82</td>
<td>AC</td>
</tr>
<tr>
<td>LVPECL</td>
<td>0.82</td>
<td>AC</td>
</tr>
<tr>
<td>LVDS</td>
<td>1.1</td>
<td>AC, DC</td>
</tr>
</tbody>
</table>

Note to Table 1–6:
1. The differential OCT setting for Arria II GX and GZ transmitters and receivers is 85, 100, 120, and 150 Ω for Arria II GZ devices or 100 Ω for Arria II GX devices.

The following sections describe the features supported in the Arria II GX and GZ receiver input buffers.

Programmable Differential OCT

The Arria II GX transmitter buffer includes a differential OCT of 100 Ω. The Arria II GZ transmitter buffer includes a differential OCT of 85, 100, 120, and 150 Ω. The resistance is adjusted in the calibration block to compensate for temperature, voltage, and process changes (for more information, refer to “Calibration Block” on page 1–47). You can set this option in the Quartus II Assignment Editor by setting the assignment input termination to OCT 100 Ω for Arria II GX devices and OCT 85, 100, 120, and 150 Ω for Arria II GZ devices on the receiver input buffer.
Programmable Common Mode Voltage

The Arria II GX and GZ receivers have on-chip biasing circuitry to establish the required common mode voltage at the receiver input that supports two common mode voltage settings of 0.82 V and 1.1 V. You can select the voltage in the ALTGX MegaWizard Plug-In Manager. For the I/O standards supported by each common mode voltage setting, refer to Table 1–6.

This feature is effective only if you use programmable OCT for the receiver input buffers as well. If you use external termination, you must implement off-chip biasing circuitry to establish the common mode voltage at the receiver input buffer.

AC and DC Coupling

A high-speed serial link can either be AC-coupled or DC-coupled, depending on the serial protocol implementation. Most of the serial protocols require links to be AC-coupled, protocols similar to SONET optionally allow DC coupling.

In an AC-coupled link, the AC-coupling capacitor blocks the transmitter DC common mode voltage. The on-chip receiver termination and biasing circuitry automatically restores the selected common mode voltage. AC-coupled links are required in GbE, PCIe, SRIO, SDI, and XAUI protocols.

Figure 1–25 shows an AC-coupled link.

Figure 1–25. AC-Coupled Link

In a DC-coupled link, the transmitter DC common mode voltage is seen unblocked at the receiver input buffer. The link common mode voltage depends on the transmitter common mode voltage and the receiver common mode voltage. The on-chip or off-chip receiver termination and biasing circuitry must ensure compatibility between the transmitter and the receiver common mode voltage.
Figure 1–26 shows a DC-coupled link.

**Figure 1–26. DC-Coupled Link**

![DC-Coupled Link Diagram]

Figure 1–27 shows the DC-coupled link connection from an LVDS transmitter to an Arria II GX and GZ receiver.

**Figure 1–27. LVDS Transmitter to Arria II GX and GZ Receiver (PCML) DC-Coupled Link**

![LVDS Transmitter to Arria II GX and GZ Receiver Diagram]

Table 1–7 lists the settings for DC-coupled links between Altera devices. You must comply with the data rates supported by the Arria II GX and GZ receivers.

**Table 1–7. DC-Coupled Settings for Arria II Devices (Note 1) (Part 1 of 2)**

<table>
<thead>
<tr>
<th>Link</th>
<th>Transceiver Settings</th>
<th>Receiver Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arria II PCML transmitter to Arria II PCML receiver</td>
<td>TX VCM (V) 0.65</td>
<td>RX VCM (V) 0.82</td>
</tr>
<tr>
<td>Stratix II GX PCML transmitter to Arria II PCML receiver</td>
<td>TX VCM (V) 0.6, 0.7</td>
<td>RX VCM (V) 0.82</td>
</tr>
<tr>
<td>Arria II PCML transmitter to Stratix II GX PCML receiver</td>
<td>TX VCM (V) 0.65</td>
<td>RX VCM (V) 0.85</td>
</tr>
<tr>
<td>Arria II PCML transmitter to Stratix IV GX PCML receiver</td>
<td>TX VCM (V) 0.65</td>
<td>RX VCM (V) 0.82</td>
</tr>
<tr>
<td>Stratix IV GX PCML transmitter to Arria II PCML receiver</td>
<td>TX VCM (V) 0.65</td>
<td>RX VCM (V) 0.85</td>
</tr>
</tbody>
</table>
Programmable Equalization, DC Gain, and Offset Cancellation

Each Arria II GX and GZ receiver input buffer has independently programmable equalization circuitry that boosts the high-frequency gain of the incoming signal, thereby compensating for the low-pass filter effects of the physical medium. The amount of high-frequency gain required depends on the loss characteristics of the physical medium. Arria II GX and GZ equalization circuitry supports equalization settings that provide up to 7 dB (Arria II GX) and 16 dB (Arria II GZ) of high-frequency boost.

The Arria II GX and GZ receiver input buffer also supports programmable DC gain circuitry. Unlike equalization circuitry, DC gain circuitry provides equal boost to the incoming signal across the frequency spectrum.

You can select the proper equalization and DC gain settings in the ALTGX MegaWizard Plug-In Manager. The receiver buffer supports DC gain settings of 0 dB, 3 dB, and 6 dB for Arria II GX devices and up to 12 dB for Arria II GZ devices.

This offset cancellation block cancels offset voltages between the positive and negative differential signals within the equalizer stages in order to reduce the minimum $V_{ID}$ requirement. The receiver input buffer and receiver CDR require offset cancellation.

The offset cancellation for the receiver channels option is automatically enabled in both the ALTGX and ALTGX_RECONFIG MegaWizard Plug-In Managers for Receiver, Transmitter, and Receiver only configurations. When offset cancellation is automatically enabled, you must instantiate the dynamic reconfiguration controller to connect the reconfiguration ports created by the ALTGX MegaWizard Plug-In Manager.

For more information about offset cancellation, refer to AN 558: Implementing Dynamic Reconfiguration in Arria II Devices. For the transceiver reset sequence with the offset cancellation feature, refer to the Reset Control and Power Down in Arria II Devices chapter.

Signal Threshold Detection Circuitry

Signal threshold detection circuitry has a hysteresis response that filters out any high-frequency ringing caused by ISI effects or high-frequency losses in the transmission medium. If the signal threshold detection circuitry senses the signal level present at the receiver input buffer to be higher than the signal detect threshold, it asserts the $rx\_signal\_detect$ signal high. Otherwise, the $rx\_signal\_detect$ signal is held low.
In PCIe mode, you can enable the optional signal threshold detection circuitry by leaving the **Force signal detection** option unchecked in the ALTGX MegaWizard Plug-In Manager.

The appropriate signal detect threshold level that complies with the PCIe compliance parameter VRX-IDLE-DETDIFFp-p is pending characterization.

If you enable the **Force signal detection** option in the ALTGX MegaWizard Plug-In Manager, the `rx_signaldetect` signal is always asserted high, irrespective of the signal level on the receiver input buffer. When enabled, this option senses whether the signal level present at the receiver input buffer is above the signal detect threshold voltage that you specified in the **What is the signal detect and signal loss threshold?** option in the ALTGX MegaWizard Plug-In Manager.

The `rx_signaldetect` signal is also used by the LTR/LTD controller in the receiver CDR to switch between LTR and LTD lock modes. When the signal threshold detection circuitry de-asserts the `rx_signaldetect` signal, the LTR/LTD controller switches the receiver CDR from lock-to-data (LTD) to lock-to-reference (LTR) lock mode.

### CDR

Each Arria II GX and GZ receiver channel has an independent CDR unit to recover the clock from the incoming serial data stream. High-speed and low-speed recovered clocks are used to clock the receiver PMA and PCS blocks. **Figure 1–28** shows the CDR block.

**Figure 1–28. CDR Block**
The CDR operates in two modes:

- **LTR mode**—The PFD in the CDR tracks the receiver input reference clock (rx_cruclk) and controls the charge pump that tunes the VCO in the CDR. An active high rx_pll_locked status signal is asserted to indicate that the CDR has locked to phase and frequency of the receiver input reference clock. In this mode, the phase detector is inactive.

- **LTD mode**—The phase detector in the CDR tracks the incoming serial data at the receiver input buffer to keep the recovered clock phase-matched to the data. Depending on the phase difference between the incoming data and the CDR output clock, the phase detector controls the CDR charge pump that tunes the VCO.

In this mode, the PFD and the /M divider block are inactive. In addition, the rx_pll_locked signal toggles randomly and has no significance in LTD mode.

The CDR must be in LTD mode to recover the clock from the incoming serial data during normal operation. The actual LTD lock time depends on the transition density of the incoming data and the PPM difference between the receiver input reference clock and the upstream transmitter reference clock. The receiver PCS logic must be held in reset until the CDR asserts the rx_freqlocked signal and produces a stable recovered clock.

For more information about receiver reset recommendations, refer to the Reset Control and Power Down chapter.

The CDR must be kept in LTR mode until it locks to the input reference clock after the power-up and reset cycle. When locked to the input reference clock, the CDR output clock is trained to the configured data rate and can switch to LTD mode to recover the clock from the incoming data. You can use the optional input ports (rx_locktorefclk and rx_locktodata) to control the LTR or LTD mode manually or let the lock happen automatically.

Table 1–8 lists the relationship between the optional input ports and the LTR/LTD controller lock mode.

<table>
<thead>
<tr>
<th>rx_locktorefclk</th>
<th>rx_locktodata</th>
<th>LTR/LTD Controller Lock Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Manual – LTR Mode</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>Manual – LTD Mode</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Automatic Lock Mode</td>
</tr>
</tbody>
</table>

**Note to Table 1–8:**

(1) If you do not instantiate the optional rx_locktorefclk and rx_locktodata signals in the ALTGX megafunction, the Quartus II software automatically configures the LTR/LTD controller in automatic lock mode.
Automatic Lock Mode

In automatic lock mode, the LTR/LTD controller relies on the PPM detector and the phase relationship detector to set the CDR in LTR or LTD mode. Initially, the CDR is set to LTR mode. After the CDR locks to the input reference clock, the LTR/LTD controller automatically sets it to LTD mode and asserts the `rx_freqlocked` signal when the following three conditions are met:

- Signal threshold detection circuitry indicates the presence of valid signal levels at the receiver input buffer
- CDR output clock is within the configured PPM frequency threshold setting with respect to the input reference clock (frequency is locked)
- CDR output clock and input reference clock are phase matched within approximately 0.08 UI (phase is locked)

If the CDR does not stay locked-to-data due to frequency drift or severe amplitude attenuation, the LTR/LTD controller switches the CDR back to LTR mode to lock to the input reference clock. The LTR/LTD controller switches the CDR from LTD to LTR mode and de-asserts the `rx_freqlocked` signal when the following conditions are met:

- Signal threshold detection circuitry indicates the absence of valid signal levels at the receiver input buffer
- CDR output clock is not in the configured PPM frequency threshold setting with respect to the input reference clock

Manual Lock Mode

You may want to use manual lock mode if your application requires faster CDR lock time. In manual lock mode, the LTR/LTD controller sets the CDR in LTR or LTD mode, depending on the logic level on the `rx_locktorefclk` and `rx_locktodata` signals, as shown in Table 1–8 on page 1–27.

When the `rx_locktorefclk` signal is asserted high, the `rx_freqlocked` signal does not have significance and is always driven low, indicating that the CDR is in LTR mode. When the `rx_locktodata` signal is asserted high, the `rx_freqlocked` signal is always driven high, indicating that the CDR is in LTD mode. If both signals are de-asserted, the CDR is in automatic lock mode.

You must comply with the different transceiver reset sequences depending on the CDR lock mode.
**Deserializer**

The deserializer block latches the serial input data from the receiver input buffer with the high-speed serial recovery clock, deserializes it using the low-speed parallel recovery clock, and drives the deserialized data to the receiver PCS channel.

The deserializer supports 8-, 10-, 16-, and 20-bit deserialization factors. Figure 1–29 shows the deserializer operation with a 10-bit deserialization factor.

**Figure 1–29. 10-Bit Deserializer Operation**

---

Figure 1–30 shows the serial bit order of the deserializer block input and the parallel data output of the deserializer block with a 10-bit deserialization factor. The serial stream (10'b0101111100) is deserialized to a value 10'h17C. The serial data is assumed to have received the LSB first.

**Figure 1–30. 10-Bit Deserializer Bit Order**
Receiver PCS

This section describes the receiver PCS modules, which consist of the word aligner, deskew FIFO, rate-match FIFO, 8B/10B decoder, byte deserializer, byte ordering, and RX phase compensation FIFO.

The `rx_digitalreset` signal resets all modules in the receiver PCS block.

For more information about this signal, refer to the *Reset Control and Power Down in Arria II Devices* chapter.

Word Aligner

The word aligner receives parallel data from the deserializer and restores the word boundary based on a pre-defined alignment pattern that must be received during link synchronization.

Serial protocols such as GbE, PCIe, SRIO, SONET/SDH, and XAUI specify a standard word alignment pattern. The Arria II GX and GZ transceiver architecture allows you to select a custom word alignment pattern specific to your implementation if you use proprietary protocols.

Figure 1–31 shows the word aligner block diagram.

In addition to restoring word boundaries, the word aligner also implements the following features:

- Programmable run length violation detection—This feature is available in all functional modes. It detects consecutive 1s or 0s in the data stream. If a preset maximum number of consecutive 1s or 0s is detected, the run length violation status signal (`rx_rlv`) is asserted. This signal has lower latency when compared with the parallel data on the `rx_dataout` port.

The `rx_rlv` signal in each channel is clocked by its parallel recovered clock and is asserted for a minimum of two recovered clock cycles to ensure that the FPGA fabric clock can latch the `rx_rlv` signal reliably because the FPGA fabric clock might have phase differences, PPM differences (in asynchronous systems), or both, with the recovered clock.
Table 1–9 lists the detection capabilities of the run-length violation circuit.

Table 1–9. Detection Capabilities of the Run-Length Violation Circuit for Arria II Devices

<table>
<thead>
<tr>
<th>PMA-PCS Interface Width</th>
<th>Run Length Violation Detector Range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minimum</td>
</tr>
<tr>
<td>8 bit</td>
<td>4</td>
</tr>
<tr>
<td>10 bit</td>
<td>5</td>
</tr>
<tr>
<td>16 bit</td>
<td>8</td>
</tr>
<tr>
<td>20 bit</td>
<td>10</td>
</tr>
</tbody>
</table>

Receiver polarity inversion—This feature is available in all functional modes except PCIe. It offers an optional rx_invpolarity port to dynamically enable the receiver polarity inversion feature as a workaround to board re-spin or a major update to the FPGA fabric design when the positive and negative signals of a serial differential link are accidentally swapped during board layout.

A high value on the rx_invpolarity port inverts the polarity of every bit of the input data word to the word aligner in the receiver datapath. Because 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 signal is dynamic and might cause initial disparity errors in an 8B/10B encoded link. The downstream system must be able to tolerate these disparity errors.

Figure 1–32 shows an example result with the rx_invpolarity feature in a 10-bit wide datapath configuration.

Figure 1–32. 10-Bit Receiver Polarity Inversion

This generic receiver polarity inversion feature is different from the PCIe 8B/10B polarity inversion feature because it inverts the polarity of the data bits at the input of the word aligner, whereas the PCIe 8B/10B polarity inversion feature inverts the polarity of the data bits at the input of the 8B/10B decoder.
Receiver bit reversal—This feature is only available in Basic mode. By default, the Arria II GX and GZ receiver assumes LSB-to-MSB transmission. If the transmission order is MSB-to-LSB, the receiver forwards the bit-flipped version of the parallel data to the FPGA fabric on the rx_dataout port. The receiver bit reversal feature is available to correct this situation by flipping the parallel data so that the rx_dataout port contains the correct bit-ordered data.

This feature is available through the rx_revbitordwa port in Basic mode only with the word aligner configured in bit-slip mode. When you drive the rx_revbitordwa signal high in this configuration, the 8-bit or 10-bit data D[7:0] or D[9:0] at the output of the word aligner gets rewired to D[0:7] or D[0:9], respectively.

Figure 1–33 shows the receiver bit reversal feature in Basic mode with 10-bit wide datapath configurations.

Table 1–10 lists the three modes of the word aligner and their supported data width, functional mode, and allowed alignment pattern length for Arria II devices.
Manual Alignment Mode

This mode is automatically used in SONET/SDH functional mode. In Basic mode, you can configure the word aligner in manual alignment mode by selecting the **Use manual word alignment mode** option in the word aligner tab of the ALTGX MegaWizard Plug-In Manager.

In manual alignment mode, the input signal (rx_enapatternalign) controls the word aligner. The 8-bit word aligner is edge-sensitive to the rx_enapatternalign signal; the 10-bit word aligner is level-sensitive to this signal.

If the word alignment pattern is unique and does not appear between word boundaries, you can constantly hold the rx_enapatternalign signal high because there is no possibility of false word alignment. If there is a possibility of the word alignment pattern occurring across word boundaries, you must control the rx_enapatternalign signal to lock the word boundary after the desired word alignment is achieved to avoid re-alignment to an incorrect word boundary.

With 8-bit width data, a rising edge on the rx_enapatternalign signal after de-assertion of the rx_digitalreset signal triggers the word aligner to look for the word alignment pattern in the received data stream.

In SONET/SDH OC-12 and OC-48 modes, the word aligner looks for 16'hF628 (A1A2) or 32'hF6F62828 (A1A1A2A2), depending on whether the input signal (rx_a1a2size) is driven low or high, respectively. In Basic mode, the word aligner looks for the 16-bit word alignment pattern programmed in the ALTGX MegaWizard Plug-In Manager.

With 10-bit width data, the word aligner looks for the programmed 7-bit or 10-bit word alignment pattern in the received data stream, if the rx_enapatternalign signal is held high. It updates the word boundary if it finds the word alignment pattern in a new word boundary. If the rx_enapatternalign signal is de-asserted low, the word aligner maintains the current word boundary even when it sees the word alignment pattern in a new word boundary.

The rx_syncstatus and rx_patterndetect status signals have the same latency as the datapath and are forwarded to the FPGA fabric to indicate word aligner status. On receiving the first word alignment pattern after the assertion of the rx_enapatternalign signal, both the rx_syncstatus and rx_patterndetect signals are driven high for one parallel clock cycle synchronous to the most significant byte (MSByte) of the word alignment pattern. Any word alignment pattern received thereafter in the same word boundary causes only the rx_patterndetect signal to go high for one clock cycle.

Any word alignment pattern received thereafter in a different word boundary causes the word aligner to re-align to the new word boundary only if there is a rising edge in the rx_enapatternalign signal (in the 8-bit word aligner) or if the rx_enapatternalign signal is held high (in 10-bit word aligner). The word aligner asserts the rx_syncstatus and rx_patterndetect signals for one parallel clock cycle whenever it re-aligns to the new word boundary.
Figure 1–34 shows word aligner behavior in SONET/SDH OC-12 functional mode. The least significant byte (LSByte) (8’hF6) and the MSByte (8’h28) of the 16-bit word alignment pattern are received in parallel clock cycles n and n + 1, respectively. The \texttt{rx\_syncstatus} and \texttt{rx\_patterndetect} signals are both driven high for one parallel clock cycle synchronous to the MSByte (8’h28) of the word alignment pattern. After the initial word alignment, the 16-bit word alignment pattern (16’h28F6) is again received across the word boundary in clock cycles m, m + 1, and m + 2. The word aligner does not re-align to the new word boundary for lack of a preceding rising edge on the \texttt{rx\_enapatternalign} signal. If there is a rising edge on the \texttt{rx\_enapatternalign} signal before the word alignment pattern occurs across these clock cycles, the word aligner re-aligns to the new word boundary, causing both the \texttt{rx\_syncstatus} and \texttt{rx\_patterndetect} signals to go high for one parallel clock cycle.

Figure 1–35 shows the manual alignment mode word aligner operation in 10-bit PMA-PCS interface mode. In this example, a /K28.5/ (10'b0101111100) is specified as the word alignment pattern. The word aligner aligns to the /K28.5/ alignment pattern in cycle n because the \texttt{rx\_enapatternalign} signal is asserted high. The \texttt{rx\_syncstatus} signal goes high for one clock cycle indicating alignment to a new word boundary. The \texttt{rx\_patterndetect} signal also goes high for one clock cycle to indicate initial word alignment. At time n + 1, the \texttt{rx\_enapatternalign} signal is de-asserted to instruct the word aligner to lock the current word boundary. The alignment pattern is detected again in a new word boundary across cycles n + 2 and n + 3. The word aligner does not align to this new word boundary because the \texttt{rx\_enapatternalign} signal is held low. The /K28.5/ word alignment pattern is detected again in the current word boundary during cycle n + 5, causing the \texttt{rx\_patterndetect} signal to go high for one parallel clock cycle.
With a 16- or 20-bit width data, the word aligner starts looking for the programmed 8-bit, 16-bit, or 32-bit word alignment pattern in the received data stream as soon as \texttt{rx\_digitalreset} is de-asserted low. It aligns to the first word alignment pattern received regardless of the logic level driven on the \texttt{rx\_enapatternalign} signal. Any word alignment pattern received thereafter in a different word boundary does not cause the word aligner to re-align to this new word boundary. After the initial word alignment, following de-assertion of the \texttt{rx\_digitalreset} signal, if a word re-alignment is required, you must use the \texttt{rx\_enapatternalign} signal.

Figure 1–36 shows the manual alignment mode word aligner operation in 16-bit PMA-PCS interface mode.

**Figure 1–36. Manual Alignment Mode Word Aligner in 16-Bit PMA-PCS Interface Modes**

Note to Figure 1–36:

(1) The \texttt{rx\_syncstatus} de-assertion latency after the assertion of \texttt{rx\_enapatternalign} depends on your RX PCS implementation.
**Bit-Slip Mode**

In Basic, deterministic latency, and SDI functional modes, you can configure the word aligner in bit-slip mode by selecting the **Use manual bit slipping mode** option in the ALTGX MegaWizard Plug-In Manager.

Bit slip in the 10-bit wide word aligner allows 7-bit and 10-bit word alignment patterns, whereas bit-slip in the 8-bit wide word aligner allows only a 16-bit word alignment pattern. Other than this, the bit-slip operation is the same between the 8-bit and 10-bit word aligner.

The `rx_bitslip` signal controls the word aligner operation in bit-slip mode. At every rising edge of the `rx_bitslip` signal, the bit-slip circuitry slips one bit into the received data stream, effectively shifting the word boundary by one bit. The `rx_patterndetect` signal is driven high for one parallel clock cycle when the received data after bit-slipping matches the 16-bit word alignment pattern programmed in the ALTGX MegaWizard Plug-In Manager.

You can implement a bit-slip controller in the FPGA fabric that monitors either the `rx_dataout` signal and/or the `rx_patterndetect` signal and controls the `rx_bitslip` signal to achieve word alignment.

**Figure 1–37** shows an example of the word aligner configured in bit-slip mode, which has the following events:

- 8'b11110000 is received back-to-back
- 16'b0000111100011110 is specified as the word alignment pattern
- A rising edge on the `rx_bitslip` signal at time n + 1 slips a single bit 0 at the MSB position, forcing the `rx_dataout` to 8'b01111000
- Another rising edge on the `rx_bitslip` signal at time n + 5 forces `rx_dataout` to 8'b00111100
- Another rising edge on the `rx_bitslip` signal at time n + 9 forces `rx_dataout` to 8'b00011110
- Another rising edge on the `rx_bitslip` signal at time n + 13 forces `rx_dataout` to 8'b00001111. At this instance, `rx_dataout` in cycles n + 12 and n + 13 are 8'b00011110 and 8'b00001111, respectively, which matches the specified 16-bit alignment pattern 16'b0000111100011110. This results in the assertion of the `rx_patterndetect` signal.

**Figure 1–37. Example of Word Aligner Configured in Bit-Slip Mode**
Bit-Slip Mode Word Aligner with 16-Bit PMA-PCS Interface Modes

In some Basic double-width configurations with 16-bit PMA-PCS interface, you can configure the word aligner in bit-slip mode by selecting the **Use manual bit slipping mode** option in the ALTGX MegaWizard Plug-In Manager.

The word aligner operation for Basic double-width with 16-bit PMA-PCS interface is similar to the word aligner operation in Basic single-width mode with 8-bit PMA-PCS interface. The only difference is that the bit-slip word aligner in 16-bit PMA-PCS interface modes allows 8-bit and 16-bit word alignment patterns, whereas the bit-slip word aligner in 8-bit PMA-PCS interface modes allows only a 16-bit word alignment pattern.

Word Aligner in Double-Width Mode with 20-Bit PMA-PCS Interface Modes

A 20-bit PMA-PCS interface is supported only in Basic double-width mode.

Table 1–11 shows the word aligner configurations allowed in functional modes with a 20-bit PMA-PCS interface.

<table>
<thead>
<tr>
<th>Functional Mode</th>
<th>Allowed Word Aligner Configurations</th>
<th>Allowed Word Alignment Pattern Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic double-width</td>
<td>Manual alignment, Bit-slip</td>
<td>7 bits, 10 bits, 20 bits</td>
</tr>
</tbody>
</table>

Manual Alignment Mode Word Aligner with 20-Bit PMA-PCS Interface Modes

The word aligner operation in Basic double-width mode with 20-bit PMA-PCS interface is similar to the word aligner operation in Basic double-width mode with a 16-bit PMA-PCS interface. The only difference is that the manual alignment mode word aligner in 20-bit PMA-PCS interface modes allows 7-, 10-, and 20-bit word alignment patterns, whereas the manual alignment mode word aligner in 16-bit PMA-PCS interface modes allows only 8-, 16-, and 32-bit word alignment patterns.

Bit-Slip Mode Word Aligner with 20-Bit PMA-PCS Interface Modes

In some Basic single-width configurations with a 20-bit PMA-PCS interface, you can configure the word aligner in bit-slip mode by selecting the **Use manual bit slipping mode** option in the ALTGX MegaWizard Plug-In Manager.

The word aligner operation for Basic double-width with 20-bit PMA-PCS interface is similar to the word aligner operation in Basic single-width mode with an 8-bit PMA-PCS interface. The difference is that the bit-slip word aligner in 20-bit PMA-PCS interface modes allows only 7-, 10-, and 20-bit word alignment patterns, whereas the bit-slip word aligner in 8-bit PMA-PCS interface modes allows only a 16-bit word alignment pattern.
Table 1–12 summarizes the word aligner options available in Basic single-width and double-width modes.

**Table 1–12. Word Aligner Options Available in Basic Single-Width and Double-Width Modes for Arria II Devices (Note 1) (Part 1 of 2)**

<table>
<thead>
<tr>
<th>Functional Mode</th>
<th>PMA-PCS Interface Width</th>
<th>Word Alignment Mode</th>
<th>Word Alignment Pattern Length</th>
<th>rx_enapatternalign Sensitivity</th>
<th>rx_synestatus Behavior</th>
<th>rx_patterndetect Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Single-Width</td>
<td>8-bit</td>
<td>Manual Alignment</td>
<td>16-bit</td>
<td>Rising Edge Sensitive</td>
<td>Asserted high for one parallel clock cycle when the word aligner aligns to a new word boundary.</td>
<td>Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.</td>
</tr>
<tr>
<td></td>
<td>Bit-Slip</td>
<td>16-bit</td>
<td>—</td>
<td>—</td>
<td>Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.</td>
<td></td>
</tr>
<tr>
<td>Basic Single-Width</td>
<td>10-bit</td>
<td>Manual Alignment</td>
<td>7- and 10-bit</td>
<td>Level Sensitive</td>
<td>Asserted high for one parallel clock cycle when the word aligner aligns to a new word boundary.</td>
<td>Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.</td>
</tr>
<tr>
<td></td>
<td>Bit-Slip</td>
<td>7- and 10-bit</td>
<td>—</td>
<td>—</td>
<td>Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Automatic Synchronization State Machine</td>
<td>7- and 10-bit</td>
<td>—</td>
<td>—</td>
<td>Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.</td>
<td>Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.</td>
</tr>
</tbody>
</table>
Table 1–12. Word Aligner Options Available in Basic Single-Width and Double-Width Modes for Arria II Devices *(Note 1)* (Part 2 of 2)

<table>
<thead>
<tr>
<th>Functional Mode</th>
<th>PMA-PCS Interface Width</th>
<th>Word Alignment Mode</th>
<th>Word Alignment Pattern Length</th>
<th>rx_enapatternalign Sensitivity</th>
<th>rx_syncstatus Behavior</th>
<th>rx_patterndetect Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Double-Width</td>
<td>16-bit</td>
<td>Manual Alignment</td>
<td>8-, 16-, and 32-bit</td>
<td>Rising Edge Sensitive</td>
<td>Stays high after the word aligner aligns to the word alignment pattern. Goes low on receiving a rising edge on rx_enapatternalign until a new word alignment pattern is received.</td>
<td>Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit-Slip</td>
<td>8-, 16-, and 32-bit</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>20-bit</td>
<td>Manual Alignment</td>
<td>7-, 10-, and 20-bit</td>
<td>Rising Edge Sensitive</td>
<td>Stays high after the word aligner aligns to the word alignment pattern. Goes low on receiving a rising edge on rx_enapatternalign until a new word alignment pattern is received.</td>
<td>Asserted high for one parallel clock cycle when the word alignment pattern appears in the current word boundary.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bit-Slip</td>
<td>7-, 10-, and 20-bit</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>

*Note to Table 1–12:*

(1) For more information about word aligner operation, refer to “Word Aligner” on page 1–30.
Automatic Synchronization State Machine Mode
You must use this mode with 8B/10B encoded data if the input data to the word aligner is 10 bits.

Protocols such as PCIe, XAUI, Gigabit Ethernet, and SRIO require the receiver PCS logic to implement a synchronization state machine to provide hysteresis during link synchronization. Each of these protocols defines a specific number of synchronization code groups that the link must receive to acquire synchronization and a specific number of erroneous code groups that it must receive to fall out of synchronization.

The Quartus II software configures the word aligner in automatic synchronization state machine mode for PCIe, XAUI, Gigabit Ethernet, and SRIO functional modes. It automatically selects the word alignment pattern length and the word alignment pattern as specified by each protocol. In each of these functional modes, the protocol-compliant synchronization state machine is implemented in the word aligner.

By using Basic functional mode with the 10-bit PMA-PCS interface, you can configure the word aligner in automatic synchronization state machine mode by selecting the Use the automatic synchronization state machine option in the ALTGX MegaWizard Plug-In Manager. Basic mode also allows you to program a custom 7-bit or 10-bit word alignment pattern that the word aligner uses for synchronization.

Table 1–13 lists the synchronization state machine parameters that the Quartus II software allows in supported functional modes. The synchronization state machine parameters are fixed for PCIe, XAUI, GbE, and SRIO modes as specified by the respective protocol. You can program these parameters as suited to your proprietary protocol implementation for Basic mode.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>PCIe</th>
<th>XAUI</th>
<th>GbE</th>
<th>SRIO</th>
<th>Basic Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of valid synchronization code groups or ordered sets received to achieve synchronization</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>127</td>
<td>1–256</td>
</tr>
<tr>
<td>Number of erroneous code groups received to lose synchronization</td>
<td>17</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1–64</td>
</tr>
<tr>
<td>Number of continuous good code groups received to reduce the error count by one</td>
<td>16</td>
<td>4</td>
<td>4</td>
<td>255</td>
<td>1–256</td>
</tr>
</tbody>
</table>

After de-assertion of the rx_digitalreset signal in automatic synchronization state machine mode, the word aligner starts looking for the word alignment pattern or synchronization code groups in the received data stream. When the programmed number of valid synchronization code groups or ordered sets is received, the rx_syncstatus signal is driven high to indicate that synchronization is acquired. The rx_syncstatus signal is constantly driven high until the programmed number of erroneous code groups is received without receiving intermediate good groups; after which the rx_syncstatus signal is driven low. The word aligner indicates loss of synchronization (rx_syncstatus signal remains low) until the programmed number of valid synchronization code groups are received again.
**Deskew FIFO**

Use this module, only available in XAUI mode, to align all four channels to meet the XAUI maximum skew of 40 UI (12.8 ns) as seen at the receiver of the four lanes. The deskew operation in XAUI functional mode is compliant to the PCS deskew state machine diagram specified in 8 of the IEEE P802.3ae.

The deskew circuitry consists of a 16-word deep deskew FIFO in each of the four channels and control logic in the CMU0 channel of the transceiver block that controls the deskew FIFO write and read operations in each channel. Figure 1–38 shows the deskew FIFO block diagram.

![Deskew FIFO Block Diagram](image)

**Figure 1–38. Deskew FIFO (Note 1)**

Data from the Word Aligner → Deskew FIFO → Data to the Rate-Match FIFO

rx_syncstatus → Deskew FIFO → rx_channelaligned

Note to Figure 1–38:
1. For more information about the deskew FIFO operation, refer to “XAUI” on page 1–79.

**Rate-Match FIFO**

In asynchronous systems, you can clock the upstream transmitter and local receiver with independent reference clocks. Frequency differences in the order of a few hundred PPM can corrupt the data when latching from the recovered clock domain (the same clock domain as the upstream transmitter reference clock) to the local receiver reference clock domain.

The rate match FIFO compensates for small clock frequency differences between the upstream transmitter and the local receiver clocks by inserting or removing SKP symbols or ordered-sets from the IPG or idle streams. It deletes SKP symbols or ordered-sets when the upstream transmitter reference clock frequency is higher than the local receiver reference clock frequency. It inserts SKP symbols or ordered-sets when the local receiver reference clock frequency is higher than the upstream transmitter reference clock frequency.

The rate match FIFO consists of a 20-word deep FIFO and necessary logic that controls insertion and deletion of a SKP character or ordered-set, depending on the PPM difference. This module is optional in Basic functional mode, but is mandatory and cannot be bypassed in GbE, PCIe, and XAUI functional modes.

For the Gigabit Ethernet protocol, if you enabled rate match FIFO in the auto-negotiation state machine in an FPGA core, refer to “Rate Match FIFO in GbE Mode” on page 1–61.
Figure 1–39 shows the rate-match FIFO block diagram.

**Figure 1–39. Rate Match FIFO**

![Diagram of Rate Match FIFO]

Data from either the Word Aligner, Basic, GIGE, and (PCIe, Functional Mode) or the Deskew FIFO (XAUI functional Mode)

- rx_rmfifodatainserted (1)
- rx_rmfifodatadeleted (1)
- rx_rmfifofull (1)
- rxfifoempty (1)

**Note to Figure 1–39:**
(1) These signals are not available in PCIe functional mode because the rate match FIFO status is encoded in the pipestatus[2:0] signal.

---

**8B/10B Decoder**

Figure 1–40 shows the block diagram of the 8B/10B decoder.

**Figure 1–40. 8B/10B Decoder**

![Diagram of 8B/10B Decoder]

The 8B/10B decoder is compliant to Clause 36 in the IEEE802.3 specification, decoding the 10-bit data input into an 8-bit data and a 1-bit control identifier. This module is required in GbE, PCIe, SRIO, and XAUI functional modes, but is optional in Basic functional mode.

Figure 1–41 shows a 10-bit code group decoded into an 8-bit data and a 1-bit control identifier by the 8B/10B decoder.

**Figure 1–41. 8B/10B Decoder Operation**

![Diagram of 8B/10B Decoder Operation]
The 8B/10B decoder indicates whether the decoded 8-bit code group is a data or control code group on the \texttt{rx_ctrldetect} port. If the received 10-bit code group is one of the 12 control code groups (\texttt{/Kx.y/}) specified in the IEEE802.3 specification, the \texttt{rx_ctrldetect} signal is driven high. If the received 10-bit code group is a data code group (\texttt{/Dx.y/}), the \texttt{rx_ctrldetect} signal is driven low.

Figure 1–42 shows the 8B/10B decoder decoding the received 10-bit \texttt{/K28.5/} control code group into an 8-bit data code group (8’hBC) driven on the \texttt{rx_dataout} port. The \texttt{rx_ctrldetect} signal is asserted high synchronous with 8’hBC on the \texttt{rx_dataout} port, indicating that it is a control code group. The rest of the codes received are data code groups \texttt{/Dx.y/}.

**Byte Deserializer**

Some serial data rates violate the maximum frequency for the FPGA fabric-to-transceiver interface. In such configurations, the byte deserializer is required to reduce the FPGA fabric-to-transceiver interface frequency to half while doubling the parallel data width. This module is optional in configurations that do not exceed the FPGA fabric-to-transceiver interface clock upper frequency limit.

For example, at a 3.2 Gbps data rate with a deserialization factor of 10, the receiver PCS datapath runs at 320 MHz. The byte serializer converts the 10-bit parallel received data at 320 MHz into 20-bit parallel data at 160 MHz before forwarding it to the FPGA fabric.

In Arria II GX devices, you cannot enable the byte deserializer in double-width mode. However, in Arria II GZ devices, you can enable both double-width mode and the byte deserializer to achieve a 32- or 40-bit PCS-FPGA interface.

Figure 1–43 shows the block diagram of the byte deserializer with 8-bit or 10-bit PMA-to-PCS interface.
Byte Ordering

Depending on when the receiver PCS logic comes out of reset, byte ordering at the output of the byte deserializer may not match the original byte ordering of the transmitted data. The byte misalignment resulting from byte deserialization is unpredictable because it depends on which byte is being received by the byte deserializer when it comes out of reset.

Figure 1–44 shows a scenario in which the MSByte and LSByte of the two-byte transmitter data appears straddled across two word boundaries after getting byte deserialized at the receiver.

![Diagram](image_url)

Figure 1–44. MSByte and LSByte of the Two-Byte Transmitter Data Straddled Across Two Word Boundaries

The byte ordering block looks for the user-programmed byte ordering pattern in the byte-deserialized data. You must select a byte ordering pattern that you know appears at the LSByte(s) position of the parallel transmitter data. If the byte ordering block finds the programmed byte ordering pattern in the MSbyte(s) position of the byte-deserialized data, it inserts the appropriate number of user-programmed PAD bytes to push the byte ordering pattern to the LSByte(s) position, thereby restoring proper byte ordering.

The byte ordering block is available in the following functional modes:

- **SONET/SDH OC-48**—The Quartus II software automatically configures the byte ordering pattern and byte ordering PAD pattern in this mode.

- **SONET/SDH OC-96**—For Arria II GZ devices only.

- **Basic mode with 16-bit FPGA fabric-to-transceiver interface, no 8B/10B decoder (8-bit PMA-PCS interface) and word aligner in manual alignment mode**—You can program a custom 8-bit byte ordering pattern and 8-bit byte ordering PAD pattern in the ALTGX MegaWizard Plug-In Manager.

- **Basic mode with 16-bit FPGA fabric-to-transceiver interface, 8B/10B decoder, and word aligner in automatic synchronization state machine mode**—You can program a custom 9-bit byte ordering pattern and 9-bit byte ordering PAD pattern in the ALTGX MegaWizard Plug-In Manager. If a /Kx.y/ control code group is selected as the byte ordering pattern, the MSB of the 9-bit byte ordering pattern must be 1'b1. If a /Dx.y/ data code group is selected as the byte ordering pattern, the MSB of the 9-bit byte ordering pattern must be 1'b0. The least significant 8 bits must be the 8B/10B decoded version of the code group used for byte ordering.
The byte ordering modules have two different modes of operation when you select the `rx_syncstatus` signal from the word aligner option in the **What do you want the byte ordering to be based on?** field in the ALTGX MegaWizard Plug-In Manager:

- **Word-Alignment-Based Byte Ordering**—In this mode, the byte ordering block starts looking for the byte ordering pattern in the byte-deserialized data every time it sees a rising edge on the `rx_syncstatus` signal.

  If the byte ordering block finds the first data byte that matches the programmed byte ordering pattern in the MSByte position of the byte-deserialized data, it inserts one programmed PAD pattern to push the byte ordering pattern into the LSByte position after a rising edge on the `rx_syncstatus` signal.

  If the byte ordering block finds the first data byte that matches the programmed byte ordering pattern in the LSByte position of the byte-deserialized data, it considers the data to be ordered and does not insert any PAD pattern. In either case, the byte ordering block asserts the `rx_byteorderalignstatus` signal.

  **Figure 1–45** shows an example of the byte ordering operation where A is the programmed byte ordering pattern and PAD is the programmed PAD pattern. The byte deserialized data places the byte ordering pattern A in the MSByte position, resulting in incorrect byte ordering. Assuming that a rising edge on the `rx_syncstatus` signal had occurred before the byte ordering block sees the byte ordering pattern A in the MSByte position, the byte ordering block inserts a PAD byte and pushes the byte ordering pattern A into the LSByte position. The data at the output of the byte ordering block has correct byte ordering as reflected on the `rx_byteorderalignstatus` signal.

- **User-Controlled Byte Ordering**—Unlike word-alignment-based byte ordering, user-controlled byte ordering provides control to the user logic to restore correct byte ordering at the receiver.

  When enabled, an `rx_enabyteord` port is available as a trigger to the byte ordering operation. A rising edge on the `rx_enabyteord` port triggers the byte ordering block. If the byte ordering block finds the first data byte that matches the programmed byte ordering pattern in the MSByte position of the byte-deserialized data, it inserts one programmed PAD pattern to push the byte ordering pattern
into the LSByte position after a rising edge on the rx_enabyteord signal. If the byte ordering blocks find the first data byte that matches the programmed byte ordering pattern in the LSByte position of the byte-deserialized data, it considers the data to be byte ordered and does not insert any PAD byte. In either case, the byte ordering block asserts the rx_byteorderalignstatus signal.

RX Phase Compensation FIFO

The RX phase compensation FIFO in each channel ensures reliable transfer of data and status signals between the receiver channel and the FPGA fabric. The RX phase compensation FIFO compensates for the phase difference between the parallel receiver PCS clock (FIFO write clock) and the FPGA fabric clock (FIFO read clock).

Table 1–14 lists the RX phase compensation FIFO modes. The Quartus II software automatically sets the mode that applies to a particular functional mode.

<table>
<thead>
<tr>
<th>Mode</th>
<th>FIFO Depth</th>
<th>Latency Through FIFO</th>
<th>Applicable Functional Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Latency</td>
<td>4-words deep</td>
<td>2-to-3 parallel clock cycles</td>
<td>All functional modes except PCIe and Deterministic Latency</td>
</tr>
<tr>
<td>High-Latency</td>
<td>8-words deep</td>
<td>4-to-5 parallel clock cycles</td>
<td>PCIe</td>
</tr>
<tr>
<td>Register</td>
<td>—</td>
<td>1</td>
<td>Deterministic Latency</td>
</tr>
</tbody>
</table>

Table 1–14. RX Phase Compensation FIFO Modes for Arria II Devices

Figure 1–46 shows the RX phase compensation FIFO block diagram. The write clock and read clock to the FIFO are at half-rate if the byte serializer is used in the configuration.

Figure 1–46. RX Phase Compensation FIFO Block Diagram

Notes to Figure 1–46:
(1) The tx clkout clock is from the transmitter channel clock divider and is used in non-bonded configurations with the rate match FIFO.
(2) The rx_clkout clock is from the receiver channel CDR and is used in non-bonded configurations that does not use the rate match FIFO.
(3) The coreclkout clock is from the CMU0 block of the associated transceiver block or the master transceiver block for ×4 bonded or ×8 bonded channel configurations, respectively.
(4) Use this clock for the FIFO read clock if you instantiate the rx_coreclk port in the ALTGX MegaWizard Plug-In Manager, regardless of the channel configurations. Otherwise, use the same clock used for the write clock for the read clock.
(5) The rx_phase_comp fifo_error signal is optional and available in all functional modes. This signal is asserted high to indicate an overflow or underflow condition.
Calibration Block

This block calibrates the OCT resistors and the analog portions of the transceiver blocks to ensure that the functionality is independent of process, voltage, or temperature (PVT) variations. Figure 1–47 shows the location of the calibration block and how it is connected to the transceiver blocks in Arria II GX devices.

Figure 1–47. Arria II GX Transceiver Calibration Blocks Location and Connection (Note 1), (2), (3)

Notes to Figure 1–47:

1. You must connect a separate 2 kΩ (tolerance max ± 1%) external resistor on each RREF pin in the Arria II device to GND. To ensure proper operation of the calibration block, the RREF resistor connection in the board must be free from external noise.

2. The Quartus II software automatically selects the appropriate calibration block based on the pin assignments of the transceiver tx_dataout and rx_datain pins.

3. For RREF pin information, refer to the Device Pin Connection Guidelines.
Figure 1–48 shows the location of the calibration block and how it is connected to the transceiver blocks in Arria II GZ devices.

**Notes to Figure 1–48:**

1. You must connect a separate 2 kΩ (tolerance max ± 1%) external resistor on each RREF pin in the Arria II device to GND. To ensure proper operation of the calibration block, the RREF resistor connection in the board must be free from external noise.

2. The Quartus II software automatically selects the appropriate calibration block based on the pin assignments of the transceiver tx_dataout and rx_datain pins.

3. For RREF pin information, refer to the Device Pin Connection Guidelines.

The calibration block internally generates a constant internal reference voltage, independent of PVT variations and uses this voltage and the external reference resistor on the RREF pin to generate constant reference currents. These reference currents are used by the analog block calibration circuit to calibrate the transceiver blocks.

The OCT calibration circuit calibrates the OCT resistors present in the transceiver channels. You can enable the OCT resistors in the transceiver channels through the ALTGX MegaWizard Plug-In Manager.
Figure 1–49 shows the calibration block diagram. You can instantiate the `cal_blk_clk` and `cal_blk_powerdown` ports in the ALTGX MegaWizard Plug-In Manager to control the calibration block. Drive the `cal_blk_clk` port of all ALTGX instances that are associated with the same calibration block from the same input pin or logic.

**Figure 1–49. Input Signals to the Calibration Blocks**

Notes to Figure 1–49:

1. You must connect a separate 2 kΩ (tolerance max ± 1%) external resistor on each `RREF` pin in the Arria II GX and GZ device to GND. To ensure proper operation of the calibration block, the `RREF` resistor connection in the board must be free from external noise.

2. This is the input clock to the calibration block. You can use dedicated clock routes such as the global or regional clock; however, if you do not have a suitable input reference clock or dedicated clock routing resources available, use divide-down logic from the FPGA fabric to generate a slow clock, local clocking routing, or both.

3. Drive this signal of all ALTGX instances that are associated with the same calibration block from the same input or logic. During calibration, the link may experience higher bit error rates due to changes in the signaling condition. The transceiver on-chip termination calibration process takes approximately 33,000 `cal_blk_clk` cycles from the de-assertion of the `cal_blk_powerdown` signal.
PCIe Hard IP Block

Figure 1–50 shows the block diagram of the PCIe hard IP block used to implement the PHY MAC, Data Link Layer, and Transaction Layer for PCIe interfaces. When you use this block, the PIPE interface is used as the interface between the FPGA fabric and the transceiver.

**Figure 1–50. PCIe Hard IP High-Level Block Diagram**

![PCIe Hard IP Block Diagram]

This block is enabled only when you use the PCIe MegaCore function. For more information about using this block, refer to the *PCI Express Compiler User Guide*.

You can configure the root port or end point in x1, x4, and x8 modes. You can also include instances of both the soft and hard IP PCIe MegaCore function in a single device.

The PCIe hard IP block ignores autonomous and speed change bits set in TS1 and TS2 ordered sets. Altera is fully compatible to the PCIe Base Specification v2.0.

**Functional Modes**

You can configure Arria II GX and GZ transceivers in one of the following functional modes with the ALTGX MegaWizard Plug-In Manager:

- Basic at 600 Mbps to 6.375 Gbps
- Deterministic latency, used for CPRI and OBSAI protocols
- GbE (1.25 Gbps)
- PCIe (Gen1 at 2.5 Gbps) (Gen2 at 5.0 Gbps for Arria II GZ devices only)
- SDI (HD at 1.485 and 1.4835 Gbps, 3G at 2.97 and 2.967 Gbps)
- SRIO (1.25 Gbps, 2.5 Gbps, and 3.125 Gbps)
- SONET/SDH (OC-12 and OC-48) (OC-96 for Arria II GZ devices only)
The following sections describe the functional modes available in the ALTGX MegaWizard Plug-In Manager that you can set through the Which protocol will you be using? option.

Table 1–15 lists the PCS latency for the different functional modes of Arria II devices.

### Table 1–15. Functional Modes PCS Latency for Arria II Devices (Note 1)

<table>
<thead>
<tr>
<th>Functional Mode</th>
<th>PCS Latency (FPGA Fabric-Transceiver Interface Clock Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TX PCS</td>
</tr>
<tr>
<td>Basic</td>
<td>(2)</td>
</tr>
<tr>
<td>Deterministic latency</td>
<td>4</td>
</tr>
<tr>
<td>GbE</td>
<td>5–6</td>
</tr>
<tr>
<td>PCIe</td>
<td>4–5.5 (byte SERDES enabled)</td>
</tr>
<tr>
<td>SDI</td>
<td>4–5.5 (byte SERDES enabled)</td>
</tr>
<tr>
<td>SRI0</td>
<td>4–5</td>
</tr>
<tr>
<td>SONET/SDH</td>
<td>5–6 (OC-12)</td>
</tr>
<tr>
<td>XAUI</td>
<td>4.5–6</td>
</tr>
</tbody>
</table>

Notes to Table 1–15:
1. Not all modes are supported by both Arria II GX and Arria II GZ devices. Refer to the respective functional mode sections for the supported modes in each device family.
2. For basic mode latency values, refer to Figure 1–51, Figure 1–52, Figure 1–53, and Figure 1–54.

### Basic Mode

The Arria II GX and GZ transceiver datapath is highly flexible in Basic functional mode. It allows 8-bit and 10-bit PMA-to-PCS interface, which is determined by whether you bypass or use the 8B/10B encoder/decoder.

Depending on the targeted data rate, you can optionally bypass the byte serializer and deserializer blocks in Basic mode but the transmitter and RX phase compensation FIFOs are always enabled. The word aligner is always enabled in regular Basic mode, but bypassed in low latency PCS mode, which can be enabled through the Enable low latency PCS mode option in the ALTGX MegaWizard Plug-In Manager.

The low latency PCS mode creates a Basic functional mode configuration that bypasses the following transmitter and receiver channel PCS blocks to form a low latency PCS datapath:
- 8B/10B encoder and decoder
- Word aligner
- Deskew FIFO
- Rate match (clock rate compensation) FIFO
- Byte ordering
Figure 1–51 shows the Arria II GX and GZ transceiver configurations allowed in Basic functional mode with an 8-bit wide PMA-PCS interface.

**Figure 1–51. Transceiver Configurations in Basic Mode with an 8-Bit Wide PMA-to-PCS Interface**

(Note 1)

<table>
<thead>
<tr>
<th>Functional Mode</th>
<th>Basic 8-Bit PMA-PCS Interface Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Rate (2)</td>
<td>0.6 - 3.0 Gbps</td>
</tr>
<tr>
<td>Number of Channels</td>
<td>x1, x4, x8</td>
</tr>
<tr>
<td>Low-Latency PCS</td>
<td>8-Bit</td>
</tr>
<tr>
<td>Word Aligner (Pattern Length)</td>
<td>Manual Alignment (16-Bit)</td>
</tr>
<tr>
<td>8B/10B Encoder/Decoder</td>
<td>Disabled</td>
</tr>
<tr>
<td>Rate Match FIFO</td>
<td>Disabled</td>
</tr>
<tr>
<td>Byte SERDES (3)</td>
<td>Disabled</td>
</tr>
<tr>
<td>Byte Ordering</td>
<td>Disabled</td>
</tr>
<tr>
<td>FPGA Fabric-to-Transceiver Interface Width</td>
<td>8-Bit, 16-Bit</td>
</tr>
<tr>
<td>FPGA Fabric-to-Transceiver Interface Frequency (MHz) (2)</td>
<td>75-240, 37.5-187.5, 37.5-187.5, 75-240, 37.5-187.5, 75-240, 37.5-187.5</td>
</tr>
<tr>
<td>TX PCS Latency (FPGA Fabric-Transceiver Interface Clock Cycles)</td>
<td>4 - 5.5, 4 - 5.5, 4 - 5.5, 4 - 5.5, 4 - 5.5, 3 - 4, 3 - 4.5</td>
</tr>
<tr>
<td>RX PCS Latency (FPGA Fabric-Transceiver Interface Clock Cycles)</td>
<td>11 - 13, 7 - 9, 8 - 10, 11 - 13, 7 - 9, 3 - 4, 3 - 4.5</td>
</tr>
</tbody>
</table>

**Notes to Figure 1–51:**

1. The 10-bit configuration is listed in Figure 1–52.
2. The maximum data rate specification shown in Figure 1–51 is valid only for the -3 speed grade devices. For data rate specifications for other speed grades offered, refer to the Device Datasheet for Arria II Devices chapter.
3. When you enable byte SERDES, the maximum data is 3G; otherwise, it is 1.92G.
Figure 1–52 shows Arria II GX and GZ transceiver configurations allowed in Basic functional mode with a 10-bit wide PMA-to-PCS interface.

**Figure 1–52. Transceiver Configurations in Basic Mode with a 10-Bit Wide PMA-to-PCS Interface (Note 1)**

### Notes to Figure 1–52:

1. The 8-bit configuration is listed in Figure 1–51.
2. The maximum data rate specification shown in Figure 1–52 is valid only for the -3 speed grade devices with byte SERDES enabled. For data rate specifications for other speed grades offered, refer to the Device Datasheet for Arria II Devices chapter.
3. When you enable byte SERDES, the maximum data is 3G; otherwise, it is 1.92G.
Figure 1–53 shows Arria II transceiver configurations allowed in Basic double-width functional mode with a 16-bit PMA-PCS interface.

### Figure 1–53. Transceiver Configurations in Basic Double-Width Mode with a 16-Bit PMA-PCS Interface for Arria II Devices

<table>
<thead>
<tr>
<th>Functional Mode</th>
<th>Basic Double-Width 16-Bit PMA-PCS Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Rate (Gbps)</td>
<td>1.0 - 6.375 (3)</td>
</tr>
<tr>
<td>Number of Channels</td>
<td>x1, x4, x8</td>
</tr>
<tr>
<td>Low-Latency PCS</td>
<td></td>
</tr>
<tr>
<td>Word Aligner (Pattern Length)</td>
<td>Manual Alignment (8-, 16-, 32-Bit)</td>
</tr>
<tr>
<td>8B/10B Encoder/Decoder</td>
<td>Disabled</td>
</tr>
<tr>
<td>Rate Match FIFO</td>
<td>Disabled</td>
</tr>
<tr>
<td>Byte SERDES</td>
<td>Disabled</td>
</tr>
<tr>
<td>Data Rate (Gbps)</td>
<td>1.0 - 4.0 (GZ) or 5.1 (GX)</td>
</tr>
<tr>
<td>Byte Ordering</td>
<td>Disabled</td>
</tr>
<tr>
<td>FPGA Fabric-to-Transceiver Interface Width (4)</td>
<td>16-Bit, 32-Bit</td>
</tr>
<tr>
<td>FPGA Fabric-to-Transceiver Interface Frequency (MHz) (2)</td>
<td>62.5 - 250 (GZ) or 318.75 (GX)</td>
</tr>
<tr>
<td>TX PCS Latency (FPGA Fabric-Transceiver Interface Clock Cycles)</td>
<td>5 - 6</td>
</tr>
<tr>
<td>RX PCS Latency (FPGA Fabric-Transceiver Interface Clock Cycles)</td>
<td>11 - 13</td>
</tr>
</tbody>
</table>

### Notes to Figure 1–53:

1. The maximum data rate specification shown in Figure 1–53 is valid only for the -3 (fastest) speed grade devices.
2. The byte ordering block is available only if you select the word alignment pattern length of 16 or 32 bits.
3. Arria II GX I3 devices can support up to 6.375 Gbps. For more information, refer to the Device Datasheet for Arria II Devices.
4. The FPGA fabric-to-transceiver interface width of 32-bits applies to Arria II GZ devices only.
5. Byte SERDES is only supported for Arria II GZ devices in double-wide mode.
Figure 1–54 shows Arria II transceiver configurations allowed in Basic double-width functional mode with a 20-bit PMA-PCS interface.

**Notes to Figure 1–54:**
1. The maximum data rate specification shown in Figure 1–54 is valid only for the -3 (fastest) speed grade devices.
2. The byte ordering block is available only if you select the word alignment pattern length of 20 bits.
3. Arria II GX I3 devices can support up to 6.375 Gbps. For more information, refer to the Device Datasheet for Arria II Devices.
4. The FPGA fabric-to-transceiver interface width of 40-bits applies to Arria II GZ devices only.
5. Byte SERDES is only supported for Arria II GZ devices in double-wide mode.
Deterministic Latency

This mode is typically used to create a CPRI or Open Base Station Architecture Initiative Reference Point 3 (OBSAI RP3) interface to connect radio frequency (RF) processing remote radio heads located at the top of cell phone towers with the base band processing equipment typically found at the bottom of cell phone towers.

CPRI and OBSAI protocols have a requirement for the accuracy of the round trip delay measurement for single-hop and multi-hop connections to be within ± 16.276 ns. For single hops, the round trip delay may only vary within ± 16.276 ns. For multi-hop connections, the round trip variation is equal to ± 16.276 ns divided by the number of hops.

Deterministic latency is the only functional mode that allows 16-bit and 20-bit data on the PCS-to-PMA interface. This is to allow data rates of 2457.6, 3072, 4915.2, and 6144 Mbps for the CPRI protocol and to allow 3072 and 6144 Mbps data rates for the OBSAI protocol.

When you choose the deterministic latency protocol in the ALTGX MegaWizard Plug-In Manager, the bit-slip circuitry in the transmitter channel is automatically enabled and the RX phase compensation FIFO is automatically set to register mode. In addition, two extra ports are created—the rx_bitslipboundaryselectout output port from the receiver’s word aligner and the tx_bitslipboundaryselect input port for the transceiver bit-slip circuitry. You can also set the TX phase compensation FIFO in register mode.

In register mode, the phase compensation FIFO acts as a register and removes the uncertainty in latency. To ensure that the phase relationship between the low-speed parallel clock and the CMU PLL input reference clock is deterministic, you can enable the CMU PLL feedback path, which is only available in this mode. When the feedback path is enabled, you must provide an input reference clock to the CMU PLL that has the same frequency as the low-speed parallel clock.

The information on the rx_bitslipboundaryselectout[4:0] output port helps calculate the latency through the receiver datapath. Connect rx_bitslipboundaryselectout[4:0] to tx_bitslipboundaryselect[4:0] to cancel out the latency uncertainty.

The number of bits slipped in the receiver’s word aligner is shown on the rx_bitslipboundaryselectout[4:0] output port. In 8- or 10-bit channel width, the number of bits slipped in the receiver path is given out sequentially on this output. For example, if zero bits are slipped, the output on rx_bitslipboundaryselectout[4:0] shows a value of 0(5'b00000); if two bits are slipped, the output on rx_bitslipboundaryselectout[4:0] shows a value of 2 (5'b00010). In 16- or 20-bit channel width, the output is 19 minus the number of bits slipped. For example, if zero bits are slipped, the output on rx_bitslipboundaryselectout[4:0] shows a value of 19 (5'b10011); if two bits are slipped, the output on rx_bitslipboundaryselectout[4:0] shows a value of 17 (5'b10001).

You can slip zero to nine bits with 8- or 10-bit channel width and you can slip zero to 19 bits with 16- or 20-bit channel width.
Figure 1–55 shows the block diagram for deterministic latency.

**Notes to Figure 1–55:**

1. The transmitter is in bit-slip mode.
2. This block is optional in this mode.
3. The RX phase compensation FIFO is automatically set in register mode. However, you have the option to set the TX phase compensation FIFO in register mode, which is not set by default.
4. Typically, the 8B/10B encoder and decoder are used when you use deterministic latency to implement CPRI or OBSAI protocols. However, you have the option to disable this module.

**GbE**

IEEE 802.3 defines the 1000 Base-X PHY as an intermediate, or transition, layer that interfaces various physical media with the media access control (MAC) in a GbE system. It shields the MAC layer from the specific nature of the underlying medium. The 1000 Base-X PHY, which has a physical interface data rate of 1.25 Gbps, is divided into three sublayers—the physical coding sublayer (PCS), physical media attachment (PMA), and physical medium dependent (PMD). These sublayers interface with the MAC through the gigabit medium independent interface (GMII).
Figure 1–56 shows the 1000 Base-X PHY position in a GbE OSI reference model.

**Figure 1–56. 1000 Base-X PHY in a GbE OSI Reference Model**

In GbE functional mode, Arria II GX and GZ devices have built-in circuitry to support the following PCS and PMA functions defined in the IEEE 802.3 specification:

- 8B/10B encoding and decoding
- Synchronization
- Clock recovery from the encoded data forwarded by the receiver PMD
- Optional `rx_recovclkout` port enables the recovered clock at the pin level (use with the voltage-controlled crystal oscillator [VCXO])
- Serialization and deserialization

If you enabled the auto-negotiation state machine in the FPGA core with the rate match FIFO, refer to the “Rate Match FIFO in GbE Mode” on page 1–61.

Arria II GX and GZ transceivers do not have built-in support for some PCS functions; for example, auto-negotiation state machine, collision-detect, and carrier-sense. If required, you must implement these functions in a PLD logic array or external circuits.
Figure 1–57 shows the transceiver datapath when configured in GbE functional mode.

### Idle Ordered-Set Generation in GbE Mode

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

In GbE functional mode, any /Dx.y/ following a /K28.5/ comma is replaced by the transmitter with either a /D5.6/ (/I1/ ordered set) or a /D16.2/ (/I2/ ordered set), depending on the current running disparity. The exception is when the data following the /K28.5/ is /D21.5/ (/C1/ ordered set) or /D2.2/ (/C2/) ordered set. If the running disparity before the /K28.5/ is positive, an /I1/ ordered set is generated. If the running disparity is negative, an /I2/ ordered set is generated. 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.

/D14.3/, /D24.0/, and /D15.8/ are replaced by /D5.6/ or /D16.2/ (for /I1/ and /I2/ ordered sets). /D21.5/ (part of the /C1/ order set) is not replaced.
Figure 1–58 shows the automatic idle ordered set generation.

**Figure 1–58. Example of Automatic Ordered Set Generation**

<table>
<thead>
<tr>
<th>clock</th>
<th>tx_datain</th>
<th>tx_dataout</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>K28.5</td>
<td>K28.5</td>
</tr>
<tr>
<td></td>
<td>D14.3</td>
<td>D24.0</td>
</tr>
<tr>
<td></td>
<td>K28.5</td>
<td>D15.8</td>
</tr>
<tr>
<td></td>
<td>K28.5</td>
<td>D21.51</td>
</tr>
<tr>
<td></td>
<td>Dx.y</td>
<td></td>
</tr>
</tbody>
</table>

Ordered Set: /I1/ /I2/ /I2/ /C1/

**GbE Mode Reset Condition**

After de-assertion of the tx_digitalreset signal, the GbE transmitter automatically transmits three /K28.5/ comma code groups before transmitting user data on the tx_datain port. This could affect the synchronization state machine behavior at the receiver. Depending on when you start transmitting the synchronization sequence, there could be an even or odd number of /Dx.y/ code groups transmitted between the last of the three automatically sent /K28.5/ code groups and the first /K28.5/ code group of the synchronization sequence. If there is an even number of /Dx.y/ code groups received between these two /K28.5/ code groups, the first /K28.5/ code group of the synchronization sequence begins at an odd code group boundary. An IEEE802.3-compliant GbE synchronization state machine treats this as an error condition and goes into the Loss-of-Sync state.

Figure 1–59 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 /K28.5/ code group received at an odd code group boundary in cycle n + 3 takes the receiver synchronization state machine in Loss-of-Sync state. The first synchronization ordered-set /K28.5/Dx.y/ in cycles n + 3 and n + 4 are discounted and three additional ordered sets are required for successful synchronization.

**Figure 1–59. Example of Reset Condition in GbE Mode**

<table>
<thead>
<tr>
<th>clock</th>
<th>tx_digitalreset</th>
<th>tx_dataout</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>K28.5</td>
<td>K28.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K28.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K28.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dx.y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K28.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K28.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K28.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dx.y</td>
</tr>
</tbody>
</table>

**Word Aligner in GbE Mode**

The word aligner in GbE functional mode is configured in automatic synchronization state machine mode, which complies with the IEEE P802.3ae standard. The Quartus II software automatically configures the synchronization state machine to indicate synchronization when the receiver acquires three consecutive synchronization ordered sets. A synchronization ordered set is a /K28.5/ code group followed by an odd number of valid /Dx.y/ code groups. The fastest way for the receiver to achieve synchronization is to receive three continuous (/K28.5/, /Dx.y/) ordered sets.
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; a low on the **rx_syncstatus** port indicates that the lane has fallen out of synchronization. Each invalid code group increases the error count. The error count can be reduced by 1 if the state machine sees four continuous valid code groups. The receiver loses synchronization when it detects four invalid code groups separated by less than three valid code groups, or when it is reset.

## Rate Match FIFO in GbE Mode

In GbE mode, the rate match FIFO is capable of compensating up to ±100 PPM (200 PPM total) difference between the upstream transmitter and the local receiver reference clock. The GbE protocol requires the transmitter to send idle ordered sets `/I1/ (/K28.5/D5.6/) and `/I2/ (/K28.5/D16.2/) during inter-packet gaps, adhering to the rules listed in the IEEE P802.3ae specification.

The rate match operation begins after the synchronization state machine in the word aligner indicates synchronization has been acquired by driving the **rx_syncstatus** signal high. The rate match FIFO deletes or inserts both symbols of the `/I2/ ordered sets (which consist of `/K28.5/ and `/D16.2/) to prevent the rate match FIFO from overflowing or underflowing. It can insert or delete as many `/I2/ ordered sets as necessary to perform the rate match operation.

If you have the auto-negotiation state machine in the FPGA, note that the rate match FIFO is capable of inserting or deleting the first two bytes (`/K28.5//D2.2/) of `/C2/ ordered sets during auto-negotiation. However, the insertion or deletion of the first two bytes of `/C2/ ordered sets can cause the auto-negotiation link to fail. For more information, refer to the Altera Knowledge Base Support Solution.

The status flags **rx_rmfifodatadeleted** and **rx_rmfifodatainserted**, indicating rate match FIFO deletion and insertion events, respectively, are forwarded to the FPGA fabric. These two flags are asserted for two clock cycles for each deleted and inserted `/I2/ ordered set, respectively.

*Figure 1–60 shows an example of rate match FIFO deletion where three symbols must be deleted. Because the rate match FIFO can only delete `/I2/ ordered sets, it deletes two `/I2/ ordered sets (four symbols deleted).*

*Figure 1–60. Example of Rate Match Deletion in GbE Mode*
Figure 1–61 shows an example of rate match FIFO insertion in the case where one symbol must be inserted. Because the rate match FIFO can only insert /I2/ ordered sets, it inserts one /I2/ ordered set (two symbols inserted).

**Figure 1–61. Example of Rate Match Insertion in GbE Mode**

The rate match FIFO does not insert or delete code groups automatically to overcome FIFO empty or full conditions. In this case, the rate match FIFO asserts the rx_rmfifofull and rx_rmfifoempty flags for at least two recovered clock cycles to indicate rate match FIFO full and empty conditions, respectively. You must then assert the rx_digitalreset signal to reset the receiver PCS blocks.

### PCIe Mode

Intel Corporation has developed a PHY interface for the PCIe architecture specification to enable implementation of a PCIe-compliant physical layer device. This specification also defines a standard interface between the physical layer device and the media access control layer (MAC). Version 2.00 of the specification provides implementation details for a PCIe-compliant physical layer device at both Gen1 (2.5 Gbps) and Gen2 (5 Gbps) signaling rates.

Arria II GX and GZ transceivers support ×1, ×4, and ×8 lane configurations in PCIe functional mode at Gen1 (2.5 Gbps) data rates. Arria II GZ devices also support ×1 and ×4 lane configurations in PCIe functional mode at Gen2 (5.0 Gbps). In PCIe ×1 configuration, the PCS and PMA blocks of each channel are clocked and reset independently. PCIe ×4 and ×8 configurations support channel bonding for four-lane and eight-lane PCIe links, where the PCS and PMA blocks of all bonded channels share common clock and reset signals.

You can configure Arria II GX and GZ transceivers to implement a Version 2.00 PCIe-compliant PHY using one of the following methods:

- **PCIe Compiler**—This method allows you to use the Arria II GX and GZ devices built-in PCIe hard IP blocks to implement the PHY-MAC layer, Data Link layer, and Transaction layer of the PCIe protocol stack. In this mode, each Arria II GX and GZ transceiver channel uses a PIPE interface block that transfers data, control, and status signals between the PHY-MAC layer and the transceiver channel PCS and PMA blocks. The PIPE interface block is used only in this mode and cannot be bypassed.

- **ALTGX MegaWizard Plug-In Manager**—This method requires implementing the PHY-MAC layer, Data Link layer, and Transaction layer in the FPGA fabric using a soft IP. Use this method if you do not use the PCIe hard IP block. (You cannot access the PCIe hard IP block if you use this method.)
For descriptions of PCIe hard IP architecture and PCIe mode configurations allowed when using the PCIe hard IP block, refer to the *PCI Express Compiler User Guide*.

For more information about transceiver datapath clocking in different PCIe configurations, refer to the *Transceiver Clocking in Arria II Devices* chapter.

The transmitter datapath in PCIe mode consists of the:

- PIPE interface
- TX phase compensation FIFO
- Optional byte serializer (enabled for 16-bit and disabled for 8-bit FPGA fabric-transceiver interface)
- 8B/10B encoder
- 10:1 serializer
- Transmitter buffer with receiver detect circuitry

The receiver datapath in PCIe mode consists of the:

- Receiver input buffer with signal detect circuitry
- 1:10 deserializer
- Word aligner that implements PCIe-compliant synchronization state machine
- Optional rate match FIFO (clock rate compensation) that can tolerate up to 600 PPM frequency difference
- 8B/10B decoder
- Optional byte deserializer (enabled for 16-bit and disabled for 8-bit FPGA fabric-transceiver interface)
- RX phase compensation FIFO
- PIPE interface

Table 1–16 lists features supported in PCIe functional mode for Gen1 (2.5 Gbps) and Gen2 (5.0 Gbps) data rate configurations.

<table>
<thead>
<tr>
<th>Feature</th>
<th>2.5 Gbps (Gen1)</th>
<th>5.0 Gbps (Gen2) (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>×1, ×4, ×8 link configurations</td>
<td>✓</td>
<td>Only ×1 and ×4 are supported</td>
</tr>
<tr>
<td>PCIe-compliant synchronization state machine</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>±300 PPM (total 600 PPM) clock rate compensation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>8-bit FPGA fabric-transceiver interface</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>16-bit FPGA fabric-transceiver interface</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transmitter buffer electrical idle</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Receiver detection</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>8B/10B encoder disparity control when transmitting compliance pattern</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Power state management</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Receiver status encoding</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Table 1–16. Supported Features in PCIe Mode for Arria II Devices  (Part 2 of 2)

<table>
<thead>
<tr>
<th>Feature</th>
<th>2.5 Gbps (Gen1)</th>
<th>5.0 Gbps (Gen2) (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic switch between 2.5 Gbps and 5 Gbps signaling rate</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>Dynamically selectable transmitter margining for differential output voltage control</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>Dynamically selectable transmitter buffer de-emphasis of -3.5 db and -6 db</td>
<td>—</td>
<td>✓</td>
</tr>
</tbody>
</table>

Note to Table 1–16:
(1) For Arria II GZ devices only.

Figure 1–62 shows the Arria II GX and GZ transceiver datapath when configured in PCIe functional mode.

Figure 1–62. Arria II GX and GZ Transceiver Datapath in PCIe Mode  (Note 1)

Notes to Figure 1–62:
(1) The transceiver datapath clock varies between non-bonded (×1) and bonded (×4 and ×8) configurations in PCIe mode, described in the Transceiver Clocking for Arria II Devices chapter.
(2) The word aligner uses automatic synchronization state machine mode (10-Bit /K28.5/).
(3) This can be ×1, ×4, or ×8 at 2.5 Gbps or ×1 or ×4 at 5.0 Gbps (for Arria II GZ devices only).
(4) The high-speed serial clock is running at 1.25 GHz.
(5) The parallel clocks are running at 250 MHz.
(6) This clock is running at 125 MHz if the byte serializer and deserializer are used. Otherwise, this clock is running at 250 MHz.
(7) If you use the PCIe hard IP, you can enable the byte serializer and deserializer with an 8-bit FPGA fabric-to-transceiver interface running at 250 MHz or disabled with a 16-bit FPGA fabric-to-transceiver interface running at 125 MHz. Otherwise, these blocks are always disabled and your 16-bit FPGA fabric-to-transceiver interface is running at 125 MHz.
Besides transferring data, control, and status signals between the PHY-MAC layer and the transceiver, the PIPE interface block implements the following functions required in a PIPE-compliant physical layer device:

- Manages the PIPE power states
- Forces the transmitter buffer in the electrical idle state
- Initiates the receiver detect sequence
- Controls 8B/10B encoder disparity when transmitting compliance pattern
- Indicates completion of various PHY functions, such as receiver detection and power state transitions on the pipephydonestatus signal
- Encodes receiver status and error conditions on the pipestatus[2:0] signal as specified in the PIPE specification

The following subsections describe each Arria II GX and GZ transceiver function.

### Power State Management

The PCIe specification defines four power states that the physical layer device must support to minimize power consumption:

- **P0** is the normal operation state during which packet data is transferred on the PCIe link.
- **P0s, P1, and P2** are low-power states into which the physical layer must transition as directed by the PHY-MAC layer to minimize power consumption.

The PCIe specification provides the mapping of these power states to the long-term sample storage module (LTSSM) states specified in the PCIe Base Specification 2.0. The PHY-MAC layer is responsible for implementing the mapping logic between the LTSSM states and the four power states in the PCIe-compliant PHY.

The PIPE interface in Arria II GX and GZ transceivers provide an input port (powerdn[1:0]) to set the transceivers in one of the four power states, as shown in Table 1–17.

<table>
<thead>
<tr>
<th>powerdn [1:0] Values</th>
<th>Power State</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>2'b00</td>
<td>P0</td>
<td>Normal operation mode</td>
<td>Transmits normal data, transmits electrical idle, or enters into loopback mode</td>
</tr>
<tr>
<td>2'b01</td>
<td>P0s</td>
<td>Low recovery time saving state</td>
<td>Only transmits electrical idle</td>
</tr>
<tr>
<td>2'b10</td>
<td>P1</td>
<td>High recovery time power saving state</td>
<td>Transmitter buffer is powered down and can perform a receiver detect while in this state</td>
</tr>
<tr>
<td>2'b11</td>
<td>P2</td>
<td>Lowest power saving state</td>
<td>Transmits electrical idle or a beacon to wake up the downstream receiver</td>
</tr>
</tbody>
</table>

When the device transitions from the P0 power state to lower power states (P0s, P1, and P2), the PCIe specification requires the physical layer device to implement power saving measures. Arria II GX and GZ transceivers do not implement these power saving measures except when putting the transmitter buffer in electrical idle in the lower power states.
The PIPE interface block indicates a successful power state transition by asserting the pipephydonestatus signal for one parallel clock cycle as specified in the PCIe specification. The PHY-MAC layer must not request any further power state transition until the pipephydonestatus signal has indicated the completion of the current power state transition request.

Figure 1–63 shows an example waveform for a transition from the P0 to the P2 power state.

Figure 1–63. Example of Power State Transition from P0 to P2

The PCIe specification allows the PIPE interface to perform protocol functions such as receiver detect, loopback, and beacon transmission in specified power states only. This requires the PHY-MAC layer to drive the tx_detectrxloop and tx_forceelecidle signals appropriately in each power state to perform these functions. Table 1–18 lists the logic levels that the PHY-MAC layer must drive on the tx_detectrxloop and tx_forceelecidle signals in each power state.

Table 1–18. Logic Levels for the PHY-MAC Layer for Arria II Devices

<table>
<thead>
<tr>
<th>Power State</th>
<th>tx_detectrxloop Value</th>
<th>tx_forceelecidle Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0: normal mode</td>
<td>0: must be de-asserted</td>
</tr>
<tr>
<td></td>
<td>1: loopback mode</td>
<td>1: illegal mode</td>
</tr>
<tr>
<td>P0s</td>
<td>Don’t care</td>
<td>0: illegal mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: must be asserted in this state</td>
</tr>
<tr>
<td>P1</td>
<td>0: electrical state</td>
<td>0: illegal mode</td>
</tr>
<tr>
<td></td>
<td>1: receiver detect</td>
<td>1: must be asserted in this state</td>
</tr>
<tr>
<td>P2</td>
<td>Don’t care</td>
<td>De-asserted in this state for sending beacon. Otherwise asserted.</td>
</tr>
</tbody>
</table>

Transmitter Buffer Electrical Idle

The PCIe specification requires the transmitter buffer to be in electrical idle in the P1 power state, as shown in Table 1–18. During electrical idle, the transmitter buffer differential and common mode output voltage levels are compliant to the PCIe Base Specification 2.0 for both PCIe Gen1 and Gen2 data rates.

In Arria II GX and GZ transceivers, asserting the input signal tx_forceelecidle puts the transmitter buffer in that channel in the electrical idle state. Figure 1–64 shows the relationship between asserting the tx_forceelecidle signal and the transmitter buffer output on the tx_dataout port. Time T1 taken from the assertion of the tx_forceelecidle signal to the transmitter buffer reaching electrical idle voltage levels is a minimum of 8 ns. When in the electrical idle state, the PCIe protocol requires the transmitter buffer to stay in electrical idle for a minimum of 20 ns for both Gen1 and Gen2 data rates.
Receiver Detection

During the detect substate of the LTSSM state machine, the PCIe protocol requires the transmitter channel to perform a receiver detect sequence to detect if a receiver is present at the far end of each lane. The PCIe specification requires that a receiver detect operation be performed during the P1 power state where the transmitter output buffer is in electrical idle (tri-stated).

This feature requires the transmitter output buffer to be tri-stated (in electrical idle mode), have OCT utilization, and run at 125 MHz on the fixedclk signal. You can enable this feature in PCIe functional mode by setting the tx_forceelecidle and tx_detectrxloop ports to 1'b1.

When the tx_detectrxloop signal is asserted high in the P1 power state, the PIPE interface block sends a command signal to the transmitter buffer in that channel to initiate a receiver detect sequence. On receiving this command signal, the receiver detect circuitry creates a step voltage at the transmitter output buffer common mode voltage. If an active receiver (that complies with the PCIe input impedance requirements) is present at the far end, the time constant of the step voltage on the trace is higher compared to when the receiver is not present. The receiver-detect circuitry monitors the time constant of the step signal seen on the trace to decide if a receiver was detected or not.

If a far-end receiver is successfully detected, the PIPE interface block asserts the pipephydonestatus signal for one clock cycle and synchronously drives the pipestatus[2:0] signal to 3'b011. If a far-end receiver is not detected, the PIPE interface block asserts the pipephydonestatus signal for one clock cycle and synchronously drives the pipestatus[2:0] signal to 3'b000.

There is some latency after asserting the tx_detectrxloop signal before receiver detection is indicated on the pipephydonestatus port. In addition, the tx_forceelecilide port must be asserted at least 10 parallel clock cycles prior to the tx_detectrxloop port to ensure that the transmitter buffer is tri-stated.

For the receiver detect circuitry to function reliably, the AC-coupling capacitor on the serial link and the receiver termination values used in your system must be compliant to the PCIe Base Specification 2.0. Receiver detect circuitry communicates the status of the receiver detect operation to the PIPE interface block.
Figure 1–65 and Figure 1–66 show the receiver detect operation where a receiver was successfully detected and where a receiver was not detected, respectively.

### Figure 1–65. Receiver Detect Successful Operation

<table>
<thead>
<tr>
<th>powerdown[1:0]</th>
<th>2'b10(P1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx_detectxloopback</td>
<td></td>
</tr>
<tr>
<td>pipephydonestatus</td>
<td></td>
</tr>
<tr>
<td>pipestatus[2:0]</td>
<td>3'b000</td>
</tr>
<tr>
<td></td>
<td>3'b011</td>
</tr>
</tbody>
</table>

### Figure 1–66. Receiver Detect Unsuccessful Operation

<table>
<thead>
<tr>
<th>powerdown[1:0]</th>
<th>2'b10(P1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx_detectxloopback</td>
<td></td>
</tr>
<tr>
<td>pipephydonestatus</td>
<td></td>
</tr>
<tr>
<td>pipestatus[2:0]</td>
<td>3'b000</td>
</tr>
</tbody>
</table>

### Compliance Pattern Transmission Support

The LTSSM state machine can enter the polling.compliance substate where the transmitter must transmit a compliance pattern as specified in the PCIe Base Specification 2.0. The polling.compliance substate is intended to assess if the transmitter is electrically compliant with the PCIe voltage and timing specifications.

The compliance pattern is a repeating sequence of the following four code groups:

- /K28.5/
- /D21.5/
- /K28.5/
- /D10.2/

The PCIe protocol requires the first /K28.5/ code group of the compliance pattern to be encoded with negative current disparity. To satisfy this requirement, the PIPE interface block provides an input signal (tx_forcedispcompliance). A high level on the tx_forcedispcompliance signal forces the associated parallel transmitter data on the tx_datain port to transmit with a negative current running disparity.

For 8-bit transceiver channel width configurations, you must drive the tx_forcedispcompliance signal high in the same parallel clock cycle as the first /K28.5/ of the compliance pattern on the tx_datain port. For 16-bit transceiver channel width configurations, you must drive only the LSB of the tx_forcedispcompliance[1:0] signal high in the same parallel clock cycle as /K28.5/D21.5/ of the compliance pattern on the tx_datain port.
Figure 1–67 and Figure 1–68 show the required level on the tx_forcedispcompliance signal while transmitting the compliance pattern in 8-bit and 16-bit channel width configurations, respectively.

### Figure 1–67. Compliance Pattern Transmission Support, 8-Bit Wide Channel Configuration

<table>
<thead>
<tr>
<th>tx_datain[7:0]</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BC</td>
<td>B5</td>
<td>BC</td>
<td>A</td>
</tr>
<tr>
<td>tx_ctrldetect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tx_forcedispcompliance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Figure 1–68. Compliance Pattern Transmission Support, 16-Bit Wide Channel Configuration

<table>
<thead>
<tr>
<th>tx_datain[15:0]</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BC</td>
<td>A</td>
<td>BC</td>
<td>BC</td>
</tr>
<tr>
<td>tx_ctrldetect</td>
<td>01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tx_forcedispcompliance[1:0]</td>
<td>01</td>
<td>00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Receiver Status

The PCIe specification requires the PHY to encode the receiver status on a 3-bit RxStatus[2:0] signal. The PIPE interface block receives status signals from the transceiver channel PCS and PMA blocks and drives the status on the 3-bit output signal (pipestatus[2:0]) to the FPGA fabric. The encoding of the status signals on the pipestatus[2:0] port is compliant with the PCIe specification and is listed in Table 1–19.

### Table 1–19. Encoding of the Status Signals on the pipestatus[2:0] Port for Arria II Devices

<table>
<thead>
<tr>
<th>pipestatus[2:0]</th>
<th>Description</th>
<th>Error Condition Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>3'b000</td>
<td>Received data OK</td>
<td>N/A</td>
</tr>
<tr>
<td>3'b001</td>
<td>One SKP symbol added</td>
<td>5</td>
</tr>
<tr>
<td>3'b010</td>
<td>One SKP symbol deleted</td>
<td>6</td>
</tr>
<tr>
<td>3'b011</td>
<td>Receiver detected</td>
<td>N/A</td>
</tr>
<tr>
<td>3'b100</td>
<td>8B/10B decode error</td>
<td>1</td>
</tr>
<tr>
<td>3'b101</td>
<td>Elastic buffer (rate match FIFO) overflow</td>
<td>2</td>
</tr>
<tr>
<td>3'b110</td>
<td>Elastic buffer (rate match FIFO) underflow</td>
<td>3</td>
</tr>
<tr>
<td>3'b111</td>
<td>Received disparity error</td>
<td>4</td>
</tr>
</tbody>
</table>

### Note to Table 1–19:

(1) The PIPE interface follows the priority listed in Table 1–19 while encoding the receiver status on the pipestatus[2:0] port. Two or more error conditions; for example, 8B/10B decode error (code group violation), rate match FIFO overflow or underflow, or receiver disparity error, can occur simultaneously. When this happens, PIPE drives 3'b100 on the pipestatus[2:0] signal.
Fast Recovery Mode
The PCIe Base Specification fast training sequences (FTS) are used for bit and byte synchronization to transition from L0s to L0 (PIPE P0s to P0) power states. The PCIe specification requires the physical layer device to acquire bit and byte synchronization after you transition from L0s to L0 state within 16 ns to 4 μs.

If the Arria II GX and GZ receiver CDR is configured in Automatic Lock mode, the receiver cannot meet the PCIe specification of acquiring bit and byte synchronization within 4 μs due to the signal detect and PPM detector time. To meet this specification, each Arria II GX and GZ transceiver has built-in fast recovery circuitry that you can optionally enable in the ALTGX MegaWizard Plug-In Manager with the Enable fast recovery mode option.

Fast recovery circuitry controls the receiver CDR \( rx\_locktorefclk \) and \( rx\_locktodata \) signals to force the receiver CDR in LTR or LTD modes, by relying on the Electrical Idle Ordered Sets (EIOS), NFTS sequences received in L0 power state, and the signal detect signal from the receiver input buffer to control the receiver CDR lock mode. It is self-operational and does not require user inputs.

When you enable fast recovery mode, the \( rx\_locktorefclk \) and \( rx\_locktodata \) ports are not available in the ALTGX MegaWizard Plug-In Manager.

Electrical Idle Inference
The PCIe protocol allows inferring the electrical idle condition at the receiver instead of detecting the electrical idle condition using analog circuitry. PCIe Base Specification 2.0, section 2.4.3, specifies conditions to infer electrical idle at the receiver in various sub-states of the LTSSM state machine.

In all PCIe modes (×1, ×4, and ×8), each receiver channel PCS has an optional electrical idle inference module designed to implement the electrical idle inference conditions specified in the PCIe Base Specification 2.0.

You can enable the electrical idle inference module by selecting the Enable electrical idle inference functionality option in the ALTGX MegaWizard Plug-In Manager. This feature infers electrical idle depending on the logic level driven on the \( rx\_elecidleinfersel[2:0] \) input signal. The electrical idle inference module drives the \( pipeelecidle \) signal high in each receiver channel when an electrical idle condition is inferred. For the electrical idle inference module to correctly infer an electrical idle condition in each LTSSM substate, you must drive the \( rx\_elecidleinfersel[2:0] \) signal appropriately, as shown in Table 1–20.

<table>
<thead>
<tr>
<th>( rx_elecidleinfersel[2:0] )</th>
<th>LTSSM State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3'b100</td>
<td>L0</td>
<td>Absence of update FC or alternatively skip ordered set in 128 μs window</td>
</tr>
<tr>
<td>3'b101</td>
<td>Recovery.RcvrCfg</td>
<td>Absence of TS1 or TS2 ordered set in 1280 UI interval</td>
</tr>
<tr>
<td>3'b101</td>
<td>Recovery.Speed when successful speed negotiation = 1'b1</td>
<td>Absence of TS1 or TS2 ordered set in 1280 UI interval</td>
</tr>
</tbody>
</table>
The electrical idle inference module cannot detect an electrical idle exit condition based on the reception of the electrical idle exit ordered set (EIEOS), as specified in the PCIe Base Specification.

If you select the **Enable Electrical Idle Inference Functionality** option in the ALTGX MegaWizard Plug-In Manager and drive `rx_elecideleinfersel[2:0] = 3'b0xx`, the electrical idle inference block uses EIOS detection from the fast recovery circuitry to drive the `pipeelecidle` signal. Otherwise, the electrical idle inference module is disabled. In this case, the `rx_signaldetect` signal from the signal detect circuitry in the receiver input buffer is inverted and driven as the `pipeelecidle` signal.

### PCIe Cold Reset Requirements

The PCIe Base Specification 2.0 defines the following three types of conventional resets to the PCIe system components:

- Cold reset—fundamental reset after power up
- Warm reset—fundamental reset without removal and re-application of power
- Hot reset—in-band conventional reset initiated by higher layer by setting the hot reset bit in the TS1 or TS2 training sequences

Figure 1–69 shows the PCIe cold reset timing requirements.

#### Table 1–20. Electrical Idle Inference Conditions for Arria II Devices (Part 2 of 2)

<table>
<thead>
<tr>
<th><code>rx_elecideleinfersel[2:0]</code></th>
<th>LTSSM State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3'b110</td>
<td>Recovery.Speed when successful speed negotiation = 1'b0</td>
<td>Absence of an exit from electrical idle in 2000 UI interval</td>
</tr>
<tr>
<td>3'b111</td>
<td>Loopback.Active (as slave)</td>
<td>Absence of an exit from electrical idle in 128 μs window</td>
</tr>
</tbody>
</table>

---

**Figure 1–69. PCIe Cold Reset Requirements**

1. Power Rail
2. PERST#
3. TPVPERL 100 ms
4. Marker 1: Power becomes stable
5. Marker 2: PERST# gets de-asserted
6. Marker 3: Maximum time for the LTSSM to enter the Detect state
7. Marker 4: Maximum time for the link to become active
The following is the time taken by a PCIe port, implemented in an Arria II GX or GZ device, to go from the power-up to the link-active state:

- **Power-on reset**—begins after power rails become stable, which typically takes 12 ms
- **FPGA configuration and programming**—begins after power on reset. Configuration time depends on the FPGA density
- **Time taken from de-assertion of PERST# to link active**—typically takes 40 ms

To meet the PCIe specification of 200 ms from the power-on to the link-active state, the Arria II GX and GZ device configuration time must be less than 148 ms (200 ms to 12 ms for power on reset, 40 ms for the link to become active after PERST# de-assertion).

For the typical Arria II GX and GZ configuration times using the Fast Passive Parallel (FPP) configuration scheme at 125 MHz, refer to the *Device Datasheet for Arria II Devices*.

For more information about the FPP configuration scheme, refer to the *Configuration, Design Security, Remote System Upgrades in Arria II Devices* chapter.

Most flash memories available in the market can run up to 100 MHz. Altera recommends using a MAX II device to convert the 16-bit flash memory output at 62.5 MHz to 8-bit configuration data input to the Arria II GX and GZ devices at 125 MHz.

**SDI**

The Society of Motion Picture and Television Engineers (SMPTE) defines various SDI standards for the 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 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 2970 Mbps or 2967 Mbps.

Table 1–21 lists the data rates, refclk frequencies, and interface widths supported by Arria II GX and GZ transceivers in SDI mode.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Data Rate (Mbps)</th>
<th>Support refclk Frequencies (MHz)</th>
<th>FPGA Fabric-to-Transceiver Width</th>
<th>Byte Serializer/Deserializer Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD</td>
<td>1483.5</td>
<td>74.175, 148.35</td>
<td>20-bit, 10-bit</td>
<td>Used, Not used</td>
</tr>
<tr>
<td>1485</td>
<td>74.25, 148.5</td>
<td>20-bit, 10-bit</td>
<td>Used, Not used</td>
<td></td>
</tr>
</tbody>
</table>
Figure 1–70 shows the transceiver datapath when configured in SDI mode.

Figure 1–70. SDI Mode Datapath (Note 1)

Notes to Figure 1–70:
(1) For the frequency, data rate, and interface width supported, refer to Table 1–21 on page 1–72.
(2) This allows the fabric-to-transceiver interface to run below the maximum interface frequency. For more information, refer to Table 1–21 on page 1–72.
(3) The word aligner uses bit-slip mode. However, this block is not useful because word alignment and framing occurs after de-scrambling. Altera recommends driving the ALTGXB megafunction  rx_bitslip signal low to prevent the word aligner from inserting bits in the received data stream.

In HD-SDI mode, the transmitter is purely a parallel-to-serial converter. SDI transmitter functions, such as scrambling and cyclical redundancy check (CRC) code generation, must be implemented in the FPGA logic array. Similarly, SDI receiver functions, such as de-scrambling, framing, and CRC checker, must be implemented in the FPGA logic array.
SRIO

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.

The SRIO physical layer specification defines three line rates—1.25 Gbps, 2.5 Gbps, and 3.125 Gbps. It also defines two link widths—single-lane (×1) and bonded four-lane (×4) at each line rate. Arria II GX and GZ transceivers support only single-lane (×1) configuration at all three line rates. You can instantiate four ×1 channels configured in SRIO mode to achieve one non-bonded ×4 SRIO link. The four receiver channels in this ×4 SRIO link do not have lane alignment or deskew capability.

Arria II GX and GZ transceivers, when configured in SRIO functional mode, provide the following PCS and PMA functions:

- 8B/10B encoding and decoding
- Word alignment
- Lane synchronization state machine
- Clock recovery from the encoded data
- Serialization and deserialization

Arria II GX and GZ transceivers do not have built-in support for some PCS functions, such as 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.
Figure 1–71 shows the ALTGX transceiver datapath when configured in SRIO mode.

Figure 1–71. SRIO Mode Datapath

Notes to Figure 1–71:

(1) This allows the fabric-to-transceiver interface to run below the maximum interface frequency and is always enabled for SRIO functional mode.
(2) The word aligner uses the automatic synchronization state machine (10 bit /K28.5/) and is compliant with the SRIO protocol.
(3) This module is optional.
(4) This can run at 1.25, 2.5, or 3.125 Gbps.
(5) This is running at half the rate of the data rate.
(6) This is running at 62.5 MHz for 1.25 Gbps data rate, 125 MHz for 2.5 Gbps data rate, or 156.25 MHz for 3.125 Gbps data rate.

In SRIO mode, the ALTGX MegaWizard Plug-In Manager automatically defaults the synchronization state machine to indicate synchronization (a high logic level on the rx_syncstatus port) when the receiver acquires 127 K28.5 (10'b0101111100 or 10'b1010000011) synchronization code groups without receiving an intermediate invalid code group. When synchronized, the state machine indicates loss of synchronization (a low logic level on the rx_syncstatus port) when it detects three invalid code groups separated by less than 255 valid code groups, or when it is reset.

SRIO only allows one insertion or deletion of the skip character /R/ from the /K/, /R/, /R/, /R/ clock compensation sequence. However, the Arria II GX and GZ rate match FIFO may perform multiple insertions or deletions if the PPM difference is more than the ±200 PPM range. Ensure that the PPM difference in your system is less than ±200 ppm.
SONET/SDH

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.

You can use Arria II GX and GZ transceivers 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 an A1A2 or A1A1A2A2 pattern.

In SONET/SDH systems, A1 is defined as 8’hF6 and A2 is defined as 8’h28. Transport overhead bytes A1 and A2 are used for restoring 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 μs. In an OC-12 system, 12 A1 bytes are followed by 12 A2 bytes. Similarly, in an OC-48 system, 48 A1 bytes are followed by 48 A2 bytes. OC-96 systems are for Arria II GZ devices only and have 96 A1 bytes followed by 96 A2 bytes.

Arria II GX transceivers are designed to support the protocols OC-12 at 622 Mbps with 8-bit channel width and OC-48 at 2488.32 Mbps with 16-bit channel width. Arria II GZ transceivers are designed to support the protocol OC-96 at 4,967 Mbps.

Figure 1–72 shows the transceiver datapath when configured in SONET/SDH OC-12 mode.

Figure 1–72. SONET/SDH OC-12 Datapath
SONET/SDH OC-48 Datapath

Figure 1–73 shows the transceiver datapath when configured in SONET/SDH OC-48 mode.

Figure 1–73. SONET/SDH OC-48 Datapath

SONET/SDH OC-96 Datapath

Figure 1–74 shows the transceiver datapath when configured in SONET/SDH OC-96 mode.

Figure 1–74. SONET/SDH OC-96 Datapath
Unlike Ethernet, where the LSB of the parallel data byte is transferred first, SONET/SDH requires the MSB to be transferred first. To facilitate the MSB-to-LSB transfer, you must enable the **Flip Transmitter input data bits** and **Flip Receiver output data bits** options in the ALTGX MegaWizard Plug-In Manager.

Depending on whether data bytes are transferred MSB-to-LSB or LSB-to-MSB, you must select the appropriate word aligner settings in the ALTGX MegaWizard Plug-In Manager.

**Word Aligner in SONET/SDH Mode**

The word aligner in SONET/SDH functional mode is configured in manual alignment mode. You can configure the word aligner to either align to a 16-bit A1A2 pattern or a 32-bit A1A1A2A2 pattern, 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; a high level configures the word aligner to align to a 32-bit A1A1A2A2 pattern.

In OC-96 configurations, the word aligner is only allowed to align to an A1A1A2A2 pattern, so the 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.

You can also configure the word aligner to flip the word alignment pattern bits programmed in the ALTGX MegaWizard Plug-In Manager and compare them with the incoming data for alignment. This feature offers flexibility to the SONET backplane system for either a MSB-to-LSB or LSB-to-MSB data transfer. **Table 1–22** lists word alignment patterns that you must program in the ALTGX MegaWizard Plug-In Manager based on the bit-transmission order and the word aligner bit-flip option.

<table>
<thead>
<tr>
<th>Serial Bit Transmission Order</th>
<th>Flip the Word Alignment Pattern Bits</th>
<th>Word Alignment Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB-to-LSB</td>
<td>On</td>
<td>16’hF628</td>
</tr>
<tr>
<td>MSB-to-LSB</td>
<td>Off</td>
<td>16’h146F</td>
</tr>
<tr>
<td>LSB-to-MSB</td>
<td>Off</td>
<td>16’h28F6</td>
</tr>
</tbody>
</table>

**OC-48 and OC-96 Byte Serializer and Deserializer**

The OC-48 and OC-96 transceiver datapath 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 “**Byte Serializer**” on page 1–14 and “**Byte Deserializer**” on page 1–43, respectively.

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

**Byte Ordering in SONET/SDH OC-48 Mode**

Because of byte deserialization, the MSByte of a word might appear at the rx_dataout port along with the LSByte of the next word. In a SONET/SDH OC-48 configuration, you can use the byte ordering block that is built into the datapath to perform byte ordering. Byte ordering in a SONET/SDH OC-48 configuration is in word alignment-based mode, where the byte ordering block is triggered by the rising edge of the rx_syncstatus signal.

At the rising edge of the rx_syncstatus signal, the byte ordering block compares the LSByte coming out of the byte deserializer with the A2 byte of the A1A2 alignment pattern. If the LSByte coming out of the byte deserializer does not match the A2 byte set in the ALTGX MegaWizard Plug-In Manager, the byte ordering block inserts a PAD character, as shown in Figure 1–75. Inserting this PAD character enables the byte ordering block to restore the correct byte order.

---

The PAD character is defaulted to the A1 byte of the A1A2 alignment pattern.

---

**Figure 1–75. SONET/SDH OC-48 Byte Ordering Example**

![Diagram of byte ordering example](image)

---

**XAUI**

Use this functional mode for XAUI, HiGig, or HiGig+ protocols.

The XAUI is an optional, self-managed interface that you can insert between the reconciliation sublayer and the PHY layer to transparently extend the physical reach of the XGMII.
Figure 1–76 shows the relationships between the XGMII and XAUI layers.

The XGMII interface consists of four 8-bit lanes. At the transmit side of the XAUI interface, the data and control characters are converted within the XGMII extender sublayer (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/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 are 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.

Arria II GX and GZ transceivers configured in XAUI mode provide the following protocol features:

- **XGMII-to-PCS code conversion at the transmitter**—The 8B/10B encoder in the Arria II GX and GZ transmitter datapath is controlled by a transmitter state machine that maps various 8-bit XGMII codes to 10-bit PCS code groups. This state machine complies with the IEEE P802.3ae PCS transmit source state diagram.

- **PCS-to-XGMII code conversion at the receiver**—The 8B/10B decoder in the Arria II GX and GZ receiver datapath is controlled by a XAUI receiver state machine that converts received PCS code groups into specific 8-bit XGMII codes.

- **8B/10B encoding and decoding**

- **IEEE P802.3ae-compliant synchronization state machine**

- **±100 PPM clock rate compensation**

- **Channel deskew of four lanes of the XAUI link**
Figure 1–77 shows the ALTGX megafuction transceiver datapath when configured in XAUI mode.

**Figure 1–77. Transceiver Datapath in XAUI Mode**

**Notes to Figure 1–77:**

1. This allows the fabric-to-transceiver interface to run below the maximum interface frequency.
2. The word aligner uses the automatic synchronization state machine (10-bit /K28.5/).
3. This is running at half the rate of the data rate.
Word Aligner in XAUI Mode
The word aligner in XAUI functional mode is configured in automatic synchronization state machine mode that is compliant to the PCS synchronization state diagram specified in section 8 of the IEEE P802.3ae specification. The Quartus II software automatically configures the synchronization state machine to indicate synchronization when the receiver acquires four /K28.5/ comma code groups without intermediate invalid code groups.

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. The receiver loses synchronization when it detects four invalid code groups separated by less than four valid code groups, or when it is reset.

Deskew FIFO in XAUI Mode
The XAUI protocol requires the physical layer device to implement deskew circuitry to align all four channels. The skew introduced in the physical medium and the receiver channels can cause the /A/ code groups to be received misaligned with respect to each other. To enable the deskew circuitry at the receiver to align the four channels, the transmitter sends an /A/ (/K28.3/) code group simultaneously on all four channels during an inter-packet gap (IPG). The deskew operation begins only after link synchronization is achieved on all four channels as indicated by a high level on the rx_syncstatus signal from the word aligner in each channel. Until the first /A/ code group is received, the deskew FIFO read and write pointers in each channel are not incremented. After the first /A/ code group is received, the write pointer starts incrementing for each word received but the read pointer is frozen. If the /A/ code group is received on each of the four channels in 10 recovered clock cycles of each other, the read pointer of all four deskew FIFOs is released simultaneously, aligning all four channels.

The deskew FIFO operation in XAUI functional mode is compliant to the PCS deskew state machine diagram specified in 8 of the IEEE P802.3ae specification.
Figure 1–78 shows lane skew at the receiver input and how the deskew FIFO uses the /A/ code group to align the channels.

Figure 1–78. Deskew FIFO–Lane Skew at the Receiver Input

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>K</th>
<th>K</th>
<th>R</th>
<th>A</th>
<th>K</th>
<th>K</th>
<th>R</th>
<th>R</th>
<th>K</th>
<th>K</th>
<th>R</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lane 1</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>A</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Lane 2</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>A</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Lane 3</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>A</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

Lanes Skew at Receiver Input

<table>
<thead>
<tr>
<th>Lane 0</th>
<th>K</th>
<th>K</th>
<th>R</th>
<th>A</th>
<th>K</th>
<th>K</th>
<th>R</th>
<th>R</th>
<th>K</th>
<th>K</th>
<th>R</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lane 1</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>A</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Lane 2</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>A</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Lane 3</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>A</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
<td>K</td>
<td>K</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

Lanes are Deskewed by Aligning the /A/ Code Groups

After alignment of the first ||A|| column, if three additional aligned ||A|| columns are observed at the output of the deskew FIFOs of the four channels, the rx_channelaligned signal is asserted high, indicating channel alignment is acquired. After acquiring channel alignment, if four misaligned ||A|| columns are seen at the output of the deskew FIFOs in all four channels with no aligned ||A|| columns in between, the rx_channelaligned signal is de-asserted low, indicating loss of channel alignment.

Rate Match FIFO in XAUI Mode

In XAUI mode, the rate match FIFO is capable of compensating up to ±100 PPM (200 PPM total) difference between the upstream transmitter and the local receiver reference clock. The XAUI protocol requires the transmitter to send /R/ (/K28.0/) code groups simultaneously on all four lanes (denoted as the ||R|| column) during inter-packet gaps, adhering to rules listed in the IEEE P802.3ae specification.

The rate match operation begins after rx_syncstatus and rx_channelaligned are asserted. The rx_syncstatus signal is from the word aligner, indicating that synchronization is acquired on all four channels; the rx_channelaligned signal is from the deskew FIFO, indicating channel alignment.

The rate match FIFO looks for the ||R|| column (simultaneous /R/ code groups on all four channels) and deletes or inserts ||R|| columns to prevent the rate match FIFO from overflowing or underrunning. The rate match FIFO can insert or delete as many ||R|| columns as necessary to perform the rate match operation.

The rx_rmfifodatadeleted and rx_rmfifodatased inserted flags indicate rate match FIFO deletion and insertion events, respectively, and are forwarded to the FPGA fabric. If an ||R|| column is deleted, the rx_rmfifodatadeleted flag from each of the four channels goes high for one clock cycle per deleted ||R|| column. If an ||R|| column is inserted, the rx_rmfifodatased inserted flag from each of the four channels goes high for one clock cycle per inserted ||R|| column.
Figure 1–79 shows an example of rate match deletion in the case where three \(|R|\) columns must be deleted.

**Figure 1–79. Example of Rate Match Deletion in XAUI Mode**

<table>
<thead>
<tr>
<th>First</th>
<th>Second</th>
<th>Third</th>
<th>Fourth</th>
</tr>
</thead>
<tbody>
<tr>
<td>datain[0]</td>
<td>K28.5</td>
<td>K28.3</td>
<td>K28.5</td>
</tr>
</tbody>
</table>

Figure 1–80 shows an example of rate match insertion in the case where two \(|R|\) columns must be inserted.

**Figure 1–80. Example of Rate Match Insertion in XAUI Mode**

<table>
<thead>
<tr>
<th>First</th>
<th>Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataout[1]</td>
<td>K28.5</td>
</tr>
<tr>
<td>dataout[0]</td>
<td>K28.5</td>
</tr>
</tbody>
</table>

The rate match FIFO does not insert or delete code groups automatically to overcome FIFO empty or full conditions. In this case, the rate match FIFO asserts the `rx_rmfifofull` and `rx_rmfifoempty` flags for at least three recovered clock cycles to indicate rate match FIFO full and empty conditions, respectively. You must then assert the `rx_digitalreset` signal to reset the receiver PCS blocks.
Test Modes

Arria II GX and GZ devices provide various loopback options, pattern generators, and verifiers that allow you to ensure the working of different functional blocks in the transceiver channel. These modes include:

- Serial loopback
- Reverse serial loopback
- Reverse serial pre-CDR loopback
- PCIe reverse parallel loopback
- BIST and PRBS Modes

If you generate a **Transmitter-only** or **Receiver-only** configuration and enable loopback mode, you will have an extra port that you must connect to the transceiver’s counterpart port. These ports are described in Table 1–31 on page 1–98 (the PMA port list).

Serial Loopback

This option is available for all functional modes except PCIe mode. Figure 1–81 shows the datapath for serial loopback.

![Serial Loopback Datapath](image)

The data from the FPGA fabric passes through the transmitter channel and loops back to the receiver channel, bypassing the receiver input buffer. The received data is available to the FPGA logic for verification. Using this option, you can check the operation for all enabled PCS and PMA functional blocks in the transmitter and receiver channels.

When you enable the serial loopback option, the ALTGX MegaWizard Plug-In Manager provides the `rx_seriallpbken` port to dynamically enable serial loopback on a channel-by-channel basis when the signal is asserted high.
When you enable serial loopback, the transmitter channel sends the data to both the tx_dataout output port and the receiver channel. The differential output voltage on the tx_dataout ports is based on the selected VOD settings. The looped back data is received by the receiver CDR and then timed again through different clock domains. You must provide an alignment pattern for the word aligner to enable the receiver channel to retrieve the byte boundary.

**Reverse Serial Loopback**

Reverse serial loopback is available in Basic functional mode only and is often implemented when using a bit error rate tester (BERT) on the upstream transmitter. In this mode, the data is received through the rx_datain port, timed again through the receiver CDR, and sent out to the tx_dataout port. The received data is also available to the FPGA logic. You can enable the reverse serial loopback option using the ALTGX MegaWizard Plug-In Manager. Unlike other loopback modes, there is no dynamic pin control to enable or disable reverse serial loopback.

Figure 1–82 shows the transceiver channel datapath for reverse serial loopback mode.

**Figure 1–82. Reverse Serial Loopback Datapath (Note 1)**

![Reverse Serial Loopback Datapath Diagram](image)

**Note to Figure 1–82:**

(1) The only active block of the transmitter channel is the transmitter buffer.

You can change the output differential voltage on the transmitter buffer through the ALTGX MegaWizard Plug-In Manager. However, you cannot alter the pre-emphasis settings for the transmitter buffer.
**Reverse Serial Pre-CDR Loopback**

Reverse serial pre-CDR loopback is available in Basic functional mode only. In this mode, the data received through the `rx_datain` port is looped back to the `tx_dataout` port before the receiver CDR. The received data is also available to the FPGA logic. You can enable the reverse serial pre-CDR loopback option using the ALTGX MegaWizard Plug-In Manager. Unlike other loopback modes, there is no dynamic pin control to enable or disable reverse serial pre-CDR loopback.

Figure 1–83 shows the transceiver channel datapath for reverse serial pre-CDR loopback mode.

**Figure 1–83. Reverse Serial Pre-CDR Loopback Datapath  (Note 1)**

![Datapath Diagram](image)

**Note to Figure 1–83:**
1. The only active block of the transmitter channel is the transmitter buffer.

You can change the VOD on the transmitter buffer through the ALTGX MegaWizard Plug-In Manager. However, you cannot change the pre-emphasis settings for the transmitter buffer.
**PCIe (Reverse Parallel Loopback)**

PCIe reverse parallel loopback is only available in PCIe functional mode for the Gen1 and Gen2 data rates. As shown in Figure 1–84, the received serial data passes through the receiver CDR, deserializer, word aligner, and rate match FIFO buffer. The data is then looped back to the transmitter serializer and transmitted out through the `tx_dataout` port. The received data is also available to the FPGA fabric through the `rx_dataout` port. This loopback mode is compliant with the PCIe Base Specification 2.0. To enable PCIe reverse parallel loopback mode, assert the `tx_detectrxloop` port.

**Built-In Self Test (BIST) and Pseudo Random Binary Sequence (PRBS)**

Each transceiver channel in Arria II GX and GZ devices contain a pattern generator and a pattern verifier circuit. Using these patterns, you can verify the functionality of the functional blocks in the transceiver channel without requiring user logic. The functionality is provided as an optional mechanism for debugging transceiver channels. To use the Arria II GX and GZ pattern generator and verifier, use the pattern BIST and PRBS sub-protocols under Basic functional mode.
BIST mode allows you to verify the complete PCS blocks for both the transmitter and receiver channel. This mode is available only with a built-in 16-bit incremental pattern generator and verifier; therefore, you must set the channel width to 16 bits in this mode. The incremental pattern 00-FF is looped back to the receiver channel at the PCS functional block boundary before the PMA and is sent out to the tx_dataout port.

The received data is verified by the verifier, but is not available in the FPGA fabric. The $V_{OD}$ of the transmitted serial data on the tx_dataout port is based on the selected $V_{OD}$ settings.
Figure 1–86 shows the datapath for the PRBS patterns. The generated pattern is sent to the serializer. The verifier checks the data from the word aligner.

**Figure 1–86. Datapath for the PRBS Mode**

The PRBS mode has two pattern generator options, selectable in the BIST tab of the MegaWizard Plug-In Manager when you choose PRBS as a sub protocol under Basic functional mode.

- **PRBS7, PRBS8, PRBS10, and PRBS23 generator and verifier**—This is the generator and verifier interface with the serializer and deserializer in the PMA blocks. 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 repeats after completing an iteration. The number of bits the PRBSx pattern sends before repeating the pattern is \((2^x-1)\) bits. This mode is available as a sub protocol under Basic functional mode.

- **High-frequency and low-frequency pattern generator**—The high-frequency patterns generate alternate ones and zeros and the low-frequency patterns generate five ones and five zeroes in single width mode, and ten ones and ten zeroes in double width mode. These patterns do not have a corresponding verifier.

Table 1–23 lists various PRBS patterns and corresponding word alignment patterns.

**Table 1–23. Patterns in PRBS Mode for Arria II Devices (Part 1 of 2)**

<table>
<thead>
<tr>
<th>Patterns</th>
<th>Polynomial</th>
<th>Channel Width of 8-Bit ((\ell))</th>
<th>Word Alignment Pattern</th>
<th>Maximum Data Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRBS 7</td>
<td>(x^7 + x^6 + 1)</td>
<td>8 bit</td>
<td>16'h3040</td>
<td>2.5</td>
</tr>
<tr>
<td>PRBS 8</td>
<td>(x^8 + x^7 + 1)</td>
<td>8 bit</td>
<td>16'hFF5A</td>
<td>2.5</td>
</tr>
<tr>
<td>PRBS 10</td>
<td>(x^{10} + x^7 + 1)</td>
<td>10 bit</td>
<td>10'h3FF</td>
<td>3.125</td>
</tr>
<tr>
<td>PRBS 23</td>
<td>(x^{23} + x^{18} + 1)</td>
<td>8 bit</td>
<td>16'hFFFF</td>
<td>2.5</td>
</tr>
</tbody>
</table>

**Note to Figure 1–86:**

1. Serial loopback can be dynamically enabled through the `rx_seriallpbken` port.
Table 1–23. Patterns in PRBS Mode for Arria II Devices (Part 2 of 2)

<table>
<thead>
<tr>
<th>Patterns</th>
<th>Polynomial</th>
<th>Channel Width of 8-Bit (1)</th>
<th>Word Alignment Pattern</th>
<th>Maximum Data Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>High frequency (1)</td>
<td>1010101010</td>
<td>8 or 10 bit</td>
<td>NA</td>
<td>2.5 for 8-bit pattern and 3.125 for 10-bit pattern</td>
</tr>
<tr>
<td>Low frequency (1)</td>
<td>0000011111</td>
<td>10 bit</td>
<td>NA</td>
<td>3.125</td>
</tr>
</tbody>
</table>

Note to Table 1–23:
(1) A verifier is not available for the specified patterns.

Figure 1–87 shows the enabled input and output ports of the pattern generator and pattern verifier.

**Figure 1–87. Input and Output Ports for the BIST and PRBS Modes**

Note to Figure 1–87:
(1) rx_serialpbken is optional.

You can reset the PRBS pattern generator and verifier by asserting the tx_digitalreset and rx_digitalreset signals, respectively.

rx_digitalreset does not reset the BIST output signals when the following conditions are true:

- pll_powerdown is high in BIST mode
- pll_powerdown or rx_analogreset is high in PRBS mode

### Dynamic Reconfiguration

Dynamic reconfiguration allows you to reconfigure the transceiver block without reconfiguring the FPGA. For hot-plug or open-standard systems, this feature allows you to support multiple data rates or standards without reconfiguring the system. For all systems, it allows you to make changes to the bit error rate to compensate in-system for the effects of process and temperature.

For more information, refer to *AN 558: Implementing Dynamic Reconfiguration in Arria II Devices.*
Each transceiver channel has multiple physical medium attachment controls that you can program to achieve the desired bit error ratio (BER) for your system. When you enable the dynamic reconfiguration feature, you can reconfigure the following portions of each transceiver channel dynamically (one channel at a time) without powering down the other transceiver channels or the FPGA fabric of the device:

- Transmit and receive analog settings
- Transmit data rate in multiples of 1, 2, and 4
- Channel and clock multiplier unit PLL
- CMU PLL only

The dynamic reconfiguration controller is a soft IP that uses FPGA-fabric resources. You can use only one dynamic reconfiguration controller per transceiver block. You cannot use the dynamic reconfiguration controller to control multiple Arria II GX and GZ devices or off-chip interfaces. Figure 1–88 shows the conceptual view of the dynamic reconfiguration controller architecture.

**Figure 1–88. Block Diagram of the Dynamic Reconfiguration Controller**

Notes to Figure 1–88:

1. The PMA control ports consist of the $V_{OD}$ controls, pre-emphasis controls, DC gain controls, and manual equalization controls.
2. Only PMA reconfiguration mode supports manual equalization controls.
The dynamic reconfiguration controller requires input from one of the following:

- Its input ports through user logic where they are translated to the address and data bus inside the controller. The address and data bus are then converted into serial data and forwarded to the selected transceiver channel.

- A Memory Initialization File (.mif) where the controller receives 16-bit words from the .mif that you generate and sends this information to the transceiver channel selected.

The different modes of dynamic reconfiguration are:

- PMA settings reconfiguration, available for the following PMA settings:
  - Pre-emphasis settings
  - Equalization settings (channel reconfiguration mode does not support equalization settings)
  - DC gain settings
  - $V_{OD}$ settings

- Receiver offset cancellation

Process variations create offsets in analog circuit voltages, pushing them outside the expected range. The Arria II GX and GZ devices provide an offset cancellation circuit per receiver channel to counter the offset variations due to process.

Calibration of the offset cancellation circuit is done at power-up. The receiver input buffer and receiver CDR require offset calibration. Offset cancellation is automatically executed whenever the device is powered on. The control logic for offset cancellation is integrated into the dynamic reconfiguration controller.

The offset cancellation for the receiver channels option is automatically enabled in both the ALTGX and ALTGX_RECONFIG MegaWizard Plug-In Managers for Receiver, Transmitter, and Receiver only configurations. It is not available for Transmitter only configurations.

When offset cancellation is automatically enabled, you must instantiate the dynamic reconfiguration controller to connect the reconfiguration ports created by the ALTGX MegaWizard Plug-In Manager.

For more information about implementing the ALTGX_RECONFIG MegaWizard Plug-In Manager, refer to AN 558: Implementing Dynamic Reconfiguration in Arria II Devices.

You must always connect the ALTGX_RECONFIG instance to the ALTGX (with receiver channels) instance in your design. Connect the `reconfig_fromgxb`, `reconfig_togxb`, and necessary clock signals to both the ALTGX_RECONFIG and ALTGX (with receiver channels) instances. For transmitter-only configuration, the ALTGX_RECONFIG must also be connected to the ALTGX either by selecting a dynamic reconfiguration mode or by instantiating a dummy receiver-only ALTGX instance in each side of the device.

The offset cancellation process changes the transceiver reset sequence. For more information, refer to the Reset Control and Power Down chapter.
Transceiver channel reconfiguration—for transceiver channels, dynamic reconfiguration involves the reconfiguration of the following:

- Data Rate Reconfiguration—achievable by switching between two TX PLLs set to different data rates or reconfiguring the RX PLLs or reconfiguring the local dividers in the transmit side

Ensure that the functional mode of the transceiver channel supports the reconfigured data rate.

- CMU PLL Reconfiguration
- Functional Mode Reconfiguration

Ensure that the various clocks involved support the transition.

Transceiver Port List

You instantiate the Arria II GX and GZ transceivers with the ALTGX megafunction instance in the Quartus II MegaWizard Plug-In Manager. The ALTGX megafunction instance allows you to configure the transceivers for your intended protocol and select optional control and status ports to and from the instantiated transceiver channels.

These signals are available if you enable the block associated with them.

Table 1–24 lists the CMU port names and descriptions for the ALTGX megafunction.

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pll_inclk</td>
<td>Input</td>
<td>Input reference clock for the CMU PLL.</td>
</tr>
<tr>
<td>pll_powerdown</td>
<td>Input</td>
<td>Asynchronous active-high signal to power down both CMU PLLs. The minimum pulse width for this signal is specified in the Device Datasheet for Arria II Devices chapter. Note: Asserting the pll_powerdown signal does not power down the refclk buffers. Note: While each CMU PLL has its own pll_powerdown port, the ALTGX MegaWizard Plug-In Manager instantiation provides only one port per transceiver block. This port power downs one or both CMU PLLs (if used).</td>
</tr>
<tr>
<td>coreclkout</td>
<td>Output</td>
<td>A low-speed parallel clock generated by the CMU0 clock divider for bonded channel configurations. This signal is generated by the CMU0 clock divider in the master transceiver block in ×8 bonded channel configurations and is not available in non-bonded channel configurations. If the byte serializer block is enabled in bonded channel modes, the coreclkout clock output is half the frequency of the low-speed parallel clock. Otherwise, the coreclkout clock output is the same frequency as the low-speed parallel clock. You can also use this clock on the write and read clock ports of the TX phase compensation FIFOs in all bonded channels if tx_coreclk is not enabled in the ALTGX MegaWizard Plug-In Manager.</td>
</tr>
<tr>
<td>pll_locked</td>
<td>Output</td>
<td>Asynchronous active-high signal to indicate whether the CMU PLL is locked.</td>
</tr>
</tbody>
</table>
Table 1–25 lists the word aligner port names and descriptions for the ALTGX megafuntion.

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_ala2size</td>
<td>Input</td>
<td>Available only in SONET OC-12 and OC-48 modes to select between one of the following two word alignment options:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Logic Level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>rx_bitslip</td>
<td>Input</td>
<td>Asynchronous bit-slip control when the word aligner is configured in bit-slip mode. At every rising edge of this signal, the word aligner slips one bit into the received data stream, effectively shifting the word boundary by one bit. The minimum pulse-width is two recovered clock cycles.</td>
</tr>
<tr>
<td>rx_enapatternalign</td>
<td>Input</td>
<td>Asynchronous manual word alignment enable control. This signal is edge-sensitive with 8-bit width data and level sensitive with 10-bit width data. The minimum pulse-width is two recovered clock cycles.</td>
</tr>
<tr>
<td>rx_invpolarity</td>
<td>Input</td>
<td>Asynchronous receiver polarity inversion control. When asserted high, the polarity of every bit of the 8-bit or 10-bit input data word to the word aligner is inverted.</td>
</tr>
<tr>
<td>rx_revbitorderwa</td>
<td>Input</td>
<td>Asynchronous receiver bit reversal control. Available only in Basic mode with the word aligner configured in bit-slip mode. When asserted high in Basic mode, the 8-bit or 10-bit data ( D[7:0] ) or ( D[9:0] ) at the output of the word aligner is rewired to ( D[0:7] ) or ( D[0:9] ), respectively.</td>
</tr>
<tr>
<td>rx_bitslipboundaryselectout</td>
<td>Output</td>
<td>Asynchronous signal indicating the number of bits slipped in the word aligner when the word aligner is configured in manual mode.</td>
</tr>
<tr>
<td>rx_patterndetect</td>
<td>Output</td>
<td>Word alignment pattern detect indicator. A high level indicates that the word alignment pattern is found on the current word boundary. The width of this signal depends on the channel width shown below:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Channel Width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8/10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16/20</td>
</tr>
</tbody>
</table>
Table 1–25. ALTGX Megafunction Word Aligner Ports for Arria II Devices (Part 2 of 2)

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_rlv</td>
<td>Output</td>
<td>Asynchronous run-length violation indicator. A high pulse is driven when the number of consecutive 1s or 0s in the received data stream exceeds the programmed run length violation threshold. This signal is driven for a minimum of two recovered clock cycles in configurations without byte serializer and a minimum of three recovered clock cycles in configurations with byte serializer.</td>
</tr>
<tr>
<td>rx_syncstatus</td>
<td>Output</td>
<td>Word alignment synchronization status indicator. For word aligner in automatic synchronization state machine mode, this signal is driven high if the conditions required to remain in synchronization are met. For word aligner in manual alignment mode, this signal is driven high for one parallel clock cycle synchronous to the MSByte of the word alignment pattern. This signal is not available for word aligner in bit-slip mode. The width of this signal depends on the channel width shown below:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8/10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16/20</td>
</tr>
</tbody>
</table>

Table 1–26 lists the deskew FIFO port name and description for the ALTGX megafunction.

Table 1–26. ALTGX Megafunction Deskew FIFO Port for Arria II Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_channelaligned</td>
<td>Output</td>
<td>Indicates whether all the channels are aligned. This signal is only available in XAUI mode. A high level indicates that the XAUI deskew state machine is either in a ALIGN_ACQUIRED_1, ALIGN_ACQUIRED_2, ALIGN_ACQUIRED_3, or ALIGN_ACQUIRED_4 state, as specified in the PCS deskew state diagram in IEEE P802.3ae specification. A low level indicates that the XAUI deskew state machine is either in a LOSS_OF_ALIGNMENT, ALIGN_DETECT_1, ALIGN_DETECT_2, or ALIGN_DETECT_3 state, as specified in the PCS deskew state diagram in IEEE P802.3ae specification.</td>
</tr>
</tbody>
</table>

Table 1–27 lists the rate match (clock rate compensation) FIFO port names and descriptions for the ALTGX megafunction.

Table 1–27. ALTGX Megafunction Rate Match (Clock Rate Compensation) FIFO Ports for Arria II Devices (Part 1 of 2)

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_rmfifodatadeleted</td>
<td>Output</td>
<td>Rate match FIFO deletion status indicator. A high level indicates that the rate match pattern byte was deleted to compensate for the PPM difference in reference clock frequencies between the upstream transmitter and the local receiver.</td>
</tr>
<tr>
<td>rx_rmfifodatainserted</td>
<td>Output</td>
<td>Rate match FIFO insertion status indicator. A high level indicates that the rate match pattern byte was inserted to compensate for the PPM difference in reference clock frequencies between the upstream transmitter and the local receiver.</td>
</tr>
</tbody>
</table>
Table 1–28 lists the 8B/10B decoder port names and descriptions for the ALTGX megafuction. These ports are 1-bit wide with 8-bit channel width and 2-bit wide with 16-bit channel width.

Table 1–27. ALTGX Megafunction Rate Match (Clock Rate Compensation) FIFO Ports for Arria II Devices  (Part 2 of 2)

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_rmfifoempty</td>
<td>Output</td>
<td>Asynchronous rate match FIFO empty status indicator. A high level indicates that the rate match FIFO is empty. This signal is driven for a minimum of two recovered clock cycles in configurations without byte serializer and a minimum of three recovered clock cycles in configurations with byte serializer. You must then assert the rx_digitalreset signal to reset this signal.</td>
</tr>
<tr>
<td>rx_rmfifofull</td>
<td>Output</td>
<td>Asynchronous rate match FIFO full status indicator. A high level indicates that the rate match FIFO is full. This signal is driven for a minimum of two recovered clock cycles in configurations without byte serializer and a minimum of three recovered clock cycles in configurations with byte serializer. You must then assert the rx_digitalreset signal to reset this signal.</td>
</tr>
</tbody>
</table>

Table 1–28. ALTGX Megafunction 8B/10B Decoder Ports for Arria II Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_ctrldetect</td>
<td>Output</td>
<td>Receiver control code indicator. A high level indicates that the associated received code group is a control (Kx.y/) code group. A low level indicates that the associated received code group is a data (Dx.y/) code group.</td>
</tr>
<tr>
<td>rx_disperr</td>
<td>Output</td>
<td>8B/10B disparity error indicator port. A high level indicates that a disparity error was detected on the associated received code group.</td>
</tr>
<tr>
<td>rx_errdetect</td>
<td>Output</td>
<td>8B/10B code group violation or disparity error indicator. A high level indicates that a code group violation or disparity error was detected on the associated received code group. Use with the rx_disperr signal to differentiate between a code group violation and/or a disparity error as follows: [rx_errdetect: rx_disperr] 2'b00—no error 2'b10—code group violation 2'b11—disparity error or both</td>
</tr>
<tr>
<td>rx_runningdisp</td>
<td>Output</td>
<td>8B/10B running disparity indicator. A high level indicates that data on the rx_dataout port was received with a negative running disparity. A low level indicates that data on the rx_dataout port was received with a positive running disparity.</td>
</tr>
</tbody>
</table>
Table 1–29 lists the byte ordering block port names and descriptions for the ALTGX megafunction.

Table 1–29. ALTGX Megafunction Byte Ordering Block Ports for Arria II GX and GZ Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_enabyteord</td>
<td>Input</td>
<td>Asynchronous enable byte ordering control. The byte ordering block is rising-edge sensitive to this signal. A low-to-high transition triggers the byte ordering block to restart the byte ordering operation.</td>
</tr>
<tr>
<td>rx_byteorderalignstatus</td>
<td>Output</td>
<td>Byte ordering status indicator. A high level indicates that the byte ordering block has detected the programmed byte ordering pattern in the LSByte of the received data from the byte deserializer.</td>
</tr>
</tbody>
</table>

Table 1–30 lists the RX phase compensation FIFO port names and descriptions for the ALTGX megafunction.

Table 1–30. ALTGX Megafunction RX Phase Compensation FIFO Ports for Arria II Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coreclkout</td>
<td>Input</td>
<td>Clock from the CMU0 block of the associated transceiver block or the master transceiver block for ×4 bonded or ×8 bonded channel configurations, respectively. This is the default read and write clocks for those configurations.</td>
</tr>
<tr>
<td>rx_coreclk</td>
<td>Input</td>
<td>Optional read clock port for the RX phase compensation FIFO. If not enabled, the Quartus II software automatically selects rx_clkout/tx_clkout/coreclkout as the read clock for the RX phase compensation FIFO. If selected, you must drive this port with a clock that has 0 PPM difference with respect to the FIFO write clock.</td>
</tr>
<tr>
<td>rx_clkout</td>
<td>Input</td>
<td>Recovered clock from the receiver channel. This is the default read and write clocks for the RX phase compensation FIFO in non-bonded configurations without the rate-match FIFO.</td>
</tr>
<tr>
<td>tx_clkout</td>
<td>Input</td>
<td>Clock from the transmitter channel local clock divider. This is the default read and write clocks for the RX phase compensation FIFO in non-bonded configurations with the rate-match FIFO.</td>
</tr>
<tr>
<td>rx_dataout</td>
<td>Output</td>
<td>Parallel data output from the receiver to the FPGA fabric. The bus width depends on the channel width multiplied by the number of channels per instance.</td>
</tr>
<tr>
<td>rx_phase_comp_fifo_error</td>
<td>Output</td>
<td>RX phase compensation FIFO full or empty indicator. A high level indicates that the RX phase compensation FIFO is either full or empty.</td>
</tr>
</tbody>
</table>

Table 1–31 lists the receiver physical medium attachment (PMA) port names and descriptions for the ALTGX megafunction.

Table 1–31. ALTGX Megafunction Receiver PMA Ports for Arria II Devices (Part 1 of 2)

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_cruclk</td>
<td>Input</td>
<td>Input reference clock for the receiver CDR.</td>
</tr>
<tr>
<td>rx_datain</td>
<td>Input</td>
<td>Receiver serial data input port.</td>
</tr>
<tr>
<td>rx_locktodata</td>
<td>Input</td>
<td>Asynchronous receiver CDR LTD mode control signal. When asserted high, the receiver CDR is forced to LTD mode. When de-asserted low, the receiver CDR lock mode depends on the rx_locktorefclk signal level.</td>
</tr>
</tbody>
</table>
Table 1–31. ALTGX Megafunction Receiver PMA Ports for Arria II Devices (Part 2 of 2)

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_locktorefcclk</td>
<td>Input</td>
<td>Asynchronous receiver CDR LTR mode control signal. The <code>rx_locktorefcclk</code> and <code>rx_locktodata</code> signals control whether the receiver CDR is in LTR or LTD mode, as follows: 0/0—receiver CDR is in automatic mode 0/1—receiver CDR is in LTR mode 1/x—receiver CDR is in LTD mode</td>
</tr>
<tr>
<td>rx_seriallpbken</td>
<td>Input</td>
<td>Active-high serial loopback control port.</td>
</tr>
<tr>
<td>rx_seriallpbkin</td>
<td>Input</td>
<td>Input on a Receiver-only configuration when you enable serial loopback. You must connect this port to the <code>tx_seriallpbkout</code> port.</td>
</tr>
<tr>
<td>tx_reverseriallpbkin</td>
<td>Input</td>
<td>Input on a Transmitter-only configuration when you enable reverse serial loopback. You must connect this port to the <code>rx_reverseriallpbkout</code> port.</td>
</tr>
<tr>
<td>rx_freqlocked</td>
<td>Output</td>
<td>Asynchronous receiver CDR lock mode indicator. A high level indicates that the receiver CDR is in LTD mode. A low level indicates that the receiver CDR is in LTR mode.</td>
</tr>
<tr>
<td>rx_pll_locked</td>
<td>Output</td>
<td>Asynchronous active high receiver CDR LTR indicator. The receiver CDR is locked to the input reference clock.</td>
</tr>
<tr>
<td>tx_revseriallpbkout</td>
<td>Output</td>
<td>Output on a Receiver-only configuration when you enable reverse serial loopback. You must connect this port to the <code>tx_revseriallpbkout</code> port.</td>
</tr>
<tr>
<td>tx_seriallpbkout</td>
<td>Output</td>
<td>Output on a Transmitter-only configuration when you enable serial loopback. You must connect this port to the <code>tx_seriallpbkout</code> port.</td>
</tr>
<tr>
<td>rx_signaldetect</td>
<td>Output</td>
<td>Asynchronous signal threshold detect indicator for PCIe mode only. A high level indicates that the signal present at the receiver input buffer is above the programmed signal detection threshold value. If the electrical idle inference block is disabled in PCIe mode, the <code>rx_signaldetect</code> signal is inverted and driven on the <code>pipeelecidle</code> port.</td>
</tr>
</tbody>
</table>

Table 1–32 lists the TX phase compensation FIFO port names and descriptions for the ALTGX megafuction.

Table 1–32. ALTGX Megafunction TX Phase Compensation FIFO Ports for Arria II Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx_coreclk</td>
<td>Input</td>
<td>Optional write clock port for the TX phase compensation FIFO. If enabled, you must drive this port with a clock that is frequency locked to <code>tx_clkout/coreclkout</code> (with 0 PPM frequency difference).</td>
</tr>
<tr>
<td>tx_datain</td>
<td>Input</td>
<td>Parallel data input from the FPGA fabric to the transmitter. The bus width depends on the channel width multiplied by the number of channels per instance.</td>
</tr>
<tr>
<td>tx_clkout</td>
<td>Input</td>
<td>FPGA fabric-transceiver interface clock. Each channel has a <code>tx_clkout</code> signal in non-bonded channel configurations.</td>
</tr>
<tr>
<td>tx_phase_comp_fifo_error</td>
<td>Output</td>
<td>TX phase compensation FIFO full or empty indicator. A high level indicates that the TX phase compensation FIFO is either full or empty.</td>
</tr>
<tr>
<td>coreclkout</td>
<td>Input</td>
<td>Clock from the CMU0 block of the associated transceiver block or of the master transceiver block for ×4 bonded or ×8 bonded channel configurations, respectively. This is the default read and write clocks for those configurations.</td>
</tr>
</tbody>
</table>
Table 1–33 lists the 8B/10B encoder port names and descriptions for the ALTGX megafun.

### Table 1–33. ALTGX Megafunction 8B/10B Encoder Ports for Arria II Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx_bitslipboundaryselect</td>
<td>Input</td>
<td>Indicates the number of bits to slip at the transmitter for word alignment at the receiver.</td>
</tr>
<tr>
<td>tx_ctrlenable</td>
<td>Input</td>
<td>8B/10B encoder /Kx.y/ or /Dx.y/ control. When asserted high, the 8B/10B encoder encodes the data on the tx_datain port as a /Kx.y/ control code group. When de-asserted low, it encodes the data on the tx_datain port as a /Dx.y/ data code group. The width of this signal depends on the channel width shown below:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Channel Width</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tx_dispval</td>
<td>Input</td>
<td>8B/10B encoder force disparity value. A high level on the tx_dispval signal when the tx_forcedisp signal is asserted high forces the 8B/10B encoder to encode the data on the tx_datain port with a negative starting running disparity. A low level on the tx_dispval signal when the tx_forcedisp signal is asserted high forces the 8B/10B encoder to encode the data on the tx_datain port with a positive starting running disparity. The width of this signal depends on the channel width shown below:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Channel Width</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tx_forcedisp</td>
<td>Input</td>
<td>8B/10B encoder force disparity control. When asserted high, it forces the 8B/10B encoder to encode the data on the tx_datain port with a positive or negative disparity, depending on the tx_dispval signal level. When de-asserted low, the 8B/10B encoder encodes the data on the tx_datain port according to the 8B/10B running disparity rules. The width of this signal depends on the channel width shown below:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Channel Width</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tx_invpolarity</td>
<td>Input</td>
<td>Asynchronous transmitter polarity inversion control. Useful feature for correcting situations where the positive and negative signals of the differential serial link are accidentally swapped during board layout. When asserted high the polarity of every bit of the 8-bit or 10-bit input data to the serializer is inverted.</td>
</tr>
</tbody>
</table>

Table 1–34 lists the transmitter PMA port names and descriptions for the ALTGX megafun.

### Table 1–34. ALTGX Megafunction Transmitter PMA Ports for Arria II Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixedclk</td>
<td>Input</td>
<td>125-MHz clock for receiver detect and offset cancellation in PCIe mode.</td>
</tr>
<tr>
<td>tx_dataout</td>
<td>Output</td>
<td>Transmitter serial data output port.</td>
</tr>
</tbody>
</table>
Table 1–35 lists the reconfiguration block port names and descriptions for the ALTGX megafunction.

For more information about these ports, refer to AN 558: Implementing Dynamic Reconfiguration in Arria II Devices.

### Table 1–35. ALTGX Megafunction Reconfiguration Block Ports for Arria II Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reconfig_clk</td>
<td>Input</td>
<td>Dynamic reconfiguration clock. This clock is also used for offset cancellation in all modes except PCIe mode.</td>
</tr>
<tr>
<td>reconfig_fromgxb</td>
<td>Input</td>
<td>The width of this signal is determined by the value you set in the What is the number of channels controlled by the reconfig controller? option in the Reconfiguration settings screen.</td>
</tr>
<tr>
<td>reconfig_togxb[3:0]</td>
<td>Output</td>
<td>The width of this signal is fixed to four bits. It is independent of the value you set in the What is the number of channels controlled by the reconfig controller? option in the Reconfiguration settings screen.</td>
</tr>
</tbody>
</table>

Table 1–36 lists the PIPE interface port names and descriptions for the ALTGX megafunction.

### Table 1–36. ALTGX Megafunction PIPE Interface Ports for Arria II Devices (Available only in PCIe functional mode) (Part 1 of 2)

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipe8b10binvpolarity</td>
<td>Input</td>
<td>PCIe polarity inversion control. Functionally equivalent to the RxPolarity signal defined in PIPE specification revision 2.00. Available only in PCIe mode. When asserted high, the polarity of every bit of the 10-bit input data to the 8B/10B decoder is inverted.</td>
</tr>
<tr>
<td>powerdn</td>
<td>Input</td>
<td>PCIe power state control. Functionally equivalent to the PowerDown[1:0] signal defined in PIPE specification revision 2.00. The width of this signal is 2 bits and is encoded as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ 2'b00: P0—Normal Operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ 2'b01: P0s—Low Recovery Time Latency, Low Power State</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ 2'b10: P1—Longer Recovery Time Latency, Lower Power State</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ 2'b11: P2—Lowest Power State</td>
</tr>
<tr>
<td>tx_detectrxloop</td>
<td>Input</td>
<td>Receiver detect or PCIe loopback control. Functionally equivalent to the TxDetectRx/Loopback signal defined in PIPE specification revision 2.00. When asserted high in the P1 power state with the tx_forceelecidle signal asserted, the transmitter buffer begins the receiver detection operation. When the receiver detect completion is indicated on the pipephydonestatus port, this signal must be de-asserted. When asserted high in the P0 power state with the tx_forceelecidle signal de-asserted, the transceiver datapath is dynamically configured to support parallel loopback, as described in “PCIe (Reverse Parallel Loopback)” on page 1–88.</td>
</tr>
<tr>
<td>tx_forcedispcompliance</td>
<td>Input</td>
<td>Forces the 8B/10B encoder to encode with a negative running disparity. Functionally equivalent to the TxCompliance signal defined in PIPE specification revision 2.00. Must be asserted high only when transmitting the first byte of the PCI Express Compliance Pattern to force the 8B/10B encode with a negative running disparity, as required by the PCIe protocol.</td>
</tr>
</tbody>
</table>
Table 1–36. ALTGX Megafunction PIPE Interface Ports for Arria II Devices (Available only in PCIe functional mode) (Part 2 of 2)

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx_forceelecidle</td>
<td>Input</td>
<td>Force transmitter buffer to PCIe electrical idle signal levels. Functionally equivalent to the TxElecIdle signal defined in PIPE specification revision 2.00.</td>
</tr>
<tr>
<td>pipeelecidle</td>
<td>Output</td>
<td>Asynchronous signal to indicate whether electrical idle is detected or inferred at the receiver. Functionally equivalent to the RxElecIdle signal defined in PIPE specification revision 2.00. If you enable the electrical idle inference block, it drives this signal high when it infers an electrical idle condition, as described in “Electrical Idle Inference” on page 1–70. Otherwise, it drives this signal low. If the electrical idle inference block is disabled, the rx_signaldetect signal from the signal detect circuitry in the receiver input buffer is inverted and driven on this port.</td>
</tr>
<tr>
<td>pipephydonestatus</td>
<td>Output</td>
<td>PHY function completion indicator. Functionally equivalent to the PhyStatus signal defined in PIPE specification revision 2.00. Asserted high for one parallel clock cycle to communicate completion of several PHY functions, such as power state transition and receiver detection.</td>
</tr>
</tbody>
</table>
| pipestatus      | Output       | PCIe receiver status port. Functionally equivalent to the RxStatus[2:0] signal defined in PIPE specification revision 2.00. The width of this signal is 3 bits per channel. The encoding of receiver status on the pipestatus port is as follows:  
- 000—Received data OK  
- 001—1 skip added  
- 010—1 skip removed  
- 011—Receiver detected  
- 100—8B/10B decoder error  
- 101—Elastic buffer overflow  
- 110—Elastic buffer underflow  
- 111—Received disparity error |
| rx_pipedatavalid| Output       | Valid data and control on the rx_dataout and rx_ctrldetect ports indicator. Functionally equivalent to the RxValid signal defined in PIPE specification revision 2.00. |

Table 1–37 lists the reset and power down port names and descriptions for the ALTGX megafuction.

For more information, refer to the Reset Control and Power Down in Arria II Devices chapter.

Table 1–37. ALTGX Megafunction Reset and Power Down Ports for Arria II Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gxb_powerdown</td>
<td>Input</td>
<td>Asynchronous transceiver block power down signal. When asserted high, all digital and analog circuitry in the PCS, PMA, CMU of the transceiver block is powered down, except for the refclk buffers.</td>
</tr>
<tr>
<td>rx_analogreset</td>
<td>Input</td>
<td>Active-high receiver PMA reset.</td>
</tr>
</tbody>
</table>
Table 1–37. ALTGX Megafunction Reset and Power Down Ports for Arria II Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx.digitalreset</td>
<td>Input</td>
<td>Active-high receiver PCS reset.</td>
</tr>
<tr>
<td>tx.digitalreset</td>
<td>Input</td>
<td>Active-high transmitter PCS reset.</td>
</tr>
</tbody>
</table>

Table 1–38 lists the calibration block port names and descriptions for the ALTGX megafunction.

Table 1–38. ALTGX Megafunction Calibration Block Ports for Arria II Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cal_blk_clk</td>
<td>Input</td>
<td>Clock for transceiver calibration blocks.</td>
</tr>
<tr>
<td>cal_blk_powerdown</td>
<td>Input</td>
<td>Calibration block power down control.</td>
</tr>
</tbody>
</table>

Table 1–39 lists the verifier port names and descriptions for the ALTGX megafunction.

Table 1–39. ALTGX Megafunction Verifier Ports for Arria II Devices

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_bisterr</td>
<td>output</td>
<td>For BIST mode, the rx_bisterr signal asserts and stays high when the verifier detects an error. For PRBS mode, the rx_bisterr signal asserts and stays high for a minimum of three rx_clkout clock cycles when the verifier detects an error and de-asserts if the following PRBS sequence has no error.</td>
</tr>
<tr>
<td>rx_bistdone</td>
<td>output</td>
<td>For BIST mode, the rx_bistdone port asserts and stays high when the verifier either receives one full cycle of incremental pattern or detects an error in the receiver data. For PRBS mode, the rx_bistdone port asserts high and stays high when the verifier a full cycle of PRBS pattern.</td>
</tr>
</tbody>
</table>

Document Revision History

Table 1–40 lists the revision history for this chapter.

Table 1–40. Document Revision History (Part 1 of 2)

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 2015</td>
<td>4.6</td>
<td>■ Added a note to the “Rate-Match FIFO” section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated the “Rate Match FIFO in GbE Mode” section.</td>
</tr>
<tr>
<td>September 2014</td>
<td>4.5</td>
<td>■ Global: changed tx_detectrxloopback to tx_detectrxloop.</td>
</tr>
<tr>
<td>October 2013</td>
<td>4.4</td>
<td>■ Updated the “Dynamic Reconfiguration” section.</td>
</tr>
<tr>
<td>July 2012</td>
<td>4.3</td>
<td>■ Removed Fiber Channel in Table 1–2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Finalized information in Table 1–4, Table 1–14, and Table 1–15.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Removed OC-3 (155 Mbps) from Table 1–1 and Table 1–2.</td>
</tr>
<tr>
<td>December 2011</td>
<td>4.2</td>
<td>■ Updated Table 1–1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added Table 1–15.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated Figure 1–55.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Minor text edits.</td>
</tr>
<tr>
<td>Date</td>
<td>Version</td>
<td>Changes</td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>June 2011</td>
<td>4.1</td>
<td>Added Table 1–38.&lt;br&gt;Updated Figure 1–1, Figure 1–2, Figure 1–3, Figure 1–24, Figure 1–35, Figure 1–36, Figure 1–49, Figure 1–53, Figure 1–54, Figure 1–87 and Figure 1–88.&lt;br&gt;Updated Table 1–2, Table 1–6, and Table 1–7.&lt;br&gt;Updated the “Transmitter Output Buffer”, “Programmable Differential OCT”, “Dynamic Reconfiguration”, “PCIe Hard IP Block”, and “GIGE” sections.&lt;br&gt;Minor text edits.</td>
</tr>
<tr>
<td>December 2010</td>
<td>4.0</td>
<td>Updated to add Arria II GZ information.&lt;br&gt;Updated Figure 1–7.&lt;br&gt;Updated Table 1–20.&lt;br&gt;Updated the “Programmable Equalization, DC Gain, and Offset Cancellation” section.&lt;br&gt;Minor text edits.</td>
</tr>
<tr>
<td>July 2010</td>
<td>3.0</td>
<td>Updated Figure 1–1, Figure 1–4, Figure 1–45, Figure 1–74, Figure 1–76, Figure 1–48, Figure 1–49, and Figure 1–50.&lt;br&gt;Updated the “Transceiver Block Overview”, “Programmable Equalization, DC gain, and Offset Cancellation”, “TX Phase Compensation FIFO”, “Transmitter Output Buffer”, “Functional Modes”, “PCIe Mode” “Reverse Serial Loopback”, “Reverse Serial Pre-CDR Loopback”, and “Dynamic Reconfiguration” sections.&lt;br&gt;Moved Table 1–17 to the Arria II Device Family Datasheet.&lt;br&gt;Converted protocol information to Table 1–1.&lt;br&gt;Minor text edits. For example, change “PCI Express (PIPE)” to “PCIe” and “8B10B” to “8B/10B”.</td>
</tr>
<tr>
<td>November 2009</td>
<td>2.1</td>
<td>Updated figures.&lt;br&gt;Updated Base Specification references to 2.0.&lt;br&gt;Removed table 1-4 from 2.0 version and referenced Arria II GX Device Data Sheet</td>
</tr>
<tr>
<td>June 2009</td>
<td>2.0</td>
<td>Reorganized.&lt;br&gt;Added “Deterministic Latency” on page 1–44 and “Built-In Self Test (BIST) and Pseudo Random Binary Sequence (PRBS)” on page 1–77.&lt;br&gt;Updated all figures.&lt;br&gt;Port list tables were updated.</td>
</tr>
<tr>
<td>March 2009</td>
<td>1.1</td>
<td>Updated:&lt;br&gt;Dynamic Reconfiguration Controller Architecture&lt;br&gt;All AN 558: Implementing dynamic Reconfiguration in Arria II GX Devices&lt;br&gt;Transceiver Channel Reconfiguration&lt;br&gt;Table 1.2 and Table 1.4&lt;br&gt;Figure 1.69 and Figure 1.70&lt;br&gt;Added:&lt;br&gt;Offset Cancellation in the Receiver Buffer and Receiver CDR&lt;br&gt;Basic Double-Width Mode Configurations</td>
</tr>
<tr>
<td>February 2009</td>
<td>1.0</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>
2. Transceiver Clocking in Arria II Devices

This chapter describes the Arria® II GX and GZ transceiver clocking architecture, including the input reference clocking, transceiver channel datapath clocking, FPGA fabric-transceiver interface clocking, and FPGA fabric phase-locked loop (PLL)-transceiver PLL cascading.

This chapter includes the following sections:

- “CMU PLL and Receiver CDR Input Reference Clocking”
- “Transceiver Channel Datapath Clocking” on page 2–6
- “FPGA Fabric-Transceiver Interface Clocking” on page 2–28
- “FPGA Fabric PLL-Transceiver PLL Cascading” on page 2–56
- “Using the CMU PLL for Clocking User Logic in the FPGA Fabric” on page 2–66

CMU PLL and Receiver CDR Input Reference Clocking

Each transceiver block in the Arria II GX and GZ device contains the following:

- Two clock multiplier unit (CMU) PLLs (CMU0 PLL and CMU1 PLL)
- Four clock data recovery (CDR) units, one in each receiver channel

The CMU PLLs and receiver CDRs require an input reference clock to operate. The CMU PLL synthesizes the input reference clock to generate the high-speed serial clock used in the transmitter physical media attachment (PMA). The receiver CDR uses the input reference clock as a training clock when it is in lock-to-reference (LTR) mode.

The CMU PLLs and receiver CDRs in each transceiver block can derive input reference from one of the following sources:

- refclk0 and refclk1 pins of the same transceiver block
- refclk0 and refclk1 pins of other transceiver blocks on the same side of the device using the inter-transceiver block (ITB) clock network
- Dedicated CLK input pins on the FPGA global clock network
- Clock output pins from the left side and right side PLLs in the FPGA fabric
Figure 2–1 shows the input reference clock sources for CMU PLLs and receiver CDRs within a transceiver block.

**Figure 2–1. Input Reference Clock Sources in a Transceiver Block**

Note to Figure 2–1:
(1) One global clock line is available for each CMU PLL and receiver CDR in a transceiver block. This configuration allows each CMU PLL and receiver CDR to derive its input reference clock from a separate FPGA CLK input pin.
Figure 2–2 shows the input reference clock sources for CMU PLLs and receiver CDRs in four transceiver blocks on the left side of the EP2AGX260FF35 device.

**Figure 2–2. Input Reference Clock Sources Across Transceiver Blocks**

**refclk0 and refclk1 Pins**

Each transceiver block has two dedicated refclk pins that you can use to drive the CMU PLL, receiver CDR, input reference clock, or all three. Each of the two CMU PLLs and four receiver CDRs within a transceiver block can derive its input reference clock from either the refclk0 or refclk1 pin.

The refclk pins provide the cleanest input reference clock path to the CMU PLLs. Altera recommends using the refclk pins to drive the CMU PLL input reference clock for improved transmitter output jitter performance.
Table 2–1 lists the electrical specifications for the input reference clock signal driven on the refclk pins.

### Table 2–1. Electrical Specifications for the Input Reference Clock for Arria II Devices

<table>
<thead>
<tr>
<th>Protocol</th>
<th>I/O Standard</th>
<th>Coupling</th>
<th>Termination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gigabit Ethernet (GbE)</td>
<td>1.2-V PCML, 1.5-V PCML, 2.5-V PCML, Differential LVPECL, LVDS</td>
<td>AC</td>
<td>On-chip</td>
</tr>
<tr>
<td>XAUI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serial RapidIO® (SRI0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SONET/SDH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCI Express (PIPE) (1), (2)</td>
<td>1.2-V PCML, 1.5-V PCML, 2.5-V PCML, Differential LVPECL, LVDS</td>
<td>AC</td>
<td>On-chip</td>
</tr>
<tr>
<td></td>
<td>HCSL</td>
<td>DC</td>
<td>Off-chip</td>
</tr>
</tbody>
</table>

**Notes to Table 2–1:**

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

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

**Figure 2–3. Termination Scheme for a Reference Clock Signal When Configured as HCSL (Note 1)**

![Termination Scheme](image)

**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 PCIe specification.
2. Select resistor values as recommended by the PCIe clock source vendor.
Inter-Transceiver Block Clock Lines

The ITB clock lines provide an input reference clock path from the refclk pins of one transceiver block to the CMU PLLs and receiver CDRs of other transceiver blocks. In designs that have channels located in different transceiver blocks, the ITB clock lines eliminate the need to connect the on-board reference clock crystal oscillator to the refclk pin of each transceiver block. The ITB clock lines also drive the clock signal on the refclk pins to the clock logic in the FPGA fabric.

Each refclk pin drives one ITB clock line for a total of up to eight ITB clock lines on the left side of the device, as shown in Figure 2–4.

Figure 2–4. Inter-Transceiver Block Clock Lines (Note 1)

Transceiver Block GXBL3

Two CMU PLLs and Four RX CDRs

Global Clock Line
PLL Cascade Clock

Transceiver Block GXBL2

Two CMU PLLs and Four RX CDRs

Global Clock Line
PLL Cascade Clock

Transceiver Block GXBL1

Two CMU PLLs and Four RX CDRs

Global Clock Line
PLL Cascade Clock

Transceiver Block GXBL0

Two CMU PLLs and Four RX CDRs

Global Clock Line
PLL Cascade Clock

Note to Figure 2–4:

(1) This figure shows the ITB clock lines on the left side of the EP2AGX60FF35 device. The number of ITB clock lines available in any Arria II GX or GZ device is equal to the number of refclk pins available in that device.
**Dedicated CLK Input Pins on the FPGA Global Clock Network**

Arria II GX and GZ devices provide six differential CLK[5:0] input pins located in non-transceiver I/O banks that you can use to provide the input reference clock to the transceiver blocks. The Quartus II software automatically chooses the global clock network to route the input reference clock signal from the CLK pins to the transceiver blocks.

For more information, refer to the “Dedicated Clock Input Pins” section in the Clock Networks and PLLs in Arria II Devices chapter.

One global clock resource is available for each CMU PLL and receiver CDR within a transceiver block. This configuration allows each CMU PLL and receiver CDR to derive its input reference clock from a separate FPGA CLK input pin.

**Clock Output from Left and Right PLLs in the FPGA Fabric**

You can use the synthesized clock output from one of the left or right PLLs to provide the input reference clock to the CMU PLLs and receiver CDRs. Arria II GX devices provide a dedicated clock path from the left PLLs (PLL_L1, PLL_L2, PLL_L3, and PLL_L4) in the FPGA fabric to the PLL cascade network located on the left side of the device.

Arria II GZ devices also provide a dedicated clock path from the right PLLs (PLL_R1, PLL_R2, PLL_R3, and PLL_R4) in the FPGA fabric to the PLL cascade network located on the right side of the device. The additional clock multiplication factors available in the left and right PLLs allow more options for on-board crystal oscillator frequencies. For more information, refer to “FPGA Fabric PLL-Transceiver PLL Cascading” on page 2–56.

**Transceiver Channel Datapath Clocking**

The following sections describe transmitter and receiver channel datapath clocking in various configurations. Datapath clocking varies with physical coding sublayer (PCS) configurations in different functional modes and channel bonding options.

**Transmitter Channel Datapath Clocking**

This section describes transmitter channel PMA and PCS datapath clocking in non-bonded and bonded channel configurations. Transmitter datapath clocking in bonded channel configurations provide low channel-to-channel skew when compared with non-bonded channel configurations.

The following factors contribute to transmitter channel-to-channel skew:

- High-speed serial clock and low-speed parallel clock skew between channels
- Unequal latency in the transmitter phase compensation FIFO
In non-bonded channel configurations, the high-speed serial clock and low-speed parallel clock in each channel are generated independently by its local clock divider, as shown in Figure 2–5 on page 2–8, resulting in higher channel-to-channel clock skew. The transmitter phase compensation FIFO in each non-bonded channel has its own pointers and control logic that can result in unequal latency in the transmitter phase compensation FIFO of each channel. The higher transceiver clock skew and unequal latency in the transmitter phase compensation FIFO in each channel can result in higher channel-to-channel skew in non-bonded channel configurations.

In bonded channel configurations, the high-speed serial clock and low-speed parallel clock for all bonded channels are generated by the same CMU0 clock divider block (refer to Figure 2–6 on page 2–11), resulting in lower channel-to-channel clock skew. The transmitter phase compensation FIFO in all bonded channels share common pointers and control logic generated in the CMU0 channel, resulting in equal latency in the transmitter phase compensation FIFO of all bonded channels. The lower transceiver clock skew and equal latency in the transmitter phase compensation FIFOs in all channels provide lower channel-to-channel skew in bonded channel configurations.

**Non-Bonded Channel Configurations**

The following functional modes support non-bonded transmitter channel configuration:

- PCIe x1—Gen1 and Gen2 (Gen2 for Arria II GZ only)
- GbE
- SRIO
- SONET/SDH
- SDI
- Common Public Radio Interface (CPRI)/OBSAI
- Basic (except Basic x4 mode)
Figure 2–5 shows the transmitter channel datapath clocking in a non-bonded configuration.

Figure 2–5. Transmitter Datapath Clocking in a Non-Bonded Configuration
In non-bonded channel configurations, each channel can derive its clock independently from either CMU0 PLL or CMU1 PLL within the same transceiver block. The CMU PLL synthesizes the input reference clock to generate a clock that runs at a frequency of half the configured data rate. This half-rate clock from the CMU PLL is fed to the local clock divider block in each channel. Depending on the configured functional mode, the local clock divider block in each channel generates the low-speed parallel clock and high-speed serial clock. The serializer in the transmitter channel PMA uses both the low-speed parallel clock and high-speed serial clock for its parallel-in, serial-out operation. The low-speed parallel clock clocks both the 8B/10B encoder (if enabled) and the read port of the byte serializer (if enabled) in the transmitter channel PCS.

If the configured functional mode does not use the byte serializer, the low-speed parallel clock provides a clock to the read port of the transmitter phase compensation FIFO. The low-speed parallel clock is also driven directly on the tx_clkout port as the FPGA fabric-transceiver interface clock. You can use the tx_clkout port to clock transmitter data and control logic in the FPGA fabric.

If the configured functional mode uses a byte serializer to reduce the FPGA fabric-transceiver interface speed, the low-speed parallel clock is divided by two. This divide-by-two version of the low-speed parallel clock provides a clock to the write port of the byte serializer and the read port of the transmitter phase compensation FIFO. It is also driven on the tx_clkout port as the FPGA fabric-transceiver interface clock. You can use tx_clkout to clock transmitter data and control logic in the FPGA fabric.

Table 2–2 lists the transmitter channel datapath clock frequencies in non-bonded functional modes that have a fixed data rate.

### Table 2–2. Transmitter Channel Datapath Clock Frequencies in Non-Bonded Functional Modes for Arria II Devices

<table>
<thead>
<tr>
<th>Functional Mode (2)</th>
<th>Data Rate</th>
<th>High-Speed Serial Clock Frequency</th>
<th>Low-Speed Parallel Clock Frequency (MHz)</th>
<th>FPGA Fabric-Transceiver Interface Clock Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Without Byte Serializer (MHz) With Byte Serializer (MHz)</td>
</tr>
<tr>
<td>PCIe x1 (Gen 1)</td>
<td>2.5 Gbps</td>
<td>1.25 GHz</td>
<td>250</td>
<td>250 (1)</td>
</tr>
<tr>
<td>PCIe x1 (Gen 2)</td>
<td>5 Gbps</td>
<td>2.5 GHz</td>
<td>500</td>
<td>—</td>
</tr>
<tr>
<td>GbE</td>
<td>1.25 Gbps</td>
<td>625 MHz</td>
<td>125</td>
<td>—</td>
</tr>
<tr>
<td>SRIO</td>
<td>1.25 Gbps</td>
<td>625 MHz</td>
<td>125</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>2.5 Gbps</td>
<td>1.25 GHz</td>
<td>250</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>3.125 Gbps</td>
<td>1.5625 GHz</td>
<td>312.5</td>
<td>—</td>
</tr>
<tr>
<td>SONET/SDH OC12</td>
<td>622 Mbps</td>
<td>311 MHz</td>
<td>77.75</td>
<td>77.75</td>
</tr>
<tr>
<td>SONET/SDH OC48</td>
<td>2.488 Gbps</td>
<td>1.244 GHz</td>
<td>311</td>
<td>—</td>
</tr>
<tr>
<td>HD-SDI</td>
<td>1.485 Gbps</td>
<td>742.5 MHz</td>
<td>148.5</td>
<td>148.5</td>
</tr>
<tr>
<td></td>
<td>1.4835 Gbps</td>
<td>741.75 MHz</td>
<td>148.35</td>
<td>148.35</td>
</tr>
<tr>
<td>3G-SDI</td>
<td>2.97 Gbps</td>
<td>1.485 GHz</td>
<td>297</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>2.967 Gbps</td>
<td>1.4835 GHz</td>
<td>296.7</td>
<td>—</td>
</tr>
</tbody>
</table>

**Notes to Table 2–2:**

1. 250 MHz when you enable the PCIe hard IP.
2. Altera also supports CPRI and OBSAI. For more information, refer to AN 610: Implementing CPRI and OBSAI Protocols in Altera Devices.
Bonded Channel Configurations

Arria II GX and GZ devices support x4 PCS and PMA channel bonding that allows bonding of four channels within the same transceiver block. These devices also support x8 channel bonding in PCIe and Basic modes that allows bonding of eight PCS and PMA channels across two transceiver blocks on the same side of the device. Arria II GX and GZ devices with at least two transceiver blocks support x8 bonding.

Bonding is not supported on the receive side for Basic x4 and Basic x8 functional modes. If you use rate matcher, the clocking scheme for Basic x4 and Basic x8 functional modes, the clocking is similar to PCIe x4 mode, as shown in Figure 2–6 on page 2–11 and PCIe x8 mode, as shown in Figure 2–7 on page 2–14.

×4 Bonded Channel Configurations

The following functional modes support x4 bonded transmitter channel configuration:

- PCIe x4—Gen1 and Gen2 (Gen2 for Arria II GZ only)
- XAUI
- Basic x4

In x4 bonded channel configurations, the receiver datapath clocking varies depending on whether the configured functional mode uses the deskew FIFO or not.

Figure 2–6 shows the transmitter channel datapath clocking in x4 channel bonding configurations.

The Quartus II Compiler generates an error if you do not make the following assignments:

- tx_dataout[0] of the x4 bonded link (XAUI or PCIe x4) to physical channel 0 of the transceiver block
- tx_dataout[1] to physical channel 1 of the transceiver block
- tx_dataout[2] to physical channel 2 of the transceiver block
- tx_dataout[3] to physical channel 3 of the transceiver block
Figure 2–6. Transmitter Datapath Clocking in x4 Bonded Configurations

Note to Figure 2–6:
(1) In Arria II GX and GZ devices, there is only one dedicated PCIe hard IP, which supports PCIe Gen 1 x1, x4, and x8; and PCIe Gen 2 x1 and x4.
In x4 bonded channel configurations, CMU0_PLL or CMU1_PLL synthesizes the input reference clock to generate a clock that runs at a frequency of half the configured data rate. The half-rate clock from either of the CMU PLLs is fed to the CMU0 clock in the CMU0_Channel. Depending on the configured functional mode, the CMU0 clock divider block generates the high-speed serial clock and low-speed parallel clock. The serializer in the transmitter channel PMA of the four bonded channels uses the same low-speed parallel clock and high-speed serial clock from the CMU0 block for their parallel-in, serial-out operation. The low-speed parallel clock provides a clock to the 8B/10B encoder and the read port of the byte serializer (if enabled) in the transmitter channel PCS.

If the configured functional mode does not use the byte serializer, the low-speed parallel clock from the CMU0 clock divider block clocks the read port of the transmitter phase compensation FIFO in all four bonded channels. This low-speed parallel clock is also driven directly on the coreclkout port as the FPGA fabric-transceiver interface clock. You can use the coreclkout signal to clock transmitter data and control logic in the FPGA fabric for all four bonded channels.

If the configured functional mode uses the byte serializer, the low-speed parallel clock from the CMU0 clock divider is divided by two. This divide-by-two version of the low-speed parallel clock provides a clock to the write port of the byte serializer and the read port of the transmitter phase compensation FIFO in all four bonded channels. It is also driven on the coreclkout port as the FPGA fabric-transceiver interface clock. You can use the coreclkout signal to clock transmitter data and control logic in the FPGA fabric for all four bonded channels.

In x4 bonded channel configurations, the transmitter phase compensation FIFOs in all four bonded channels share common read and write pointers and enable signals generated in the CMU0 block channel of the transceiver block, ensuring equal transmitter phase compensation FIFO latency across all four bonded channels, resulting in low transmitter channel-to-channel skew.

Table 2–3 lists the transmitter datapath clock frequencies in x4 bonded functional modes that have a fixed data rate.

<table>
<thead>
<tr>
<th>Functional Mode (1)</th>
<th>Data Rate (Gbps)</th>
<th>High-Speed Serial Clock Frequency (GHz)</th>
<th>Low-Speed Parallel Clock Frequency (MHz)</th>
<th>FPGA Fabric-Transceiver Interface Clock Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Without Byte Serializer</td>
</tr>
<tr>
<td>PCIe x4 (Gen 1)</td>
<td>2.5</td>
<td>1.25</td>
<td>250</td>
<td>—</td>
</tr>
<tr>
<td>PCIe x4 (Gen 2)</td>
<td>5</td>
<td>2.5</td>
<td>500</td>
<td>—</td>
</tr>
<tr>
<td>XAUI</td>
<td>3.125</td>
<td>1.5625</td>
<td>312.5</td>
<td>—</td>
</tr>
</tbody>
</table>

Notes to Table 2–3:
(1) Altera also supports CPRI x4 and OBSAI x4. For more information, refer to AN 610: Implementing CPRI and OBSAI Protocols in Altera Devices.
(2) 250 MHz when you enable the PCIe hard IP.
x8 Bonded Channel Configuration

The PCIe x8 and Basic x8 functional modes support x8 bonded channel configuration in Arria II GX and GZ devices with two transceiver blocks. The eight bonded channels are located in two transceiver blocks, referred to as the master transceiver block and slave transceiver block, with four channels each. The \texttt{CMU0} clock divider in the \texttt{CMU0} block of the master transceiver block provides the serial PMA clock and parallel PCS clock to all eight bonded channels. The serializer in the transmitter channel PMA of the eight bonded channels uses the same low-speed parallel clock and high-speed serial clock from the \texttt{CMU0} of the master transceiver block for their parallel-in, serial-out operation. The low-speed parallel clock from the \texttt{CMU0} of the master transceiver block clocks the 8B/10B encoder and read port of the byte serializer (if enabled) in the transmitter channel PCS of all eight channels.

For an 8-bit FPGA fabric-transceiver channel interface that does not use the byte serializer, the low-speed parallel clock from the \texttt{CMU0} clock divider block in the master transceiver block clocks the read port of the transmitter phase compensation FIFO in all eight bonded channels. This low-speed parallel clock is also driven directly on the \texttt{coreclkout} port as the FPGA fabric-transceiver interface clock. You can use the \texttt{coreclkout} signal to clock the transmitter data and control logic in the FPGA fabric for all eight bonded channels.

For a 16-bit FPGA fabric-transceiver channel interface that uses the byte serializer, the low-speed parallel clock from the \texttt{CMU0} clock divider block in the master transceiver block is divided by two. This divide-by-two version of the low-speed parallel clock provides a clock to the write port of the byte serializer and the read port of the transmitter phase compensation FIFO in all eight bonded channels. It is also driven on the \texttt{coreclkout} port as the FPGA fabric-transceiver interface clock. You can use the \texttt{coreclkout} signal to clock the transmitter data and control logic in the FPGA fabric for all eight bonded channels.

In the x8 bonded channel configuration, the transmitter phase compensation FIFOs in all eight bonded channels share common read and write pointers and enable signals generated in the \texttt{CMU0} block of the master transceiver block, ensuring equal transmitter phase compensation FIFO latency across all eight bonded channels, resulting in low transmitter channel-to-channel skew.
Figure 2–7 shows transmitter datapath clocking in PCIe x8 channel bonding configurations.

**Figure 2–7. Transmitter Datapath Clocking in a x8 Bonded Configuration**

Note to Figure 2–7:
(1) In Arria II GX and GZ devices, there is only one dedicated PCIe hard IP, which supports PCIe Gen 1 x1, x4, and x8; and PCIe Gen 2 (Arria II GZ only) x1 and x4.

Figure 2–8 through Figure 2–10 show allowed master and slave transceiver block locations and PCIe logical lane-to-physical transceiver channel mapping in all Arria II GX and GZ devices.

The Quartus II Compiler generates an error if you do not map the PCIe logical lanes to the physical transceiver channels, as shown in Figure 2–8 through Figure 2–10 on page 2–16.
Figure 2–8 shows the PCIe x8 link in two transceiver block Arria II GX devices.

![Figure 2–8. One PCIe x8 Link in Two Transceiver Block Arria II GX Devices](image)

Figure 2–9 shows the PCIe x8 link in three transceiver block devices for Arria II GX devices.

![Figure 2–9. One PCIe x8 Link in Three Transceiver Block Arria II GX Devices](image)

**Note to Figure 2–9:**

1. Arria II GX and GZ devices with three transceiver blocks allow a maximum of one PCIe x8 link occupying two transceiver blocks. You can configure the other transceiver block to implement other functional modes.
Figure 2–10 shows the PCIe x8 link in four transceiver block Arria II GX devices.

**Figure 2–10. Two PCIe x8 Link in Four Transceiver Block Arria II GX Devices**

![Diagram showing two PCIe x8 links in four transceiver blocks]

**Note to Figure 2–10:**
(1) The second x8 link does not have PCIe hard IP support. Use soft IP support for the second x8 link.
Figure 2–11 shows two PCIe x8 links in four transceiver block Arria II GZ devices.

Figure 2–11. One PCIe x8 Link in Two Transceiver Block Devices and Two PCIe x8 Links in Four Transceiver Block Arria II GZ Devices

---


Two PCIe x8 Link in Four Transceiver Block Devices

Note to Figure 2–11:

(1) The PCIe hard IP block is only available on the left side of the Arria II device.

---

Figure 2–12 shows two PCIe x8 links in six transceiver block Arria II GZ devices.

Figure 2–12. Two PCIe x8 Links in Six Transceiver Block Arria II GZ Devices (Note 1)

---

EP2AGZ225F40, EP2AGZ300F40, EP2AGZ350F40

Note to Figure 2–12:

(1) Arria II GZ devices with six transceiver blocks allow a maximum of two PCIe x8 links occupying four transceiver blocks. You can configure the other two transceiver blocks to implement other functional modes.

(2) The PCIe hard IP block is only available on the left side of the Arria II device.
Receiver Channel Datapath Clocking

This section describes receiver PMA and PCS datapath clocking in supported configurations. The receiver datapath clocking varies between non-bonded and bonded channel configurations. It also varies with the use of PCS blocks; for example, deskew FIFO and rate matcher.

Non-Bonded Channel Configurations

In non-bonded channel configurations, the receiver PCS blocks of each channel are clocked independently. Each non-bonded channel also has separate rx_analogreset and rx_digitalreset signals that allow independent reset of the receiver PCS logic in each channel.

For more information about transceiver reset and power-down signals, refer to the Reset Control and Power Down in Arria II Devices chapter.

In addition, using the rate matcher block affects PCS clocking in non-bonded channel configurations.

Non-Bonded Receiver Clocking Without Rate Matcher

The following functional modes have non-bonded receiver channel configuration without rate-matcher:

- SRI0
- SONET/SDH
- SDI
- CPRI/OBSAI
- Basic without rate matcher
Figure 2–13 shows receiver datapath clocking in non-bonded channel configurations without rate matcher.

**Figure 2–13. Receiver Datapath Clocking in Non-Bonded Configurations without Rate Matcher**

Note to Figure 2–13:

(1) In Arria II GX and GZ devices, there is only one dedicated PCIe hard IP, which supports PCIe Gen 1 x1, x4, and x8; and PCIe Gen 2 x1 and x4.
In non-bonded configurations without rate matcher, the CDR in each receiver channel recovers the serial clock from the received data. The serial recovered clock frequency is half the configured data rate due to the half-rate CDR architecture. The PMA receiver divides the serial recovered clock to generate the parallel recovered clock. The deserializer uses the serial recovered clock in the receiver PMA. The parallel recovered clock and deserialized data is forwarded to the receiver PCS. The parallel recovered clock in each channel clocks the word aligner and 8B/10B decoder (if enabled).

If the configured functional mode does not use the byte deserializer, the parallel recovered clock also clocks the write side of the receiver phase compensation FIFO. It is also driven on the `rx_clkout` port as the FPGA fabric-transceiver interface clock. You can use the `rx_clkout` signal to latch the receiver data and status signals in the FPGA fabric.

If the configured functional mode uses the byte deserializer, the parallel recovered clock is divided by two. This divide-by-two version of the parallel recovered clock clocks the read side of the byte deserializer, the byte ordering block (if enabled), and the write side of the receiver phase compensation FIFO. It is also driven on the `rx_clkout` port as the FPGA fabric-transceiver interface clock. You can use the `rx_clkout` signal to latch the receiver data and status signals in the FPGA fabric.

Table 2–4 lists the receiver datapath clock frequencies in non-bonded functional modes without rate matcher.

### Table 2–4. Receiver Datapath Clock Frequencies in Non-Bonded Functional Modes Without Rate Matcher for Arria II Devices

<table>
<thead>
<tr>
<th>Functional Mode (1)</th>
<th>Data Rate</th>
<th>High-Speed Serial Clock Frequency</th>
<th>Low-Speed Parallel Clock Frequency (MHz)</th>
<th>FPGA Fabric-Transceiver Interface Clock Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Without Byte Serializer (MHz)</td>
</tr>
<tr>
<td>SRIO</td>
<td>1.25 Gbps</td>
<td>625 MHz</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5 Gbps</td>
<td>1.25 GHz</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.125 Gbps</td>
<td>1.5625 GHz</td>
<td>312.5</td>
<td></td>
</tr>
<tr>
<td>SONET/SDH OC12</td>
<td>622 Mbps</td>
<td>311 MHz</td>
<td>77.75</td>
<td>77.75</td>
</tr>
<tr>
<td>SONET/SDH OC48</td>
<td>2.488 Gbps</td>
<td>1.244 GHz</td>
<td>311</td>
<td></td>
</tr>
<tr>
<td>HD SDI</td>
<td>1.485 Gbps</td>
<td>742.5 MHz</td>
<td>148.5</td>
<td>148.5</td>
</tr>
<tr>
<td></td>
<td>1.4835 Gbps</td>
<td>741.75 MHz</td>
<td>148.35</td>
<td>148.35</td>
</tr>
<tr>
<td>3G-SDI</td>
<td>2.97 Gbps</td>
<td>1.485 GHz</td>
<td>297</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.967 Gbps</td>
<td>1.4835 Ghz</td>
<td>296.7</td>
<td></td>
</tr>
</tbody>
</table>

Note to Table 2–4:

(1) Altera also supports CPRI and OBSAI. For more information, refer to AN 610: Implementing CPRI and OBSAI Protocols in Altera Devices.
Non-Bonded Receiver Clocking with Rate Matcher

The following functional modes have non-bonded receiver channel configurations with rate-matcher:

- PCIe x1
- GbE
- SRIO
- Basic with rate matcher

Figure 2–14 shows the receiver datapath clocking in non-bonded channel configurations with rate matcher.

Figure 2–14. Receiver Datapath Clocking in Non-Bonded Configurations with Rate Matcher
In non-bonded configurations with rate matcher, the CDR in each receiver channel recovers the serial clock from the received data. Also, the serial recovered clock frequency is half the configured data rate due to the half rate CDR architecture. The serial recovered clock is divided within the receiver PMA to generate the parallel recovered clock. The deserializer uses the serial recovered clock in the receiver PMA. The parallel recovered clock and deserialized data is forwarded to the receiver PCS.

The parallel recovered clock from the receiver PMA in each channel clocks the word aligner and the write port of the rate match FIFO. The low-speed parallel clock from the transmitter local clock divider block in each channel clocks the read port of the rate match FIFO, the 8B/10B decoder, and the write port of the byte deserializer (if enabled). The parallel transmitter PCS clock or its divide-by-two version (if byte deserializer is enabled) clocks the write port of the receiver phase compensation FIFO. It is also driven on the tx_clkout port as the FPGA fabric-transceiver interface clock. You can use the tx_clkout signal to latch the receiver data and status signals in the FPGA fabric.

Table 2–5 lists the receiver datapath clock frequencies in non-bonded functional modes with rate matcher.

Table 2–5. Receiver Datapath Clock Frequencies in Non-Bonded Functional Modes with Rate Matcher for Arria II Devices

<table>
<thead>
<tr>
<th>Functional Mode</th>
<th>Data Rate (Gbps)</th>
<th>Serial Recovered Clock Frequency</th>
<th>Parallel Recovered Clock Frequency (MHz)</th>
<th>FPGA Fabric-Transceiver Interface Clock Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Without Byte Serializer (MHz)</td>
</tr>
<tr>
<td>PCIe x1 (Gen 1)</td>
<td>2.5</td>
<td>1.25 GHz</td>
<td>250</td>
<td>250 (1)</td>
</tr>
<tr>
<td>PCIe x1 (Gen 2)</td>
<td>5</td>
<td>2.5 GHz</td>
<td>500</td>
<td>—</td>
</tr>
<tr>
<td>GbE</td>
<td>1.25</td>
<td>625 MHz</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>SRIIO</td>
<td>1.25</td>
<td>625 MHz</td>
<td>125</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>1.25 GHz</td>
<td>250</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>3.125</td>
<td>1.5625 GHz</td>
<td>312.5</td>
<td>—</td>
</tr>
</tbody>
</table>

Note to Table 2–5:
(1) 250 MHz when you enable the PCIe hard IP.

**Bonded Channel Configurations**

Arria II GX and GZ devices support x4 channel bonding that allows bonding of four channels within the same transceiver block. It also supports x8 channel bonding that allows bonding of eight channels across two transceiver blocks in PCIe mode.

**x4 Bonded Channel Configuration**

The following functional modes support x4 receiver channel bonded configuration:

- PCIe x4
- XAUI
- Basic x4

In x4 bonded channel configurations, the receiver datapath clocking varies, depending on whether the configured functional mode uses the deskew FIFO or not.
x4 Bonded Channel Configuration with Deskew FIFO

XAUI functional mode has x4 bonded channel configuration with deskew FIFO.

Figure 2–15 shows receiver datapath clocking in x4 channel bonding configurations with deskew FIFO.

Figure 2–15. Receiver Datapath Clocking in x4 Bonded Channel Configuration with Deskew FIFO
In x4 bonded channel configurations with deskew FIFO, the CDR in each receiver channel recovers the serial clock from the received data. Also, the serial recovered clock frequency is half the configured data rate due to the half-rate CDR architecture. The serial recovered clock is divided within each channel’s receiver PMA to generate the parallel recovered clock. The deserializer uses the serial recovered clock in the receiver PMA. The parallel recovered clock and deserialized data is forwarded to the receiver PCS in each channel.

The parallel recovered clock from the receiver PMA in each channel clocks the word aligner in that channel. The parallel recovered clock from the Channel 0 clocks the deskew FIFO and the write port of the rate match FIFO in all four bonded channels. The low-speed parallel clock from the CMU0 clock divider block clocks the read port of the rate match FIFO, the 8B/10B decoder, and the write port of the byte deserializer (if enabled) in all four bonded channels. The low-speed parallel clock or its divide-by-two version (if byte deserializer is enabled) clocks the write port of the receiver phase compensation FIFO. It is also driven on the coreclkout port as the FPGA fabric-transceiver interface clock. You can use the coreclkout signal to latch the receiver data and status signals in the FPGA fabric for all four bonded channels.

In x4 bonded channel configurations, the receiver phase compensation FIFOs in all four bonded channels share common read and write pointers and enable signals generated in the CMU0 block of the transceiver block.

Table 2–6 lists the receiver datapath clock frequencies in x4 bonded functional modes with deskew FIFO.

**Table 2–6. Receiver Datapath Clock Frequencies in x4 Bonded Functional Modes with Deskew FIFO for Arria II Devices**

<table>
<thead>
<tr>
<th>Functional Mode</th>
<th>Data Rate (Gbps)</th>
<th>Serial Recovered Clock Frequency (GHz)</th>
<th>Parallel Recovered Clock and Parallel Transmitter PCS Clock Frequency (MHz)</th>
<th>FPGA Fabric-Transceiver Interface Clock Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIe x4 (Gen 1)</td>
<td>2.5</td>
<td>1.25</td>
<td>250</td>
<td>— (1)</td>
</tr>
<tr>
<td>PCIe x4 (Gen 2)</td>
<td>5</td>
<td>2.5</td>
<td>500</td>
<td>—</td>
</tr>
<tr>
<td>XAUI</td>
<td>3.125</td>
<td>1.5625</td>
<td>312.5</td>
<td>—</td>
</tr>
</tbody>
</table>

**Notes to Table 2–6:**
(1) 250 MHz when you enable the PCIe hard IP.
(2) Arria II GZ devices only.
x4 Bonded Channel Configurations Without Deskew FIFO

PCIe x4 functional modes have x4 bonded channel configurations without deskew FIFO.

Figure 2–16 shows receiver datapath clocking in x4 channel bonding configurations without deskew FIFO.

Figure 2–16. Receiver Datapath Clocking in x4 Bonded Channel Configurations Without Deskew FIFO
In x4 bonded channel configurations without deskew FIFO, the CDR in each receiver channel recovers the serial clock from the received data. The serial recovered clock frequency is half the configured data rate due to the half-rate CDR architecture. The serial recovered clock is divided within each channel’s receiver PMA to generate the parallel recovered clock. The deserializer uses the serial recovered clock in the receiver PMA. The parallel recovered clock and deserialized data is forwarded to the receiver PCS in each channel.

The parallel recovered clock from the receiver PMA in each channel clocks the word aligner and write side of the rate match FIFO in that channel. The low-speed parallel clock from the CMU0 clock divider block in the CMU0 Channel clocks the read port of the rate match FIFO, the 8B/10B decoder, and the write port of the byte deserializer (if enabled). The low-speed parallel clock or its divide-by-two version (if byte deserializer is enabled) clocks the receiver phase compensation FIFO. It is also driven on the coreclkout port as the FPGA fabric-transceiver interface clock. You can use the coreclkout signal to latch the receiver data and status signals in the FPGA fabric for all four bonded channels.

In x4 bonded channel configurations, the receiver phase compensation FIFOs in all four bonded channels share common read and write pointers and enable signals generated in the CMU0 channel of the transceiver block.

Table 2–7 lists the receiver datapath clock frequencies in x4 bonded functional modes without deskew FIFO.

### Table 2–7. Receiver Datapath Clock Frequencies in x4 Bonded Functional Modes Without Deskew FIFO for Arria II Devices

<table>
<thead>
<tr>
<th>Functional Mode</th>
<th>Data Rate (Gbps)</th>
<th>Serial Recovered Clock Frequency (GHz)</th>
<th>Parallel Recovered Clock and Parallel Transmitter PCS Clock Frequency (MHz)</th>
<th>FPGA Fabric-Transceiver Interface Clock Frequency Without Byte Serializer (MHz)</th>
<th>With Byte Serializer (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIe x4 (Gen 1)</td>
<td>2.5</td>
<td>1.25</td>
<td>250</td>
<td>250 (1)</td>
<td>125</td>
</tr>
<tr>
<td>PCIe x4 (Gen 2)</td>
<td>5</td>
<td>2.5</td>
<td>500</td>
<td>—</td>
<td>250</td>
</tr>
</tbody>
</table>

**Notes to Table 2–7:**

(1) 250 MHz when you enable the PCIe hard IP.
(2) Arria II GZ devices only.

### x8 Bonded Channel Configuration

PCIe x8 and Basic x8 functional mode supports x8 receiver channel bonding configurations. The eight bonded channels are located in two transceiver blocks, referred to as the master transceiver block and slave transceiver block, with four channels each.
Figure 2–17 shows receiver datapath clocking in x8 bonded channel configuration.

The CDR in each of the eight receiver channels recovers the serial clock from the received data on that channel. The serial recovered clock frequency is half the configured data rate. The serial recovered clock is divided within each channel’s receiver PMA to generate the parallel recovered clock. The deserializer uses the serial recovered clock in the receiver PMA. The parallel recovered clock and deserialized data from the receiver PMA in each channel is forwarded to the receiver PCS in that channel.
The parallel recovered clock from the receiver PMA in each channel clocks the word aligner and write side of the rate match FIFO in that channel. The low-speed parallel clock from the CMU0 clock divider of the master transceiver block clocks the read port of the rate match FIFO, the 8B/10B decoder, and the write port of the byte deserializer (if enabled) in all eight channels. The low-speed parallel clock or its divide-by-two version (if byte-deserializer is enabled) clocks the write port of the receiver phase compensation FIFO in all eight channels. It is also driven on the coreclkout port as the FPGA fabric-transceiver interface clock. You can use the coreclkout signal to latch the receiver data and status signals in the FPGA fabric for all eight bonded channels.

Both the receiver phase compensation FIFO pointers and the control circuitry from Channel 0 in the master transceiver block are shared by the receiver phase compensation FIFOs across all eight channels in PCIe x8 mode.

Table 2–8 lists the receiver datapath clock frequencies in PCIe x8 functional mode.

### Table 2–8. Receiver Datapath Clock Frequencies in PCIe x8 Functional Modes for Arria II Devices

<table>
<thead>
<tr>
<th>Functional Mode</th>
<th>Data Rate (Gbps)</th>
<th>Serial Recovered Clock Frequency (GHz)</th>
<th>Parallel Recovered clock and Parallel Transmitter PCS Clock Frequency (MHz)</th>
<th>FPGA Fabric-Transceiver Interface Clock Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIe x8 (Gen 1)</td>
<td>2.5</td>
<td>1.25</td>
<td>250</td>
<td>— (1)</td>
</tr>
<tr>
<td>PCIe x8 (Gen 2)</td>
<td>5</td>
<td>2.5</td>
<td>500</td>
<td>250</td>
</tr>
</tbody>
</table>

Notes to Table 2–8:

(1) 250 MHz when you enable the PCIe hard IP.
(2) Arria II GZ devices only.

### FPGA Fabric-Transceiver Interface Clocking

The FPGA fabric-transceiver interface clocks consist of clock signals from the FPGA fabric to the transceiver blocks and clock signals from the transceiver blocks to the FPGA fabric.

The FPGA fabric-transceiver interface clocks are divided into the following three categories:

- “Input Reference Clocks”
- “Phase Compensation FIFO Clocks” on page 2–29
- “Other Transceiver Clocks” on page 2–29

### Input Reference Clocks

The CMU PLLs and receiver CDRs in each transceiver block derive the input reference from one of the following sources:

- refclk0 and refclk1 pins of the same transceiver block
- refclk0 and refclk1 pins of other transceiver blocks on the same side of the device using the ITB clock network
- CLK input pins on the FPGA global clock network
- Clock output pins from the left side and right side PLLs in the FPGA fabric
The input reference clock follows these guidelines:

- If the input reference clock to the CMU PLL or receiver CDR is provided through the FPGA CLK input pins or the clock output from the left PLLs in the FPGA fabric, the input reference clock becomes a part of the FPGA fabric-transceiver interface clocks.
- If the input reference clock is provided through the FPGA CLK input pins, the Quartus II software automatically routes the input reference clock on the FPGA fabric global clock network.
- If the input reference clock is provided through the output clock from a left PLL, the Quartus II software routes the input reference clock on a dedicated clock path from the left PLL to the CMU PLL or receiver CDR.

### Phase Compensation FIFO Clocks

The transmitter and receiver phase compensation FIFOs in each channel ensure the reliable transfer of data, control, and status signals between the FPGA fabric and the transceiver channels. The transceiver channel forwards the `tx_clkout` signal (in non-bonded modes) or the `coreclkout` signal (in bonded channel modes) to the FPGA fabric to clock the data and control signals into the transmitter phase compensation FIFO. The transceiver channel also forwards the recovered clock `rx_clkout` (in configurations without rate matcher) or `tx_clkout/coreclkout` (in configurations with rate matcher) to the FPGA fabric to clock the data and status signals from the receiver phase compensation FIFO into the FPGA fabric.

The phase compensation FIFO clocks form a part of the FPGA fabric-transceiver interface clocks and are routed on either a global clock resource, regional clock resource, or periphery clock resource in the FPGA fabric.

### Other Transceiver Clocks

The following transceiver clocks form a part of the FPGA fabric-transceiver interface clocks:

- `cal_blk_clk`—calibration block clock
- `fixed_clk`—125 MHz fixed-rate clock used in PCIe receiver detect circuitry

The Quartus II software automatically routes `fixed_clk` on the FPGA fabric global clock or regional clock network.

Table 2–9 lists the FPGA fabric-transceiver interface clocks.

### Table 2–9. FPGA Fabric-Transceiver Interface Clocks for Arria II Devices (Part 1 of 2)

<table>
<thead>
<tr>
<th>Clock Name</th>
<th>Clock Description</th>
<th>Interface Direction</th>
<th>FPGA Fabric Clock Resource Utilization (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pll_inclk</code></td>
<td>CMU PLL input reference clock when driven from an FPGA CLK input pin</td>
<td>FPGA fabric-to-transceiver</td>
<td>Global clock</td>
</tr>
<tr>
<td><code>rx_cruclk</code></td>
<td>Receiver CDR input reference clock when driven from an FPGA CLK input pin</td>
<td>FPGA fabric-to-transceiver</td>
<td>Global clock</td>
</tr>
<tr>
<td><code>tx_clkout</code></td>
<td>Phase compensation FIFO clock</td>
<td>Transceiver-to-FPGA fabric</td>
<td>Global, Regional, Periphery clocks</td>
</tr>
</tbody>
</table>
**FPGA Fabric-Transmitter Interface Clocking**

The transmitter phase compensation FIFO compensates for the phase difference between the FPGA fabric clock (phase compensation FIFO write clock) and the parallel transmitter PCS clock (phase compensation FIFO read clock). The transmitter phase compensation FIFO write clock forms the FPGA fabric-transmitter interface clock. The phase compensation FIFO write and read clocks must have exactly the same frequency, in other words, 0 parts per million (PPM) frequency difference.

Arria II GX and GZ transceivers provide the following two options for selecting the transmitter phase compensation FIFO write clock:

- Quartus II software-selected transmitter phase compensation FIFO write clock
- User-selected transmitter phase compensation FIFO write clock

**Quartus II Software-Selected Transmitter Phase Compensation FIFO Write Clock**

If you do not select the `tx_coreclk` port in the ALTGX MegaWizard™ Plug-In Manager, the Quartus II software automatically selects the transmitter phase compensation FIFO write clock for each channel in that ALTGX instance. The Quartus II software selects the FIFO write clock depending on the channel configuration.

**Non-Bonded Channel Configuration**

In the non-bonded channel configuration, the transmitter channels may or may not be identical. Identical transmitter channels are defined as channels that have exactly the same CMU PLL input reference clock source, have exactly the same CMU PLL configuration, and have exactly the same transmitter PMA and PCS configuration.

Identical transmitter channels may have different transmitter voltage output differential (VOD) or pre-emphasis settings.
Example 1: Four Identical Channels in a Transceiver Block

If all four channels within a transceiver block are identical, the Quartus II software automatically drives the write port of the transmitter phase compensation FIFO in all four channels with tx_clkout[0], as shown in Figure 2–18. Use the tx_clkout[0] signal to clock the transmitter data and control logic for all four channels in the FPGA fabric.

This configuration uses only one FPGA global, regional, and periphery clock resource for tx_clkout[0].

Figure 2–18. Four Identical Channels in a Transceiver Block for Example 1
Example 2: Two Groups of Two Identical Channels in a Transceiver Block

Example 2 assumes channels 0 and 1, driven by CMU0 PLL in a transceiver block, are identical. Also, channels 2 and 3, driven by CMU1 PLL in the same transceiver block, are identical. In this case, the Quartus II software automatically drives the write port of the transmitter phase compensation FIFO in channels 0 and 1 with the tx_clkout[0] signal. It also drives the write port of the transmitter phase compensation FIFO in channels 2 and 3 with the tx_clkout[2] signal. Use the tx_clkout[0] signal to clock the transmitter data and control logic for channels 0 and 1 in the FPGA fabric. Use the tx_clkout[2] signal to clock the transmitter data and control logic for channels 2 and 3 in the FPGA fabric.

This configuration uses two FPGA clock resources (global, regional, or both), one for the tx_clkout[0] signal and one for the tx_clkout[2] signal.
Figure 2–19 shows FPGA fabric-transmitter interface clocking for Example 2.
**Bonded Channel Configuration**

In the x4 bonded channel configuration, all four channels within the transceiver block are identical. The Quartus II software automatically drives the write port of the transmitter phase compensation FIFO in all four channels with the coreclkout signal. Use the coreclkout signal to clock the transmitter data and control logic for all four channels in the FPGA fabric.

In the x8 bonded channel configuration, all eight channels across two transceiver blocks are identical. The Quartus II software automatically drives the write port of the transmitter phase compensation FIFO in all eight channels with the coreclkout signal from the master transceiver block. Use the coreclkout signal to clock the transmitter data and control logic for all eight channels in the FPGA fabric.
Figure 2–20 shows FPGA fabric-transmitter interface clocking in an x4 bonded channel configuration.

**Figure 2–20. FPGA Fabric-Transmitter Interface Clocking in an x4 Bonded Channel Configuration**
Limitations of the Quartus II Software-Selected Transmitter Phase Compensation FIFO Write Clock

The Quartus II software uses a single tx_clkout signal to clock the transmitter phase compensation FIFO write port of all identical channels within a transceiver block. This usage results in one global or regional clock resource for each group of identical channels within a transceiver block.

For identical channels located across transceiver blocks, the Quartus II software does not use a single tx_clkout signal to clock the write port of the transmitter phase compensation FIFOs for all channels. Instead, it uses one tx_clkout signal for each group of identical channels per transceiver block, resulting in higher clock resource usage.

**Example 3: Sixteen Identical Channels Across Four Transceiver Blocks**

Consider 16 identical transmitter channels located across four transceiver blocks, as shown in Figure 2–21. The Quartus II software uses tx_clkout from Channel 0 in each transceiver block to clock the write port of the transmitter phase compensation FIFO in all four channels of that transceiver block, resulting in four clocks resources (global, regional, or both) being used, one for each transceiver block.

![Figure 2–21. Sixteen Identical Channels Across Four Transceiver Blocks for Example 3](image-url)
Because all 16 channels are identical, using a single tx_clkout to clock the transmitter phase compensation FIFO in all 16 channels results in only one global or regional clock resource being used instead of four. To implement this clocking scheme, you must choose the transmitter phase compensation FIFO write clocks instead of the Quartus II software automatic selection, as described in “User-Selected Transmitter Phase Compensation FIFO Write Clock”.

User-Selected Transmitter Phase Compensation FIFO Write Clock

The ALTGX MegaWizard Plug-In Manager provides an optional tx_coreclk port for each instantiated transmitter channel. If you enable this port, the Quartus II software does not automatically select the transmitter phase compensation FIFO write clock source. Instead, the signal that you drive on the tx_coreclk port of the channel clocks the write side of its transmitter phase compensation FIFO.

Use the flexibility of selecting the transmitter phase compensation FIFO write clock to reduce clock resource usage (global, regional, or both). You can connect the tx_coreclk ports of all identical channels in your design and drive them using a common clock driver that has 0 PPM frequency difference with respect to the FIFO read clocks of all your channels. Use the common clock driver to clock the transmitter data and control logic in the FPGA fabric for all identical channels. This FPGA fabric-transceiver interface clocking scheme uses only one global or regional clock resource for all identical channels in your design.
Example 4: Sixteen Identical Channels Across Four Transceiver Blocks

Figure 2–22 shows 16 identical transmitter channels located across four transceiver blocks. The tx_coreclk ports of all 16 transmitter channels are connected together and driven by a common clock driver. This common clock driver also drives the transmitter data and control logic of all 16 transmitter channels in the FPGA fabric. Only one global or regional clock resource is used with this clocking scheme, compared with four clock resources (global, regional, or both) needed without the tx_coreclk ports (the Quartus II software-selected transmitter phase compensation FIFO write clock).

Figure 2–22. Sixteen Identical Channels Across Four Transceiver Blocks for Example 4

Common Clock Driver Selection Rules

The common clock driver driving the tx_coreclk ports of all identical channels must have 0 PPM frequency difference with respect to the transmitter phase compensation FIFO read clocks of these channels. If there is any frequency difference between the FIFO write clock (tx_coreclk) and the FIFO read clock, the FIFO overflows or under runs, resulting in corrupted data transfer between the FPGA fabric and the transmitter.
Table 2–10 lists the transmitter phase compensation FIFO read clocks that the Quartus II software selects in various configurations.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Transmitter Phase Compensation FIFO Read Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Bonded Channel Configuration</td>
<td>Parallel transmitter PCS clock from the local clock divider in the associated channel (tx_clkout)</td>
</tr>
<tr>
<td>x4 Bonded Channel Configuration</td>
<td>Low-speed parallel clock from the CMU0 clock divider of the associated transceiver block (coreclkout)</td>
</tr>
<tr>
<td>x8 Bonded Channel Configuration</td>
<td>Low-speed parallel clock from the CMU0 clock divider of the master transceiver block (coreclkout from master transceiver block)</td>
</tr>
</tbody>
</table>

To ensure that you understand the 0 PPM clock driver rule, the Quartus II software expects the “GXB 0 PPM Core Clock Setting” user assignment whenever you use the tx_coreclk port to drive the transmitter phase compensation FIFO write clock.

![Failure to make this assignment when using the tx_coreclk port results in a Quartus II compilation error.](image)

**GXB 0 PPM Core Clock Setting**

The GXB 0 PPM core clock setting is intended for advanced users who know the clocking configuration of the entire system and want to reduce the FPGA fabric global and regional clock resource usage. The GXB 0 PPM core clock setting allows the following clock drivers to drive the tx_coreclk ports:

- tx_clkout in non-bonded channel configurations
- coreclkout in bonded channel configurations
- FPGA CLK input pins
- Transceiver REFCLK pins
- Clock output from the left-corner PLLs (PLL_1 and PLL_4)

![The Quartus II software does not allow gated clocks or clocks generated in the FPGA logic to drive the tx_coreclk ports.](image)

Because the GXB 0 PPM core clock setting allows FPGA CLK input pins and transceiver REFCLK pins as the clock driver, the Quartus II compiler cannot determine if there is a 0 PPM difference between the FIFO write clock and read clock for each channel.

![You must ensure that the clock driver for all connected tx_coreclk ports has a 0 PPM difference with respect to the FIFO read clock in those channels.](image)
Table 2–11 lists the Quartus II assignments that you must make in the assignment editor.

**Table 2–11. Quartus II Assignments for Arria II Devices**

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Description</th>
</tr>
</thead>
</table>
| From | Full design hierarchy name of one of the following clock drivers that you choose to drive the tx_coreclk ports of all identical channels (1):
  - tx_clkout
  - coreclkout
  - FPGA CLK input pins
  - Transceiver REFCLK pins
  - Clock output from left corner PLLs |
| To. | tx_dataout pins of all identical channels whose tx_coreclk ports are connected together and driven by the 0 PPM clock driver. |
| Assignment Name | GXB 0 PPM Core Clock Setting |
| Value | ON |

**Note to Table 2–11:**

(1) You can find the full hierarchy name of the 0 PPM clock driver using the Node Finder feature in the Quartus II Assignment Editor.
Example 5: Sixteen Identical Channels Across Four Transceiver Blocks

Figure 2-23 shows 16 identical transmitter channels located across four transceiver blocks. The tx_coreclk ports of all 16 transmitter channels are connected together and driven by the tx_clkout[4] signal from channel 0 in transceiver block GXBL1. The tx_clkout[4] signal also drives the transmitter data and control logic of all 16 transmitter channels in the FPGA fabric. Only one global clock resource is used by the tx_clkout[4] signal with this clocking scheme.

Table 2-12 lists the Quartus II assignments that you must make for the clocking scheme shown in Figure 2-23.

Table 2-12. Quartus II Assignments for Arria II Devices

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From top_level/top_xcvr_instance1/altgx_component/tx_clkout[4] (1)</td>
<td></td>
</tr>
<tr>
<td>To tx_dataout[15..0]</td>
<td></td>
</tr>
<tr>
<td>Assignment Name</td>
<td>GXB 0 PPM Core Clock Setting</td>
</tr>
<tr>
<td>Value</td>
<td>ON</td>
</tr>
</tbody>
</table>

Note to Table 2-12:

(1) This is an example design hierarchy path for the tx_clkout[4] signal.
FPGA Fabric-Receiver Interface Clocking

The receiver phase compensation FIFO compensates for the phase difference between the parallel receiver PCS clock (FIFO write clock) and the FPGA fabric clock (FIFO read clock). The receiver phase compensation FIFO read clock forms the FPGA fabric-receiver interface clock. The FIFO write and read clocks must have exactly the same frequency, in other words, 0 PPM frequency difference.

Arria II GX and GZ transceivers provide the following two options for selecting the receiver phase compensation FIFO read clock:

- Quartus II software-selected receiver phase compensation FIFO read clock
- User-selected receiver phase compensation FIFO read clock

Quartus II Software-Selected Receiver Phase Compensation FIFO Read Clock

If you do not select the rx_coreclk port in the ALTGX MegaWizard Plug-In Manager, the Quartus II software automatically selects the receiver phase compensation FIFO read clock for each channel in that ALTGX instance. The Quartus II software selects the FIFO read clock depending on the channel configuration.

Non-Bonded Channel Configuration with Rate Matcher

In the non-bonded channel configuration, the transceiver channels may or may not be identical. Identical transceiver channels are defined as channels that have the same CMU PLL and receiver CDR input reference clock source, have exactly the same CMU PLL and receiver CDR configuration, and have exactly the same PMA and PCS configuration.

**Example 6: Four Identical Channels in a Transceiver Block**

If all four channels within a transceiver block are identical, the Quartus II software automatically drives the read port of the receiver phase compensation FIFO in all four channels with tx_clkout[0], as shown in Figure 2–24. Use the tx_clkout[0] signal to latch the receiver data and status signals from all four channels in the FPGA fabric.
This configuration uses only one FPGA global or regional clock resource for tx_clkout[0].

Figure 2-24. Four Identical Channels in a Transceiver Block for Example 6
Example 7: Two Groups of Two Identical Channels in a Transceiver Block

This example assumes channels 0 and 1, driven by CMU0 PLL in a transceiver block, are identical. Also, channels 2 and 3, driven by CMU1 PLL in the same transceiver block, are identical. In this case, the Quartus II software automatically drives the read port of the receiver phase compensation FIFO in channels 0 and 1 with the tx_clkout[0] signal. It also drives the read port of the receiver phase compensation FIFO in channels 2 and 3 with the tx_clkout[2] signal. Use the tx_clkout[0] signal to latch the receiver data and status signals from channels 0 and 1 in the FPGA fabric. Use the tx_clkout[2] signal to latch the receiver data and status signals from channels 2 and 3 in the FPGA fabric.

This configuration uses two FPGA clock resources (global, regional, or both), one for the tx_clkout[0] signal, and one for the tx_clkout[2] signal.
Figure 2–25 shows FPGA fabric-receiver interface clocking for Example 7.

Figure 2–25. FPGA Fabric-Receiver Interface Clocking for Example 7
Non-Bonded Channel Configuration without Rate Matcher

In the non-bonded channel configuration without the rate matcher, the Quartus II software cannot determine if the incoming serial data in all channels have a 0 PPM frequency difference. The Quartus II software automatically drives the read port of the receiver phase compensation FIFO in each channel with the recovered clock driven on the \( rx_{\text{clkout}} \) port of that channel. Use the \( rx_{\text{clkout}} \) signal from each channel to latch its receiver data and status signals in the FPGA fabric.

This configuration uses one FPGA clock resource (global, regional, or both) per channel for the \( rx_{\text{clkout}} \) signal.

Figure 2–26 shows the FPGA fabric-receiver interface clocking for non-bonded channel configurations without rate matcher.

Figure 2–26. FPGA Fabric-Receiver Interface Clocking for Non-Bonded Channel Configurations without Rate Matcher
Bonded Channel Configuration

All bonded transceiver channel configurations have a rate matcher in the receiver data path.

In the x4 bonded channel configurations, the Quartus II software automatically drives the read port of the receiver phase compensation FIFO in all four channels with the coreclkout signal. Use the coreclkout signal to latch the receiver data and status signals from all four channels in the FPGA fabric.

In the x8 bonded channel configurations, the Quartus II software automatically drives the read port of the receiver phase compensation FIFO in all eight channels with the coreclkout signal from the master transceiver block. Use the coreclkout signal to latch the receiver data and status signals from all eight channels in the FPGA fabric.

This configuration uses one FPGA global or regional clock resource per bonded link for the coreclkout signal.
Figure 2–27 shows FPGA fabric-receiver interface clocking in an x4 bonded channel configuration.

**Figure 2–27. FPGA Fabric-Receiver Interface Clocking in an x4 Bonded Channel Configuration**
Limitations of Quartus II Software-Selected Receiver Phase Compensation FIFO Read Clock

In the non-bonded channel configurations without a rate matcher, the Quartus II software cannot determine if the incoming serial data in all channels has a 0 PPM frequency difference. The Quartus II software uses the recovered clock *rx_clkout* signal from each channel to clock the read port of its receiver phase compensation FIFO, resulting in one clock resource (global, regional, or both) being used per channel for the *rx_clkout* signal.
Example 8: Sixteen Channels Across Four Transceiver Blocks

Consider 16 non-bonded receiver channels without a rate matcher located across four transceiver blocks, as shown in Figure 2–28. The incoming serial data for all 16 channels has a 0 PPM frequency difference with respect to each other. The Quartus II software uses \texttt{rx\_clkout} from each channel to clock the read port of its receiver phase compensation FIFO, resulting in 16 clocks resources (global, regional, or both) being used, one for each channel.

Figure 2–28. Sixteen Non-Bonded Receiver Channels without Rate Matcher for Example 8
Because the recovered clock $rx_{\text{clkout}}$ signals from all 16 channels have a 0 PPM frequency difference, you can use a single $rx_{\text{clkout}}$ to clock the receiver phase compensation FIFO in all 16 channels, resulting in only one global or regional clock resource being used instead of 16. To implement this clocking scheme, you must select the receiver phase compensation FIFO read clocks instead of the Quartus II software automatic selection, as described in the “User-Selected Receiver Phase Compensation FIFO Read Clock” section.

**User-Selected Receiver Phase Compensation FIFO Read Clock**

The ALTGX MegaWizard Plug-In Manager provides an optional $rx_{\text{coreclk}}$ port for each instantiated receiver channel. If you enable this port, the Quartus II software does not automatically select the receiver phase compensation FIFO read clock source. Instead, the signal that you drive on the $rx_{\text{coreclk}}$ port of the channel clocks the read side of its receiver phase compensation FIFO.

You can use the flexibility of selecting the receiver phase compensation FIFO read clock to reduce clock resource usage (global, regional, or both). You can connect the $rx_{\text{coreclk}}$ ports of all the receiver channels in your design and drive them using a common clock driver that has a 0 PPM frequency difference with respect to the FIFO write clocks of these channels. Use this common clock driver to latch the receiver data and status signals in the FPGA fabric for these channels. This FPGA fabric transceiver interface clocking scheme uses only one global or regional clock resource for all channels.
Example 9: Sixteen Identical Channels Across Four Transceiver Blocks

Figure 2–29 shows 16 channels located across four transceiver blocks. The incoming serial data to all 16 channels has a 0 PPM frequency difference with respect to each other. The \texttt{rx\_coreclk} ports of all 16 channels are connected together and driven by a common clock driver. This common clock driver also latches the receiver data and status logic of all 16 receiver channels in the FPGA fabric. Only one clock resource (global, regional, or both) is used with this clocking scheme, compared with 16 clock resources (global, regional, or both) needed without the \texttt{rx\_coreclk} ports (the Quartus II software-selected receiver phase compensation FIFO read clock).

Figure 2–29. Sixteen Identical Channels Across Four Transceiver Blocks for Example 9

Common Clock Driver Selection Rules

The common clock driver driving the \texttt{rx\_coreclk} ports of all channels must have a 0 PPM frequency difference with respect to the receiver phase compensation FIFO write clocks of these channels. If there is any frequency difference between the FIFO read clock (\texttt{rx\_coreclk}) and the FIFO write clock, the FIFO overflows or under runs, resulting in corrupted data transfer between the FPGA fabric and the receiver.
Table 2–13 lists the receiver phase compensation FIFO write clocks that the Quartus II software selects in various configurations.

### Table 2–13. Receiver Phase Compensation FIFO Write Clocks for Arria II Devices

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Without Byte Deserializer</th>
<th>With Byte Deserializer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Bonded Channel Configuration with rate matcher</td>
<td>Low-speed parallel clock from the local clock divider in the associated channel (tx_clkout)</td>
<td>Divide-by-two version of the low-speed parallel clock from the local clock divider in the associated channel (tx_clkout)</td>
</tr>
<tr>
<td>Non-Bonded Channel Configuration without rate matcher</td>
<td>Parallel recovered clock from the receiver PMA in the associated channel (rx_clkout)</td>
<td>Divide-by-two version of the parallel recovered clock from the receiver PMA in the associated channel (rx_clkout)</td>
</tr>
<tr>
<td>x4 Bonded Channel Configuration</td>
<td>Low-speed parallel clock from the CMU0 clock divider of the associated transceiver block (coreclkout)</td>
<td>Divide-by-two version of the low-speed parallel clock from the CMU0 clock divider of the associated transceiver block (coreclkout)</td>
</tr>
<tr>
<td>x8 Bonded Channel Configuration</td>
<td>Low-speed parallel clock from the CMU0 clock divider of the master transceiver block (coreclkout from the master transceiver block)</td>
<td>Divide-by-two version of the low-speed parallel clock from the CMU0 clock divider of the master transceiver block (coreclkout from the master transceiver block)</td>
</tr>
</tbody>
</table>

To ensure that you understand the 0 PPM clock driver rule, the Quartus II software expects the “GXB 0 PPM Core Clock Setting” user assignment whenever you use the rx_coreclk port to drive the receiver phase compensation FIFO read clock.

![Warning] Failing to make this assignment correctly when using the rx_coreclk port results in a Quartus II compilation error.

**GXB 0 PPM Core Clock Setting**

The GXB 0 PPM core clock setting is intended for advanced users who know the clocking configuration of the entire system and want to reduce the FPGA fabric global and regional clock resource usage. The GXB 0 PPM core clock setting allows the following clock drivers to drive the rx_coreclk ports:

- tx_clkout in non-bonded channel configurations with rate matcher
- tx_clkout and rx_clkout in non-bonded configurations without rate matcher
- coreclkout in bonded channel configurations
- FPGA CLK input pins
- Transceiver REFCLK pins
- Clock output from the left corner PLLs (PLL_1 and PLL_4)

![Warning] The Quartus II software does not allow gated clocks or clocks generated in FPGA logic to drive the tx_coreclk ports.
Because the 0 PPM clock group assignment allows the FPGA CLK input pins and transceiver REFCLK pins as clock drivers, the Quartus II compiler cannot determine if there is a 0 PPM difference between the FIFO write clock and read clock for each channel.

You must ensure that the clock driver for all connected rx_coreclk ports has a 0 PPM difference with respect to the FIFO write clock in those channels.

Table 2–14 lists the Quartus II assignments that you must make for the clocking scheme shown in Figure 2–29.

Table 2–14. Quartus II Assignments for Arria II Devices

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>Full design hierarchy name of one of the following clock drivers that you choose to drive the rx_coreclk ports of all identical channels (1):</td>
</tr>
<tr>
<td></td>
<td>■ tx_clkout</td>
</tr>
<tr>
<td></td>
<td>■ rx_clkout</td>
</tr>
<tr>
<td></td>
<td>■ coreclkout</td>
</tr>
<tr>
<td></td>
<td>■ FPGA CLK input pins</td>
</tr>
<tr>
<td></td>
<td>■ Transceiver REFCLK pins</td>
</tr>
<tr>
<td></td>
<td>■ Clock output from left and right or top and bottom PLLs</td>
</tr>
<tr>
<td>To</td>
<td>rx_datain pins of all channels whose rx_coreclk ports are connected together and driven by the 0 PPM clock driver.</td>
</tr>
<tr>
<td>Assignment Name</td>
<td>GXB 0 PPM Core Clock Setting</td>
</tr>
<tr>
<td>Value</td>
<td>ON</td>
</tr>
</tbody>
</table>

Note to Table 2–14:

(1) You can find the full hierarchy name of the 0 PPM clock driver using the Node Finder feature in the Quartus II Assignment Editor.
Example 10: Sixteen Channels Across Four Transceiver Blocks

Figure 2–30 shows 16 non-bonded channels without rate matcher located across four transceiver blocks. The incoming serial data to all 16 channels have a 0 PPM frequency difference with respect to each other. The rx_coreclk ports of all 16 channels are connected together and driven by rx_clkout[9] in transceiver block GXBL2. The rx_clkout[9] also clocks the receiver data and status signals of all 16 channels in the FPGA fabric. Only one global or regional clock resource is used by rx_clkout[9] with this clocking scheme.

Table 2–15 lists the Quartus II assignments that you must make for the clocking scheme shown in Figure 2–30.

Table 2–15. Quartus II Assignments for Example 10 for Arria II Devices

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>top_level/top_xcvr_instance1/altgx_component/rx_clkout[9] (1)</td>
</tr>
<tr>
<td>To</td>
<td>rx_datain[15..0]</td>
</tr>
<tr>
<td>Assignment Name</td>
<td>GXB 0 PPM Core Clock Setting</td>
</tr>
<tr>
<td>Value</td>
<td>ON</td>
</tr>
</tbody>
</table>

Note to Table 2–15:
(1) This is an example design hierarchy path for the rx_clkout[9] signal.
FPGA Fabric PLL-Transceiver PLL Cascading

The CMU PLL synthesizes the input reference clock to generate the high-speed serial clock used in the transmitter PMA. The receiver CDR synthesizes the input reference clock in lock-to-reference mode to generate the high-speed serial clock.

This high-speed serial clock output from the CMU PLL and receiver CDR runs at a frequency that is half the configured data rate. The CMU PLLs and receiver CDRs support multiplication factors (M) of 2, 4, 5, 8, 10, 16, 20, and 25. If you use an on-board crystal oscillator to provide the input reference clock through the dedicated \texttt{REFCLK} pins or ITB lines, the allowed crystal frequencies are limited by the CMU PLL and receiver CDR multiplication factors. The input reference clock frequencies are also limited by the allowed phase frequency detector (PFD) frequency range between 50 MHz and 325 MHz.

Example 11: Channel Configuration for 3 Gbps Data Rate

For a channel configured for 3 Gbps data rate, the high-speed serial clock output from the CMU PLL and receiver CDR must run at 1.5 Gbps. Table 2–16 lists the allowed input reference clock frequencies for Example 11.

<table>
<thead>
<tr>
<th>Multiplication Factor (M)</th>
<th>On-Board Crystal Reference Clock Frequency (MHz)</th>
<th>Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>with /N = 1</td>
<td>With /N = 2</td>
</tr>
<tr>
<td>2</td>
<td>750</td>
<td>1500</td>
</tr>
<tr>
<td>4</td>
<td>375</td>
<td>750</td>
</tr>
<tr>
<td>5</td>
<td>300</td>
<td>600</td>
</tr>
<tr>
<td>8</td>
<td>187.5</td>
<td>375</td>
</tr>
<tr>
<td>10</td>
<td>150</td>
<td>300</td>
</tr>
<tr>
<td>16</td>
<td>93.75</td>
<td>187.5</td>
</tr>
<tr>
<td>20</td>
<td>75</td>
<td>150</td>
</tr>
<tr>
<td>25</td>
<td>60</td>
<td>120</td>
</tr>
</tbody>
</table>

Note to Table 2–16:

(1) Violates the PFD frequency limit of 325 MHz.

For a 3 Gbps data rate, the Quartus II software allows an input reference clock frequency of 60, 75, 93.75, 150, 187.5, 300, 375, and 750 MHz. To overcome this limitation, Arria II GX and GZ devices allow the synthesized clock output from left corner PLLs in the FPGA fabric to drive the CMU PLL and receiver CDR input reference clock. The additional clock multiplication factors available in the left corner PLLs allow more options for on-board crystal oscillator frequencies.

Dedicated Left PLL Cascade Lines Network

Arria II GX devices have a dedicated PLL cascade network on the left side of the device that connects to the input reference clock selection circuitry of the CMU PLLs and receiver CDRs. The dedicated PLL cascade network on the left side of the device connects to the input reference clock selection circuitry of the CMU PLLs and receiver CDRs in transceiver blocks located on the left side of the device.
The dedicated PLL cascade networks are segmented by bidirectional tri-state buffers located along the clock line. Segmentation of the dedicated PLL cascade network allows two left PLLs to drive the cascade clock line simultaneously to provide the input reference clock to the CMU PLLs and receiver CDRs in different transceiver blocks.

The following sections describe the dedicated PLL cascade networks available in the Arria II GX device family.

The FPGA fabric PLLs-transceiver PLLs cascading option is available in the following Arria II GX devices with four channels:

- EP2AGX45CU17
- EP2AGX65CU17

Figure 2–31 shows the FPGA fabric PLLs-transceiver PLLs cascading option allowed in the EP2AGX45CU17 and EP2AGX65CU17 devices.

**Figure 2–31. FPGA Fabric PLLs-Transceiver PLLs Cascading Option Allowed in the EP2AGX45CU17 and EP2AGX65CU17 Devices**

<table>
<thead>
<tr>
<th>Transceiver Block GXBL0</th>
<th>PLL Cascade Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel 3</td>
<td>CDR</td>
</tr>
<tr>
<td>Channel 2</td>
<td>CDR</td>
</tr>
<tr>
<td>Channel 1</td>
<td>CDR</td>
</tr>
<tr>
<td>Channel 0</td>
<td>CDR</td>
</tr>
<tr>
<td>CMU 1 PLL</td>
<td></td>
</tr>
<tr>
<td>CMU 0 PLL</td>
<td></td>
</tr>
</tbody>
</table>

The FPGA fabric PLLs-transceiver PLLs cascading option is available in the following Arria II GX devices with eight channels:

- EP2AGX45DF25
- EP2AGX45DF29
- EP2AGX65DF25
- EP2AGX45DF29
- EP2AGX95DF25
- EP2AGX125DF25

Figure 2–32. FPGA Fabric PLLs-Transceiver PLLs Cascading Option Allowed in the EP2AGX45DF25, EP2AGX45DF29, EP2AGX65DF25, EP2AGX45DF29, EP2AGX95DF25, and EP2AGX125DF25 Devices

The FPGA fabric PLLs-transceiver PLLs cascading option is available in the following Arria II GX devices with twelve channels:

- EP2AGX95EF29
- EP2AGX95EF35
- EP2AGX125EF29
- EP2AGX125EF35
- EP2AGX190EF29
- EP2AGX260EF29
Figure 2–33 shows the FPGA fabric PLLs-transceiver PLLs cascading option allowed in the EP2AGX95EF29, EP2AGX95EF35, EP2AGX125EF29, EP2AGX125EF35, EP2AGX190EF29, and EP2AGX260EF29 devices.

**Figure 2–33. FPGA Fabric PLLs Transceiver PLLs Cascading Option Allowed in the EP2AGX95EF29, EP2AGX95EF35, EP2AGX125EF29, EP2AGX125EF35, EP2AGX190EF29, and EP2AGX260EF29 Devices**
The FPGA fabric PLLs-transceiver PLL’s cascading option is available in the following Arria II GX devices with sixteen channels:

- EP2AGX190FF35
- EP2AGX260FF35

Figure 2–34 shows the FPGA fabric PLLs-transceiver PLLs cascading option allowed in the EP2AGX190FF35 and EP2AGX260FF35 devices.

Figure 2–34. FPGA Fabric PLLs-Transceiver PLL’s Cascading Option Allowed in the EP2AGX190FF35 and EP2AGX260FF35 Devices
Dedicated Left and Right PLL Cascade Network in Arria II GZ Devices

Arria II GZ devices have a dedicated PLL cascade network on the left and right side of the device that connects to the input reference clock selection multiplexer of the CMU PLLs, and receiver CDRs on the left and right side of the device.

The dedicated PLL cascade networks are segmented by bidirectional tri-state buffers located along the clock line. Segmentation of the dedicated PLL cascade network allows two or more left and right PLLs to drive the cascade clock line simultaneously.

Because the number of left and right PLLs and transceiver blocks vary from device to device, the capability of cascading a left and right PLL to the CMU PLLs, and receiver CDRs also varies from device to device.

The following sections describe the Arria II GZ FPGA fabric-transceiver PLL’s cascading for the various device packages.

FPGA Fabric PLLs-Transceiver PLLs Cascading in the 780-Pin Package

Arria II GZ devices in 780-pin packages do not support FPGA fabric PLLs-transceiver PLL’s cascading.

FPGA Fabric PLLs-Transceiver PLLs Cascading in the 1152-Pin Package


Figure 2–35. FPGA Fabric PLLs-Transceiver PLLs Cascading Options Allowed for 1152-Pin Package Devices
FPGA Fabric PLLs-Transceiver PLLs Cascading in the 1517-Pin Package

Figure 2–36 shows the FPGA fabric PLLs-transceiver PLL’s cascading options allowed in the EP2AGZ225F40, EP2AGZ300F40, and EP2AGZ350F40 devices.

For the EP4S40G2KF40, EP4S40G5KF40, EP4S100G2KF40, and EP4S100G5KF40 devices, FPGA fabric PLLs-Transceiver PLLs cascading for the CMU PLLs is the same as the Arria II devices in the 1517-pin package.

Figure 2–36. FPGA Fabric PLLs-Transceiver PLLs Cascading Options Allowed in the 1517-Pin Package Devices
FPGA Fabric PLL-Transceiver PLL Cascading Rules

PLL cascade networks are single clock lines segmented by bidirectional tri-state buffers located along the clock line. Segmentation of the PLL cascade network allows two left PLLs to drive the cascade clock line simultaneously to provide two input reference clocks to the CMU PLLs and receiver CDRs in different transceiver blocks. When cascading two or more FPGA fabric PLLs to the CMU PLLs and receiver CDRs, there must be no crossover in the cascaded clock paths on the PLL cascade network.

Example 12: Design Target-EP2AGX190FF35 Device

Consider a design targeting the EP2AGX190FF35 device and requiring input reference clocks to the following CMU PLLs and receiver CDRs from two left PLLs in the FPGA fabric:

- CMU0 PLL in transceiver block GXBL1
- Receiver CDRs in channel 2 and channel 3 in transceiver block GXBL1

Case 1: PLL_4 is used to provide the input reference clock to the receiver CDRs in channel 2 and channel 3 (shown in green). PLL_1 is used to provide the input reference clock to the CMU0 PLL (shown in blue) in transceiver block GXBL1.
Figure 2–37 shows that this FPGA fabric-transceiver PLL cascading configuration is illegal due to crossover (shown in red) of cascade clock paths on the PLL cascade network.

**Figure 2–37. Illegal FPGA Fabric-Transceiver PLL Cascading Configuration**
Case 2: PLL_1 is used to provide the input reference clock to the receiver CDRs in channel 2 and channel 3 (shown in blue). PLL_4 is used to provide the input reference clock to the CMU0 PLL (shown in green) in transceiver block GXBL1.

Figure 2–38 shows that this FPGA fabric-transceiver PLL cascading configuration is legal because there is no crossover of the cascade clock paths on the PLL cascade network.

Figure 2–38. Legal FPGA Fabric-Transceiver PLL Cascading Configuration
Using the CMU PLL for Clocking User Logic in the FPGA Fabric

Some designs that use multiple clock domains may run out of PLLs in the FPGA fabric. In such a scenario, if your design has CMU PLLs that are not being used, it may be possible to use them for clocking user logic in the FPGA fabric. However, the CMU PLLs do not have many features that are supported by the PLLs in the FPGA fabric.

The following features are supported on CMU PLLs used as PLLs for clocking user logic in the FPGA fabric:

- Single clock output
- Programmable PLL bandwidth
- PLL PFD power down control
- Lock status signal

To use this feature, you must create an ALTGX instance with a single channel in Transmitter Only mode that uses the required CMU PLL. To create the ALTGX instance, complete these steps:

1. Choose Basic mode as the protocol.
2. Select Transmitter Only operation mode.
3. Select the input clock frequency.
4. Select the appropriate values of data rate and channel width based on the desired output clock frequency. To generate a 250 MHz clock using an input clock frequency of 50 MHz, select a channel width of 10 and a data rate of 2500 Mbps (Equation 2–1).

\[ f_{\text{out}} = \frac{\text{data rate}}{\text{channel width}} \]  

(Equation 2–1.)

5. You can select the PLL bandwidth by choosing Tx PLL bandwidth mode.
6. You can instantiate the pll_locked port to indicate the PLL lock status.
7. You can instantiate pll_powerdown or gxb_powerdown to enable the PLL PFD power down control.

Use tx_clkout of the ALTGX instance as the clock source for clocking user logic in the FPGA fabric.
# Document Revision History

Table 2–17 lists the revision history for this chapter.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| June 2011  | 3.1     | ■ Updated Figure 2–11, Figure 2–12, and Figure 2–20.  
■ Updated “Bonded Channel Configurations”, “CMU PLL and Receiver CDR Input Reference Clocking” and “Input Reference Clocks” sections.  
■ Minor text edits. |
| December 2010 | 3.0 | ■ Updated to add Arria II GZ information.  
■ Removed “QL” designations.  
■ Minor text edits. |
| July 2010  | 2.0     | ■ Updated Figure 2–29.  
■ Updated Table 2–2, Table 2–3, and Table 2–4.  
■ Updated the “Non-Bonded Channel Configurations”, “Non-Bonded Receiver Clocking without Rate Matcher”, and “Dedicated Left PLL Cascade Lines Network” sections.  
■ “(QLn)” (where n = 1 or 2) was added to all “GXBn” references.  
■ Minor text edits. |
| November 2009 | 1.1 | ■ Updated figures in the document for clarity.  
■ Added “Using the CMU PLL for Clocking User Logic in the FPGA Fabric” on page 2–64. |
| February 2009 | 1.0 | Initial release. |
This chapter describes the configuration of multiple protocols and data rates for Arria® II GX and GZ devices. Each transceiver channel in an Arria II GX or GZ device can run at an independent data rate or protocol mode. Within each transceiver channel, the transmitter and receiver channel can run at different data rates. Each transceiver block consists of two clock multiplier unit (CMU) phase-locked loops (PLLs) that provide clocks to all the transmitter channels within the transceiver block. Each receiver channel contains a dedicated clock data recovery (CDR).

This chapter includes the following sections:

- “Transceiver PLL Configurations” on page 3–1
- “Creating Transceiver Channel Instances” on page 3–2
- “General Requirements to Combine Channels” on page 3–2
- “Sharing CMU PLLs” on page 3–3
- “Combining Receiver Only Channels” on page 3–8
- “Combining Transmitter Channel and Receiver Channel Instances” on page 3–9
- “Combining Channels Configured in Protocol Functional Modes” on page 3–10
- “Combining Transceiver Instances Using PLL Cascade Clocks” on page 3–12
- “Combining Transceiver Instances in Multiple Transceiver Blocks” on page 3–13
- “Summary” on page 3–15

Transceiver PLL Configurations

You can configure each transmitter channel to use one of the two CMU PLLs in the transceiver block. In addition, each transmitter channel has a local divider (/1, /2, or /4) that divides the clock output of the CMU PLL to provide high-speed serial and low-speed parallel clocks for its physical coding sublayer (PCS) and physical medium attachment (PMA) functional blocks.

You can configure the RX CDR present in the receiver channel to a distinct data rate and provide separate input reference clocks. Each receiver channel also contains a local divider that divides the high-speed clock output of the RX CDR and provides clocks for its PCS and PMA functional blocks. To enable transceiver channel settings, the Quartus® II software provides the ALTGX MegaWizard™ Plug-In Manager interface. The ALTGX MegaWizard Plug-In Manager allows you to instantiate a single transceiver channel or multiple transceiver channels in Receiver and Transmitter, Receiver Only, and Transmitter Only configurations.
Creating Transceiver Channel Instances

You can instantiate multiple transceiver channels in the General screen of the ALTGX MegaWizard Plug-In Manager in the following two different ways:

- For the What is the number of channels? option, select the required value. This method creates the transceiver channels with identical configurations. For examples, refer to “Combining Transceiver Instances in Multiple Transceiver Blocks” on page 3–13.

- For the What is the number of channels? option, select 1 and create a single channel transceiver instance. To instantiate additional transceiver channels with an identical configuration, stamp the created ALTGX instance multiple times. If you require additional transceiver channels with different configurations, create separate ALTGX megafuntion instances with different settings and use them in your design.

When you create instances using the above methods, you can force the placement of up to four transceiver channels within the same transceiver block by assigning the tx_dataout and rx_datain ports of the channel instances to a single transceiver bank. If you do not assign pins to the tx_dataout and rx_datain ports, the Quartus II software chooses default pin assignments. When you compile the design, the Quartus II software combines multiple channel instances within the same transceiver block if the instances meet specific requirements. The following sections describe these requirements for different transceiver configurations.

General Requirements to Combine Channels

When you create multiple ALTGX instances, the Quartus II software requires that you set identical values on the following parameters and signals to combine the ALTGX instances within the same transceiver block or in the transceiver blocks on the same side of the device. The following sections describe these requirements.

Control Signals

The gxb_powerdown port is an optional port that you can enable in the ALTGX MegaWizard Plug-In Manager. If enabled, you must drive the gxb_powerdown port in the ALTGX instances from the same logic or the same input pin to enable the Quartus II software to assign them in the same transceiver block. If you disable the gxb_powerdown port, the Quartus II software ties the port to ground.

Calibration Clock and Power Down

Each calibration block in an Arria II GX or GZ device is shared by multiple transceiver blocks.

If your design uses multiple transceiver blocks, depending on the transceiver banks selected, you must connect the cal_blk_clk and cal_blk_powerdown ports of all channel instances to the same input pin or logic.

For more information about the calibration block and transceiver banks that are connected to a specific calibration block, refer to the “Calibration Block” section in the Transceiver Architecture in Arria II Devices chapter.
Sharing CMU PLLs

Each Arria II GX and GZ transceiver block contains two CMU PLLs. When you create multiple transceiver channel instances and intend to combine them in the same transceiver block, the Quartus II software checks whether a single CMU PLL can be used to provide clock outputs for the transmitter side of the channel instances. If a single CMU PLL is not sufficient, the Quartus II software attempts to combine the channel instances using two CMU PLLs. Otherwise, the Quartus II software issues a Fitter error.

The following two sections describe the ALTGX instance requirements to enable the Quartus II software to share the CMU PLL.

Multiple Channels Sharing a CMU PLL

To enable the Quartus II software to share the same CMU PLL for multiple channels, the following parameters in the channel instantiations must be identical:

- Base data rate (the CMU PLL is configured for this data rate)
- CMU PLL bandwidth setting
- Reference clock frequency
- Input reference clock pin
- pll_powerdown port of the ALTGX instances must be driven from the same logic

Each channel instance can have a different local divider setting. Different settings are useful when you intend to run each channel within the transceiver block at different data rates that are derived from the same base data rate using the local divider values /1, /2, and /4. This is shown in Example 1.

Example 1

Consider a design with four instances in a Receiver and Transmitter configuration in the same transceiver block at the following serial data rates. Assume that each instance contains a channel, is driven from the same clock source, and has the same CMU PLL bandwidth settings.

Table 3–1 lists the configuration for Example 1.

<table>
<thead>
<tr>
<th>User-Created Instance Name</th>
<th>ALTGX MegaWizard Plug-In Manager Settings</th>
<th>Number of Channels</th>
<th>Configuration</th>
<th>Effective Data Rate (Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst0</td>
<td></td>
<td>1</td>
<td>Receiver and Transmitter</td>
<td>3.75</td>
</tr>
<tr>
<td>inst1</td>
<td></td>
<td>1</td>
<td>Receiver and Transmitter</td>
<td>0.9375</td>
</tr>
</tbody>
</table>
You can share a single CMU PLL for all four channels:

- One CMU PLL can be configured to run at 3.75 Gbps.
- Each channel can divide the CMU PLL clock output using the local divider and achieve the required data rates of 3.75 Gbps, 1.875 Gbps, and 0.9375 Gbps. Because each receiver channel has a dedicated CDR, the receiver side in each instance can be set up for these three data rates without restrictions.

The following steps describe how to achieve the configuration.

To enable the Quartus II software to share a single CMU PLL for all four channels, set the following values in the General screen of the ALTGX MegaWizard Plug-In Manager.

- For inst0:
  - Set What is the effective data rate? to 3.75 Gbps
  - Set Specify base data rate to 3.75 Gbps

- For inst1:
  - Set What is the effective data rate? to 1.875 Gbps
  - Set Specify base data rate to 3.75 Gbps

- For inst2:
  - Set What is the effective data rate? to 0.9375 Gbps
  - Set Specify base data rate to 3.75 Gbps

- For inst3:
  - Set What is the effective data rate? to 3.75 Gbps
  - Set Specify base data rate to 3.75 Gbps

The Specify base data rate option is 3.75 Gbps for all four instances. Because the CMU PLL bandwidth setting and input reference clock are the same and the pll_powerdown ports are driven from the same logic or pin, the Quartus II software shares a single CMU PLL that runs at 3.75 Gbps.

You can force the placement of transceiver channels to a specific transceiver block by assigning pins to tx_dataout and rx_datain. Otherwise, the Quartus II software selects a transceiver block.

Figure 3–1 shows the scenario before and after the Quartus II software combines the transceiver channel instances. Because the RX CDR is not shared between channels, only the CMU PLL is shown.

<table>
<thead>
<tr>
<th>User-Created Instance Name</th>
<th>ALTGX MegaWizard Plug-In Manager Settings</th>
<th>Number of Channels</th>
<th>Configuration</th>
<th>Effective Data Rate (Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst2</td>
<td></td>
<td>1</td>
<td>Receiver and Transmitter</td>
<td>1.875</td>
</tr>
<tr>
<td>inst3</td>
<td></td>
<td>1</td>
<td>Receiver and Transmitter</td>
<td>3.75</td>
</tr>
</tbody>
</table>
Each ALTGX instance has a \texttt{pll\_powerdown} port. You must drive the \texttt{pll\_powerdown} ports of all instances from the same logic to allow the Quartus II software to share the same CMU PLL. If you drive the \texttt{pll\_powerdown} ports of the ALTGX instance using different logic, the Quartus II software does not use the same CMU PLL even if all the other required parameters of all the ALTGX instances are identical.

![Figure 3–1. ALTGX Instances Before and After Compilation for Example 1](image-url)
Multiple Channels Sharing Two CMU PLLs

In some cases, a single CMU PLL is not sufficient to run the transmitter channels within a transceiver block at the desired data rates.

Use a second CMU PLL if you want to combine channels that require different configurations, such as:

- Quartus II software-defined protocols (for example, Basic, Gbps Ethernet (GbE), SONET/synchronous digital hierarchy [SDH], Serial Digital Interface [SDI], or PCI Express® [PIPE] [PCIe] modes)
- CMU PLL bandwidth settings
- Different input reference clocks

Example 2

Consider a design that requires four channels set up in a Receiver and Transmitter configuration in the same transceiver block at the serial data rates shown in Table 3–2.

Table 3–2. Configuration for Example 2 for Arria II Devices

<table>
<thead>
<tr>
<th>User-Created Instance Name</th>
<th>ALTGX MegaWizard Plug-In Manager Settings</th>
<th>Number of Channels</th>
<th>Configuration</th>
<th>Effective Data Rate (Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst0</td>
<td></td>
<td>1</td>
<td>Receiver and Transmitter</td>
<td>3.75</td>
</tr>
<tr>
<td>inst1</td>
<td></td>
<td>1</td>
<td>Receiver and Transmitter</td>
<td>1.875</td>
</tr>
<tr>
<td>inst2</td>
<td></td>
<td>1</td>
<td>Receiver and Transmitter</td>
<td>0.9375</td>
</tr>
<tr>
<td>inst3</td>
<td></td>
<td>1</td>
<td>Receiver and Transmitter</td>
<td>2</td>
</tr>
</tbody>
</table>

Assume that instance 0, 1, and 2 are driven from the same clock source and have the same CMU PLL bandwidth settings. In this case, you can use one CMU PLL for instance 0, 1, and 2. For the ALTGX MegaWizard Plug-In Manager settings that enable the Quartus II software to share the same CMU PLL, refer to “Example 1” on page 3–3. A second CMU PLL is required for instance 3.

You can force the placement of transceiver channels to a specific transceiver block by assigning pins to the tx_dataout and rx_datain pins of the four ALTGX instances. Otherwise, the Quartus II software selects a transceiver block.

Figure 3–2 shows the transceiver configuration before and after the Quartus II software combines the transceiver channels within the same transceiver block. Because the RX CDR is not shared between channels, only the CMU PLLs are shown.

You must connect the pll_powerdown port of instance 0, 1, and 2 to the same logic output to share the same CMU PLL for these instances.
Figure 3-2. ALTGX Transceiver Channel Instances Before and After Compilation for Example 2
In cases where you have two instances with the same serial data rate but with different CMU PLL data rates, the Quartus II software creates a separate CMU PLL for the two instances. For example, consider the configuration shown in Table 3–3.

Table 3–3. Sample Configuration Where Instances Cannot Be Combined in a Single Transceiver Block for Arria II Devices

<table>
<thead>
<tr>
<th>User-Created Instance Name</th>
<th>ALTGX MegaWizard Plug-In Manager Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of Channels</td>
</tr>
<tr>
<td>inst0</td>
<td>1</td>
</tr>
<tr>
<td>inst1</td>
<td>1</td>
</tr>
<tr>
<td>inst2</td>
<td>1</td>
</tr>
</tbody>
</table>

Even though the effective data rate of inst1 and inst0 are 1.5 Gbps (3 Gbps/2 = 1.5 Gbps), when you compile the design, the Quartus II software requires two CMU PLLs to provide clocks for the transmitter side of the two instances because their base data rates are different. In this example, you have the third instance (inst2) that requires a third CMU PLL. Therefore, the Quartus II software cannot combine the above three instances within the same transceiver block.

Combining Receiver Only Channels

You can selectively use the receiver in the transceiver channel by selecting the Receiver Only configuration in the What is the Operating Mode? option on the General screen of the ALTGX MegaWizard Plug-In Manager.

You can combine Receiver Only channel instances of different configurations and data rates into the same transceiver block. Because each receiver channel contains its own dedicated CDR, each Receiver Only instance (assuming one receiver channel per instance) can have different data rates.

For the Quartus II software to combine the Receiver Only instances within the same transceiver block, you must connect gxb_powerdown (if used) of all the channel instances from the same logic or input pin. For more information, refer to “General Requirements to Combine Channels” on page 3–2.

It is possible to have up to four receiver channels that can run at different data rates by using separate input reference clocks if there are enough clock routing resources available.

If you instantiate the Receiver Only configuration, the ALTGX MegaWizard Plug-In Manager does not allow you to enable the rate matching FIFO (clock rate compensation FIFO) in the receiver channel PCS because tx_clkout is not available in a Receiver Only instance to clock the read side of the rate matching FIFO. If you have to perform clock rate compensation, implement the rate matching FIFO in the FPGA fabric.

If you create a Receiver Only instance and do not use the transmitter channel that is present in the same physical transceiver channel, the Quartus II software automatically powers down the unused transmitter channel.
Combining Transmitter Channel and Receiver Channel Instances

You can create a separate transmitter channel instance and a separate receiver channel instance and assign the `tx_dataout` and `rx_datain` pins of the transmitter and receiver instance, respectively, in the same physical transceiver channel. This configuration is useful in cases where you intend to run the transmitter and receiver channel at different serial data rates. To create a transmitter channel instance and a receiver channel instance, select the Transmitter Only and Receiver Only options in the operating mode (General screen) of the ALTGX MegaWizard Plug-In Manager.

Multiple Transmitter Channel and Receiver Channel Instances

The Quartus II software allows you to combine multiple Transmitter Only channel and Receiver Only channel instances within the same transceiver block. Based on the pin assignments, the Quartus II software combines the corresponding Transmitter Only and Receiver Only channels in the same physical channel. To enable the Quartus II software to combine the transmitter channel and receiver channel instances in the same transceiver block, follow the rules and requirements outlined in the following sections:

- “General Requirements to Combine Channels” on page 3–2
- “Multiple Channels Sharing a CMU PLL” on page 3–3
- “Multiple Channels Sharing Two CMU PLLs” on page 3–6
- “Combining Receiver Only Channels” on page 3–8

Example 3

Consider that you create four ALTGX instances, as shown in Table 3–4.

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Configuration</th>
<th>Effective Data Rate (Gbps)</th>
<th>Input Reference Clock Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst0</td>
<td>Transmitter only</td>
<td>3.125</td>
<td>156.25</td>
</tr>
<tr>
<td>inst1</td>
<td>Receiver only</td>
<td>2.5</td>
<td>156.25</td>
</tr>
<tr>
<td>inst2</td>
<td>Transmitter only</td>
<td>1.25</td>
<td>125</td>
</tr>
<tr>
<td>inst3</td>
<td>Receiver only</td>
<td>2</td>
<td>125</td>
</tr>
</tbody>
</table>

After you create the above instances, if you force the placement of the instances shown in Table 3–5, the Quartus II software combines inst0 and inst1 into physical channel 0 and inst2 and inst3 into physical channel 1.

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Physical Channel Pin Assignments in the Same Transceiver Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst0</td>
<td>TX pin of channel 0</td>
</tr>
<tr>
<td>inst1</td>
<td>RX pin of channel 0</td>
</tr>
<tr>
<td>inst2</td>
<td>TX pin of channel 1</td>
</tr>
<tr>
<td>inst3</td>
<td>RX pin of channel 1</td>
</tr>
</tbody>
</table>
Combining Channels Configured in Protocol Functional Modes

The following sections describe combining channels that are configured in protocol functional modes.

Basic ×4 Mode

The ALTGX MegaWizard Plug-In Manager provides a Basic mode with a ×4 option in the Which sub-protocol will you be using? option on the General screen. If you select this option, all the transmitter channels within the transceiver block receive the high-speed serial and low-speed parallel clock from the CMU0 clock divider block (present in the CMU0 channel). Each receiver channel within the transceiver block is clocked independently by the recovered clock from its receiver CDR.
When you use this mode, the ALTGX MegaWizard Plug-In Manager allows you to select one or more channels from the **What is the number of channels?** option on the **General** screen.

If you select the number of channels to be less than four, the remaining transmitter channels cannot be used within the transceiver block. Therefore, if you have more than one instance configured in **Transmit Only** or **Receiver and Transmitter** mode with the ×4 option enabled, the Quartus II software cannot combine the instances within the same transceiver block.

Only the transmitter channels share a common clock. The receiver channels are clocked independently. Therefore, you can configure the unused receiver channels within a transceiver block in any allowed configuration. For example, assume that you configure the ALTGX MegaWizard Plug-In Manager with the options shown in Table 3–6.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>What protocol will you be using?</td>
<td>Basic mode</td>
</tr>
<tr>
<td>Which subprotocol will you be using?</td>
<td>×4</td>
</tr>
<tr>
<td>What is the operating mode?</td>
<td>Receiver and Transmitter</td>
</tr>
<tr>
<td>What is the number of channels?</td>
<td>2</td>
</tr>
</tbody>
</table>

If you create the instance with the above selection, you cannot use the remaining two transmitter channels in the transceiver block. However, you can use the remaining two receiver channels in a different configuration.

**Figure 3–4** shows examples of supported and unsupported configurations.

**Figure 3–4. Examples of Supported and Unsupported Configurations to Combine Instances in Basic ×4 Mode**

---

**Supported Configuration**

**Instance 0**

- 2 Channels
- Receiver and Transmitter
- Basic mode and ×4 sub-protocol

**Instance 1**

- 2 Channels
- Receiver only
- Basic mode (any configuration)

**Unsupported Configuration**

**Instance 0**

- 2 Channels
- Receiver and Transmitter
- Basic mode and ×4 sub-protocol

**Instance 1**

- 2 Channels
- Transmitter only
- Basic mode (any configuration)
Combining Channels Using the PCIe hard IP Block with Other Channels

The Arria II device family contains an embedded PCIe hard IP block that performs the physical,datalink, and transaction layer functionality specified by PCIe base specification 1.1. Each PCIe hard IP block is shared by two transceiver blocks. The PCIe Compiler MegaWizard Plug-In Manager provides you the options to configure the PCIe hard IP block. When enabled, the transceiver channels associated with this block are enabled.

There are restrictions on combining transceiver channels with different functional and/or protocol modes (for example, Basic mode) within two contiguous transceiver blocks with the channels that use the PCIe hard IP block. The restrictions depend on the number of channels used (×1 or ×4) and the number of virtual channels (VC) selected in the PCIe Compiler MegaWizard Plug-In Manager. Table 3–7 lists the restrictions.

Table 3–7. PCIe Hard IP Block Restrictions When Combining Transceiver Channels with Different Functional and/or Protocol Modes for Arria II Devices

<table>
<thead>
<tr>
<th>PCI Express Configuration (PCI Express hard IP Options Enabled in the PCIe Compiler Wizard)</th>
<th>Transceiver Block 0 (3)</th>
<th>Transceiver Block 1 (4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link Width</td>
<td>Lane (Data Interface Width)</td>
<td>Virtual Channel (VC)</td>
</tr>
<tr>
<td>4</td>
<td>64 bit</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>128 bit</td>
<td>1</td>
</tr>
</tbody>
</table>

Notes to Table 3–7:

1. Avail.—the channels can be used in other configurations.
2. N/A—the channels are NOT available for use.
3. Transceiver block 0—the master transceiver block that provides high-speed serial and low-speed parallel clocks in a PCIe ×4 or ×8 configuration.
4. Transceiver block 1—the adjacent transceiver block that shares the same PCIe hard IP block with transceiver block 0.
5. The physical channel 0 in the transceiver block. For more information about physical-to-logical channel mapping in PCIe functional mode, refer to the “×8 Channel Configuration” section in the Arria II Transceiver Clocking chapter.

For more information about the PCIe Compiler MegaCore Functions and hard IP implementation, refer to the PCI Express Compiler User Guide.

Combining Transceiver Instances Using PLL Cascade Clocks

The Arria II device family provides multiple input reference clock sources to clock the CMU PLLs and RX CDRs in each transceiver block. The following are the input reference clock sources that can clock the CMU PLLs and RX CDRs:

- refclks from the same transceiver block
- Global clock lines
- refclks from transceiver blocks on the same side of the device using the inter-transceiver block (ITB) lines
- PLL cascade clock (this is the cascaded clock output from the PLLs in the FPGA fabric)
If you use the PLL cascade clock to provide input reference clocks to the CMU PLLs or RX CDRs, there are requirements for combining transceiver channels (as described in the following sections).

The Arria II GX and GZ transceiver can cascade the output of the general purpose PLLs to the CMU PLLs and receiver CDRs. The left side of the Arria II GX and GZ device contains a PLL cascade clock network—a single line network that connects the PLL cascade clock to the transceiver block. Similarly, for Arria II GZ devices, the right side PLLs can only be cascaded with the transceivers on the right side of the device. Each side of the Arria II GZ device contains a PLL cascade clock network. This clock line is segmented to allow different PLL cascade clocks to drive the transceiver CMU PLLs and RX CDRs.

The segmentation locations differ based on the device family. Therefore, there are restrictions when you want to combine transceiver channels that use different PLL cascade clocks as input reference clocks.

For more information about using the PLL cascade clock and segmentation, refer to the “PLL Cascading” section in the Transceiver Clocking in Arria II Devices chapter.

Combining Transceiver Instances in Multiple Transceiver Blocks

“Creating Transceiver Channel Instances” on page 3–2 describes the method to instantiate multiple transceiver channels using a single ALTGX instance. The following section describes the method to instantiate multiple transceiver channels using multiple transceiver blocks.

When you create a transceiver instance that has more than four transceiver channels, the Quartus II software attempts to combine the transceiver channels in multiple transceiver blocks, as shown in Example 4.

**Example 4**

Consider that you create two ALTGX instances with the configuration shown in Table 3–8.

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Number of Transceiver Channels</th>
<th>Configuration</th>
<th>Effective Data Rate (Gbps)</th>
<th>Input Reference Clock (Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst0</td>
<td>7</td>
<td>Receiver and Transmitter</td>
<td>3.125</td>
<td>156.25</td>
</tr>
<tr>
<td>inst1</td>
<td>1</td>
<td>Receiver and Transmitter</td>
<td>3.125</td>
<td>156.25</td>
</tr>
</tbody>
</table>

In this case, assuming that all the required parameters specified in “Multiple Channels Sharing a CMU PLL” on page 3–3 are identical in inst0 and inst1, the Quartus II software fits inst0 and inst1 in two transceiver blocks.
Figure 3–5 shows the transceiver instances before compilation for Example 4.

Figure 3–6 shows the transceiver instances after compilation for Example 4.

You can force the placement of the transceiver channels in specific transceiver banks by assigning pins to the tx_dataout and rx_datain ports of inst0 and inst1.
Even though inst0 instantiates seven transceiver channels, the ALTGX MegaWizard Plug-In Manager provides only one bit for the pll_inclk port for inst0. In your design, provide only one clock input for the pll_inclk port. The Quartus II software uses two transceiver blocks to fit the seven channels and internally connects the input reference clock (connected to the pll_inclk port in your design) to the CMU PLLs of two transceiver blocks.

For inst1, the ALTGX MegaWizard Plug-In Manager provides a pll_inclk port. In this example, it is assumed that a single reference clock is provided for inst0 and inst1. Therefore, connect the pll_inclk port of inst0 and inst1 to the same input reference clock pin to enable the Quartus II software to share a single CMU PLL in transceiver block1 that has three channels of inst0 and one channel of inst1 (shown as ch5, ch6, and ch7 in transceiver block 1) in Figure 3–6 on page 3–14.

For the RX CDRs in inst0, the ALTGX MegaWizard Plug-In Manager provides seven bits for the rx_cruclk port (if you do not select the Train Receiver CDR from pll_inclk option in the PLL/Ports screen), allowing separate input reference clocks to the RX CDRs of each channel.

Summary

The following summarizes how to configure multiple protocols and data rates in a transceiver block:

- You can run each transceiver channel at independent data rates or protocol functional modes.
- Each transceiver block consists of two CMU PLLs that provide clocks to run the transmitter channels within the transceiver block.
- To enable the Quartus II software to combine multiple instances of transceiver channels within a transceiver block, follow the rules specified in “General Requirements to Combine Channels” on page 3–2 and “Sharing CMU PLLs” on page 3–3.
- You can reset each CMU PLL within a transceiver block using a pll_powerdown signal. For each transceiver instance, the ALTGX MegaWizard Plug-In Manager provides an option to select the pll_powerdown port. If you want to share the same CMU PLL between multiple transceiver channels, connect the pll_powerdown ports of the instances and drive the signal from the same logic.
- If you enable the PCIe hard IP block using the PCIe Compiler, the Quartus II software has certain requirements about using the remaining transceiver channels within the transceiver block in other configurations. For more information, refer to “Combining Channels Using the PCIe hard IP Block with Other Channels” on page 3–12.
Document Revision History

Table 3–9 lists the revision history for this chapter.

Table 3–9. Document Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 2010</td>
<td>3.0</td>
<td>■ Updated to add Arria II GZ information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Minor text edits.</td>
</tr>
<tr>
<td>July 2010</td>
<td>2.0</td>
<td>■ Updated the “Transceiver PLL Configurations” and “Combining Channels Using the PCIe hard IP Block with Other Channels” sections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Minor text edits.</td>
</tr>
<tr>
<td>February 2009</td>
<td>1.0</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>
Arria® II GX and GZ devices offer multiple reset signals to control transceiver channels and clock multiplier unit (CMU) phase-locked loops (PLLs) independently. The ALTGX Transceiver MegaWizard™ Plug-In Manager provides individual reset signals for each channel instantiated in the design. It also provides one power-down signal for each transceiver block.

This chapter includes the following sections:

- “User Reset and Power-Down Signals” on page 4–1
- “Transceiver Reset Sequences” on page 4–4
- “Dynamic Reconfiguration Reset Sequences” on page 4–17
- “Hot Socketing Reset Sequence” on page 4–19
- “Power Down” on page 4–20
- “Simulation Requirements” on page 4–21

Figure 4–1 shows the reset control and power-down block for an Arria II GX or GZ device.

**Figure 4–1. Reset Control and Power-Down Block**

```
  tx_digitalreset
  rx_digitalreset
  rx_analogreset
  pll_powerdown
  gxb_powerdown

  Reset Controller
```

### User Reset and Power-Down Signals

Each transceiver channel in the Arria II device family has individual reset signals to reset its physical coding sublayer (PCS) and physical medium attachment (PMA) blocks. Each CMU PLL in the transceiver block has a dedicated reset signal. The transceiver block also has a power-down signal that affects all the channels and CMU PLLs in the transceiver block.
All reset and power-down signals are asynchronous.

Table 4–1 lists the available reset, power-down, and status signals.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reset Signals For Each Transceiver Channel:</strong></td>
<td></td>
</tr>
</tbody>
</table>
| tx_digitalreset \(1\)   | Provides asynchronous reset to all digital logic in the transmitter PCS, including the XAUI transmit state machine, the built-in self test (BIST) pseudo-random binary sequence (PRBS) generator, and the BIST pattern generator.  
This signal is available in the ALTGX MegaWizard Plug-In Manager in **Transmitter Only** and **Receiver and Transmitter** configurations. The minimum pulse width for this signal is two parallel clock cycles. |
| rx_digitalreset \(1\)   | Resets all digital logic in the receiver PCS, including the XAUI and GbE receiver state machine, the XAUI channel alignment state machine, the BIST-PRBS verifier, and the BIST-incremental verifier.  
This signal is available in the ALTGX MegaWizard Plug-In Manager in **Receiver Only** and **Receiver and Transmitter** configurations. The minimum pulse width for this signal is two parallel clock cycles. |
| rx_analogreset           | Resets the receiver CDR, receiver deserializer, and signal detect in the receiver buffer.  
This signal is available in the ALTGX MegaWizard Plug-In Manager in **Receiver Only** and **Receiver and Transmitter** configurations. The minimum pulse width is two parallel clock cycles. The **busy** signal has precedence over the **rx_analogreset** assertion. When the **busy** signal is high, **rx_analogreset** is ignored. |
| **Power-Down Signal For Each CMU PLL in the Transceiver Block:**                                                                    |                                                                                                                                                                                                           |
| pll_powerdown \(2\)     | Each transceiver block has two CMU PLLs. Each CMU PLL has a dedicated power-down signal called **pll_powerdown**. The **pll_powerdown** signal powers down the CMU PLLs that provide high-speed serial and low-speed parallel clocks to the transceiver channels.  
Note: While each CMU PLL has its own **pll_powerdown** port, the ALTGX MegaWizard Plug-In Manager instantiation provides only one port per transceiver block. This port powers downs one or both CMU PLLs (if used). |
| **Power-Down Signal Common to the Transceiver Block:**                                                                             |                                                                                                                                                                                                           |
| gxb_powerdown \(2\)     | Powers down the entire transceiver block. When this signal is asserted, the PCS and PMA in all the transceiver channels and the CMU PLLs are powered down. This signal operates independently from the other reset signals |
| **Status Signals:**                                                                                                                   |                                                                                                                                                                                                           |
| pll_locked              | Indicates the status of the transmitter PLL. A high level on this signal indicates that the transmitter PLL is locked to the incoming reference clock frequency.                                               |
| rx_pll_locked           | A high level on this signal indicates that the receiver CDR is locked to the incoming reference clock frequency.                                                                                              |
| rx_freqlocked           | Indicates the status of the receiver CDR lock mode. A high level indicates that the receiver is in lock-to-data mode. A low level indicates that the receiver CDR is in lock-to-reference mode.   |
Table 4–1. Reset, Power-Down, and Status Signals for Arria II Devices  (Part 2 of 2)

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>When the busy signal is high during offset cancellation, the signal detect (in the receiver buffer), receiver CDR, and receiver deserializer are enabled regardless of the rx_analogreset signal state. However, the rx_pll_locked and rx_freqlocked signals always deasserted if rx_analogreset is asserted regardless of the busy signal.</td>
</tr>
</tbody>
</table>

Notes to Table 4–1:
(1) The tx_digitalreset and rx_digitalreset signals must be asserted until the clocks out of the transmitter PLL and receiver CDR are stabilized. Stable parallel clocks are essential for proper operation of the transmitter and receiver phase compensation FIFOs in the PCS.
(2) The refclk (refclk0 or refclk1) buffer is not powered down by this signal.

For more information, refer to *AN 558: Implementing Dynamic Reconfiguration in Arria II Devices*.

If none of the channels is instantiated in a transceiver block, the Quartus® II software automatically powers down the entire transceiver block.

**Blocks Affected by Reset and Power-Down Signals**
Table 4–2 lists the blocks that are affected by specific reset and power-down signals.

Table 4–2. Blocks Affected by Reset and Power-Down Signals for Arria II Devices  (Part 1 of 2)

<table>
<thead>
<tr>
<th>Transceiver Block</th>
<th>rx_digitalreset</th>
<th>rx_analogreset</th>
<th>tx_digitalreset</th>
<th>pll_powerdown</th>
<th>gxb_powerdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMU PLLs</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transmitter Phase Compensation FIFO</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Byte Serializer</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>8B/10B Encoder</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Serializer</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Transmitter Buffer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Transmitter XAUI State Machine</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Receiver Buffer</td>
<td>✓ (1)</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Receiver CDR</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Receiver Deserializer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Receiver Word Aligner</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Receiver Deskew FIFO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Receiver Clock Rate Compensation FIFO</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Receiver 8B/10B Decoder</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Receiver Byte Deserializer</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Receiver Byte Ordering</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Receiver Phase Compensation FIFO</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
The dynamic reconfiguration controller is always enabled when a receiver exists in a transceiver-based design even if the channel will not be reconfigured. The dynamic reconfiguration controller is used for receiver offset cancellation. Because of this, the busy signal from the dynamic reconfiguration controller must always be monitored when instantiating a receiver.

For more information about reset and offset cancellation, refer to the reset waveforms and the “Offset Cancellation” section of AN 558: Implementing Dynamic Reconfiguration in Arria II Devices.

### Transceiver Reset Sequences

You can configure transceiver channels in Arria II GX and GZ devices in various configurations. In all functional modes except XAUI functional mode, transceiver channels can be either bonded or non-bonded. In XAUI functional mode, transceiver channels must be bonded. In PCI Express® (PIPE) (PCIe®) functional mode, transceiver channels can be either bonded or non-bonded and need to follow a specific reset sequence.

The two categories of reset sequences for the Arria II device family described in this chapter are:

- “All Supported Functional Modes Except PCIe Functional Mode” on page 4–6 describes the reset sequences in bonded and non-bonded configurations.

- “PCIe Functional Mode” on page 4–15 describes the reset sequence for the initialization/compliance phase and normal operation phase in PCIe functional modes.
The busy signal remains low for the first reconfig_clk clock cycle. It is then asserted from the second reconfig_clk clock cycle. Subsequent de-assertion of the busy signal indicates the completion of the offset cancellation process. This busy signal is required in transceiver reset sequences except for Transmitter Only channel configurations. Refer to the reset sequences shown in Figure 4–2 and the associated references listed in the notes.

**Figure 4–2. Transceiver Reset Sequences Chart**

**Notes to Figure 4–2:**

1. Refer to the Timing Diagram in Figure 4–10 on page 4–15.
2. Refer to the Timing Diagram in Figure 4–3 on page 4–7.
3. Refer to the Timing Diagram in Figure 4–4 on page 4–8.
4. Refer to the Timing Diagram in Figure 4–5 on page 4–9.
5. Refer to the Timing Diagram in Figure 4–6 on page 4–11.
6. Refer to the Timing Diagram in Figure 4–7 on page 4–12.
7. Refer to the Timing Diagram in Figure 4–8 on page 4–13.
8. Refer to the Timing Diagram in Figure 4–9 on page 4–14.

Altera strongly recommends adhering to these reset sequences for proper operation of the Arria II transceiver.
All Supported Functional Modes Except PCIe Functional Mode

This section describes the reset sequences for transceiver channels in bonded and non-bonded configurations. Timing diagrams of some typical configurations are shown to facilitate proper reset sequence implementation. In these functional modes, you can set the receiver CDR either in automatic lock or manual lock mode.

In manual lock mode, the receiver CDR locks to the reference clock (lock-to-reference) or the incoming serial data (lock-to-data), depending on the logic levels of the rx_locktorefclk and rx_locktodata signals. With the receiver CDR in manual lock mode, you can configure the transceiver channels in the Arria II GX or GZ device either in a non-bonded configuration or a bonded configuration. In a bonded configuration, such as XAUI mode, four channels are bonded together.

Table 4–3 lists the lock-to-reference (LTR) and lock-to-data (LTD) controller lock modes for the rx_locktorefclk and rx_locktodata signals.

Table 4–3. Lock-To-Reference and Lock-To-Data Modes for Arria II Devices

<table>
<thead>
<tr>
<th>rx_locktorefclk</th>
<th>rx_locktodata</th>
<th>LTR/LTD Controller Lock Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Manual, LTR Mode</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Manual, LTD Mode</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Automatic Lock Mode</td>
</tr>
</tbody>
</table>

Bonded Channel Configuration

In a bonded channel configuration, you can reset all the bonded channels simultaneously. Examples of bonded channel configurations are XAUI, PCIe, and Basic x4 functional modes. In Basic x4 functional mode, you can bond Transmitter Only channels together.

In XAUI mode, the receiver and transmitter channels are bonded. Each of the receiver channels in this mode has its own output status signals, rx_pll_locked and rx_freqlocked. The timing of these signals is considered in the reset sequence.

The following timing diagrams describe the reset and power-down sequences for bonded configurations under the following setups:

- **Transmitter Only** channel setup—applicable to Basic x4 functional mode
- **Receiver and Transmitter** channel setup—receiver CDR in automatic lock mode; applicable to XAUI functional mode
- **Receiver and Transmitter** channel setup—receiver CDR in manual lock mode; applicable to XAUI functional mode
Transmitter Only Channel

This configuration contains only a transmitter channel. If you create a Transmitter Only instance in the ALTGX MegaWizard Plug-In Manager in Basic x4 functional mode, use the reset sequence shown in Figure 4–3.

Figure 4–3. Sample Reset Sequence for Four Transmitter Only Channels

As shown in Figure 4–3, perform the following reset sequence steps for the Transmitter Only channel configuration:

1. After power up, assert `pll_powerdown` for a minimum period of 1 μs (the time between markers 1 and 2).

2. Keep the `tx_digitalreset` signal asserted during this time period. After you de-assert the `pll_powerdown` signal, the transmitter PLL starts locking to the transmitter input reference clock.

3. After the transmitter PLL locks, as indicated by the `pll_locked` signal going high (marker 3), de-assert the `tx_digitalreset` signal (marker 4). The transmitter is ready to transmit data.
Receiver and Transmitter Channel—Receiver CDR in Automatic Lock Mode

This configuration contains both a transmitter and receiver channel. For XAUI functional mode, with the receiver CDR in automatic lock mode, use the reset sequence shown in Figure 4–4.

Figure 4–4. Sample Reset Sequence for Four Receiver and Transmitter Channels—Receiver CDR in Automatic Lock Mode

As shown in Figure 4–4, perform the following reset sequence steps for the receiver CDR in automatic lock mode configuration:

1. After power up, assert \( \text{pll\_powerdown} \) for a minimum period of 1 \( \mu \text{s} \) (the time between markers 1 and 2).

2. Keep the \( \text{tx\_digitalreset} \), \( \text{rx\_analogreset} \), and \( \text{rx\_digitalreset} \) signals asserted during this time period. After you de-assert the \( \text{pll\_powerdown} \) signal, the transmitter PLL starts locking to the transmitter input reference clock.

3. After the transmitter PLL locks, as indicated by the \( \text{pll\_locked} \) signal going high, de-assert the \( \text{tx\_digitalreset} \) signal. At this point, the transmitter is ready for data traffic.

4. For the receiver operation, after de-assertion of the \( \text{busy} \) signal, wait for two parallel clock cycles to de-assert the \( \text{rx\_analogreset} \) signal. After \( \text{rx\_analogreset} \) is de-asserted, the receiver CDR of each channel starts locking to the receiver input reference clock.
5. Wait for the `rx_freqlocked` signal from each channel to go high. The `rx_freqlocked` signal of each channel may go high at different times (indicated by the slashed pattern at marker 7).

6. In a bonded channel group, when the `rx_freqlocked` signals of all the channels have gone high, from that point onwards, wait for at least 4 μs for the receiver parallel clock to be stable, then de-assert the `rx_digitalreset` signal (marker 8). At this point, all the receivers are ready for data traffic.

**Receiver and Transmitter Channel—Receiver CDR in Manual Lock Mode**

This configuration contains both a transmitter and receiver channel. For XAUI functional mode, with the receiver CDR in manual lock mode, use the reset sequence shown in **Figure 4–5**.

**Figure 4–5. Sample Reset Sequence of Four Receiver and Transmitter Channels—Receiver CDR in Manual Lock Mode**

<table>
<thead>
<tr>
<th>Reset Signals</th>
<th>1 μs (1)</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>pll_powerdown</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tx_digitalreset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rx_analogreset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rx_digitalreset</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CDR Control Signals**

- `rx_locktorefclk[0]` 8
- `rx_locktorefclk[3]` 8
- `rx_locktodata[0]` 8
- `rx_locktodata[3]` 8

**Output Status Signals**

- `pll_locked` 2
- `busy` Two parallel clock cycles
- `rx_pll_locked[0]` 3
- `rx_pll_locked[3]` 3

**Note to Figure 4–5:**

(1) For the transceiver block power down duration, refer to the *Device Datasheet for Arria II Devices* chapter.
As shown in Figure 4–5, perform the following reset sequence steps for the receiver CDR in manual lock mode configuration:

1. After power up, assert `pll_powerdown` for a minimum period of 1 \( \mu s \) (the time between markers 1 and 2).

2. Keep the `tx_digitalreset`, `rx_analogreset`, `rx_digitalreset`, and `rx_locktorefclk` signals asserted and the `rx_locktodata` signal de-asserted during this time period. After you de-assert the `pll_powerdown` signal, the transmitter PLL starts locking to the transmitter input reference clock.

3. After the transmitter PLL locks, as indicated by the `pll_locked` signal going high (marker 3), de-assert the `tx_digitalreset` signal (marker 4). For the receiver operation, after de-assertion of the `busy` signal, wait for two parallel clock cycles to de-assert the `rx_analogreset` signal. After the `rx_analogreset` signal is de-asserted, the receiver CDR of each channel starts locking to the receiver input reference clock because `rx_locktorefclk` is asserted.

4. Wait for the `rx_pll_locked` signal from each channel to go high. The `rx_pll_locked` signal of each channel may go high at different times with respect to each other (indicated by the slashed pattern at the marker 7).

5. In a bonded channel group, when the last `rx_pll_locked` signal goes high, from that point onwards, wait at least 15 \( \mu s \) and then de-assert `rx_locktorefclk` and assert `rx_locktodata` (marker 8). At this point, the receiver CDR enters lock-to-data mode and the receiver PLL starts locking to the received data.

6. De-assert `rx_digitalreset` at least 4 \( \mu s \) (the time between markers 8 and 9) after asserting the `rx_locktodata` signal.

**Non-Bonded Channel Configuration**

In non-bonded channels, each channel in the ALTGX megafunction instance contains its own `tx_digitalreset`, `rx_analogreset`, `rx_digitalreset`, `rx_pll_locked`, and `rx_freqlocked` signals.

You can reset each channel independently. For example, if there are four non-bonded channels, the ALTGX MegaWizard Plug-In Manager provides five signals: `tx_digitalreset`, `rx_analogreset`, `rx_digitalreset`, `rx_pll_locked`, and `rx_freqlocked`.

The following timing diagrams describe the reset and power-down sequences for one channel in a non-bonded configuration, under five different setups:

- **Transmitter Only** channel setup
- **Receiver Only** channel setup—receiver CDR in automatic lock mode
- **Receiver Only** channel setup—receiver CDR in manual lock mode
- **Receiver and Transmitter** channel setup—receiver CDR in automatic lock mode
- **Receiver and Transmitter** channel setup—receiver CDR in manual lock mode

Follow the same reset sequence for all the other channels in the non-bonded configuration.
**Transmitter Only Channel**

This configuration contains only a transmitter channel. If you create a Transmitter Only instance in the ALTGX MegaWizard Plug-In Manager, use the same reset sequence as shown in Figure 4–2 on page 4–5.

**Receiver Only Channel—Receiver CDR in Automatic Lock Mode**

This configuration contains only a receiver channel. If you create a Receiver Only instance in the ALTGX MegaWizard Plug-In Manager with the receiver CDR in automatic lock mode, use the reset sequence shown in Figure 4–6.

---

**Figure 4–6. Sample Reset Sequence of Receiver-Only Channel—Receiver CDR in Automatic Lock Mode**

As shown in Figure 4–6, perform the following reset sequence steps for the receiver CDR in automatic lock mode configuration:

1. After power up, wait for the busy signal to be de-asserted (marker 1).
2. De-assert the rx_analogreset signal (marker 2).
3. Keep the rx_digitalreset signal asserted during this time period. After you de-assert the rx_analogreset signal, the receiver PLL starts locking to the receiver input reference clock.
4. Wait for the rx_freqlocked signal to go high (marker 3).
5. After rx_freqlocked goes high, wait at least 4 µs and then de-assert the rx_digitalreset signal (marker 4). At this point, the receiver is ready to receive data.
Receiver Only Channel—Receiver CDR in Manual Lock Mode

This configuration contains only a receiver channel. If you create a Receiver Only instance in the ALTGX MegaWizard Plug-In Manager with the receiver CDR in manual lock mode, use the reset sequence shown in Figure 4–7.

Figure 4–7. Sample Reset Sequence of Receiver-Only Channel—Receiver CDR in Manual Lock Mode

As shown in Figure 4–7, perform the following reset sequence steps for the receiver CDR in manual lock mode:

1. After power up, assert rx_analogreset for a minimum period of two parallel clock cycles (the time between markers 1 and 2).

2. Keep the rx_digitalreset and rx_locktorefclk signals asserted and the rx_locktodata signal de-asserted during this time period.

3. After de-assertion of the busy signal, de-assert the rx_analogreset signal, after which the receiver CDR starts locking to the receiver input reference clock because the rx_locktorefclk signal is asserted.

4. Wait at least 15 μs (the time between markers 3 and 4) after the rx_pll_locked signal goes high, and then de-assert the rx_locktorefclk signal. At the same time, assert the rx_locktodata signal (marker 4). At this point, the receiver CDR enters lock-to-data mode and the receiver PLL starts locking to the received data.

5. De-assert rx_digitalreset at least 4 μs (the time between markers 4 and 5) after asserting the rx_locktodata signal.
Receiver and Transmitter Channel—Receiver CDR in Automatic Lock Mode

This configuration contains both a transmitter and receiver channel. If you create a Receiver and Transmitter instance in the ALTGX MegaWizard Plug-In Manager with the receiver CDR in automatic lock mode, use the reset sequence shown in Figure 4–8.

Figure 4–8. Sample Reset Sequence of Receiver and Transmitter Channel—Receiver CDR in Automatic Lock Mode

As shown in Figure 4–8, perform the following reset sequence steps for the receiver CDR in automatic lock mode:

1. After power up, assert $\text{pll\_powerdown}$ for a minimum period of 1 $\mu s$ (the time between markers 1 and 2).

2. Keep the $\text{tx\_digitalreset}$, $\text{rx\_analogreset}$, and $\text{rx\_digitalreset}$ signals asserted during this time period. After you de-assert the $\text{pll\_powerdown}$ signal, the transmitter PLL starts locking to the transmitter input reference clock.

3. After the transmitter PLL locks, as indicated by the $\text{pll\_locked}$ signal going high (marker 3), de-assert $\text{tx\_digitalreset}$. For receiver operation, wait for the $\text{busy}$ signal to be de-asserted, after which $\text{rx\_analogreset}$ is de-asserted. After you de-assert $\text{rx\_analogreset}$, the receiver CDR starts locking to the receiver input reference clock.

4. Wait for the $\text{rx\_freqlocked}$ signal to go high (marker 7).

5. After the $\text{rx\_freqlocked}$ signal goes high, wait at least 4 $\mu s$ and then de-assert the $\text{rx\_digitalreset}$ signal (marker 8). The transmitter and receiver are ready for data traffic.
Receiver and Transmitter Channel—Receiver CDR in Manual Lock Mode

This configuration contains both a transmitter and receiver channel. If you create a Receiver and Transmitter instance in the ALTGX MegaWizard Plug-In Manager with the receiver CDR in manual lock mode, use the reset sequence shown in Figure 4–9.

Figure 4–9. Sample Reset Sequence of Receiver and Transmitter Channel—Receiver CDR in Manual Lock Mode

As shown in Figure 4–9, perform the following reset sequence steps for the receiver in manual lock mode:

1. After power up, assert \texttt{pll\_powerdown} for a minimum period of 1 \(\mu\)s (the time between markers 1 and 2).

2. Keep the \texttt{tx\_digitalreset}, \texttt{rx\_analogreset}, \texttt{rx\_digitalreset}, and \texttt{rx\_locktorefclk} signals asserted and the \texttt{rx\_locktodata} signal de-asserted during this time period. After you de-assert the \texttt{pll\_powerdown} signal, the transmitter PLL starts locking to the transmitter input reference clock.

3. After the transmitter PLL locks, as indicated by the \texttt{pll\_locked} signal going high (marker 3), de-assert \texttt{tx\_digitalreset}. For receiver operation, wait for the \texttt{busy} signal to be de-asserted. At this point, \texttt{rx\_analogreset} is de-asserted. After \texttt{rx\_analogreset} is de-asserted, the receiver CDR starts locking to the receiver input reference clock because \texttt{rx\_locktorefclk} is asserted.
4. Wait for at least 15 μs (the time between markers 7 and 8) after the `rx_pll_locked` signal goes high, then de-assert the `rx_locktorefclk` signal. At the same time, assert the `rx_locktodata` signal (marker 8). At this point, the receiver CDR enters lock-to-data mode and the receiver CDR starts locking to the received data.

5. De-assert `rx_digitalreset` at least 4 μs (the time between markers 8 and 9) after asserting the `rx_locktodata` signal.

**PCIe Functional Mode**

You can configure PCIe functional mode with or without the receiver clock rate compensation FIFO in the Arria II device family. The reset sequence remains the same of whether or not you use the receiver clock rate compensation FIFO.

**PCIe Reset Sequence**

PCIe protocol consists of the initialization/compliance phase and normal operation phase. The reset sequences for these two phases are based on the timing diagram shown in Figure 4–10.

**Figure 4–10. Reset Sequence of PCIe Functional Mode**

**Notes to Figure 4–10:**

1. The minimum T1 and T2 period is 4 μs.
2. The minimum T3 period is two parallel clock cycles.
**PCIe Initialization/Compliance Phase**

After the device is powered up, a PCIe-compliant device goes through the compliance phase during initialization. The \( rx\_digital\_reset \) signal must be de-asserted during this compliance phase to achieve transitions on the \( pipe\_phy\_done\_status \) signal, as expected by the link layer.

The \( rx\_digital\_reset \) signal is de-asserted based on the assertion of the \( rx\_freq\_locked \) signal.

During the initialization/compliance phase, do not use the \( rx\_freq\_locked \) signal to trigger a de-assertion of the \( rx\_digital\_reset \) signal. Instead, perform the following reset sequence as shown in Figure 4–10:

1. After power up, assert \( pll\_power\_down \) for a minimum period of 1 \( \mu s \) (the time between markers 1 and 2). Keep the \( tx\_digital\_reset \), \( rx\_analog\_reset \), and \( rx\_digital\_reset \) signals asserted during this time period. After you de-assert the \( pll\_power\_down \) signal, the transmitter PLL starts locking to the transmitter input reference clock.
2. After the transmitter PLL locks, as indicated by the \( pll\_locked \) signal going high (marker 3), de-assert \( tx\_digital\_reset \). For receiver operation, wait for the \( busy \) signal to be de-asserted and for \( rx\_analog\_reset \) to be de-asserted. After \( rx\_analog\_reset \) is de-asserted, the receiver CDR starts locking to the receiver input reference clock.
3. When the receiver CDR locks to the input reference clock, as indicated by the \( rx\_pll\_locked \) signal going high at marker 7, de-assert the \( rx\_digital\_reset \) signal (marker 8). After de-asserting \( rx\_digital\_reset \), the \( pipe\_phy\_done\_status \) signal transitions from the transceiver channel to indicate the status to the link layer. Depending on its status, \( pipe\_phy\_done\_status \) helps with the continuation of the compliance phase. After successful completion of this phase, the device enters into the normal operation phase.

**PCIe Normal Phase**

For the normal operation phase, perform the following reset sequence, as shown in Figure 4–10:

1. After completion of the Initialization/Compliance phase, when the \( rx\_freq\_locked \) signal is de-asserted, (marker 10), wait for the \( rx\_pll\_locked \) signal assertion signifying the lock-to-reference clock.
2. Wait for the \( rx\_freq\_locked \) signal to go high again. In this phase, the received data is valid (not electrical idle) and the receiver CDR locks to the incoming data.
3. Proceed with the reset sequence after assertion of the \( rx\_freq\_locked \) signal.
4. After the \( rx\_freq\_locked \) signal goes high, wait for at least 4 \( \mu s \) before asserting \( rx\_digital\_reset \) (marker 12) for two parallel receive clock cycles so that the receiver phase compensation FIFO is initialized.

Data from the transceiver block is not valid from the time the \( rx\_freq\_locked \) signal goes low (marker 10) to the time \( rx\_digital\_reset \) is de-asserted (marker 13). The PLD logic ignores the data during this period (between markers 10 and 13).
You can configure the Arria II GX or GZ device in x1, x2, x4, and x8 PIPE lane configurations. The reset sequence described in “PCIe Reset Sequence” on page 4–15 applies to all these multi-lane configurations.

Dynamic Reconfiguration Reset Sequences

When using dynamic reconfiguration in data rate divisions in TX or Channel and TX CMU PLL select/reconfig modes, use the following reset sequences.

Reset Sequence with Data Rate Division in the TX Option

Use the example reset sequence shown in Figure 4–11 when you are using the dynamic reconfiguration controller to change the data rate of the transceiver channel. In this example, dynamic reconfiguration is used to dynamically reconfigure the data rate of the transceiver channel configured in Basic x1 mode with the receiver CDR in automatic lock mode.

Figure 4–11. Reset Sequence in Basic x1 Mode with the Receiver CDR in Automatic Lock Mode (TX Option)

Reset and Control Signals

- **tx_digitalreset**: 1
- **rate_switch_ctrl[1:0]**: New Value
- **write_all**: 1

Output Status Signals

- **busy**: 2, 3

As shown in Figure 4–11, perform the following reset procedure when using the dynamic reconfiguration controller to change the configuration of the transmitter channel:

1. After power up and properly establishing that the transmitter is operating correctly, write the new value for the data rate in the appropriate register (in this example, rate_switch_ctrl[1:0]) and subsequently assert the write_all signal (marker 1) to initiate the dynamic reconfiguration.

   For more information, refer to AN 558: Implementing Dynamic Reconfiguration in Arria II Devices.

2. Assert the tx_digitalreset signal.

3. As soon as write_all is asserted, the dynamic reconfiguration controller starts to execute its operation, as indicated by the assertion of the busy signal (marker 2).

4. After the completion of dynamic reconfiguration, the busy signal is de-asserted (marker 3).

5. Finally, tx_digitalreset can be de-asserted to continue with the transmitter operation (marker 4).
Reset Sequence with the Channel and TX PLL Select/Reconfig Option

Use the example reset sequence shown in Figure 4–12 when you are using the dynamic reconfiguration controller to change the TX PLL settings of the transceiver channel. In this example, the dynamic reconfiguration is used to dynamically reconfigure the data rate of the transceiver channel configured in Basic x1 mode with the receiver CDR in automatic lock mode.

Figure 4–12. Reset Sequence in Basic x1 Mode with Receiver CDR in Automatic Lock Mode (Channel and TX PLL select/reconfig Option)

As shown in Figure 4–12, perform the following reset procedure when using the dynamic reconfiguration controller to change the configuration of the transceiver channel:

1. After power up and establishing that the transceiver is operating correctly, write the new value in the appropriate registers (including reconfig_mode_sel[2:0]) and subsequently assert the write_all signal (marker 1) to initiate the dynamic reconfiguration.

   For more information, refer to AN 558: Implementing Dynamic Reconfiguration in Arria II Devices.

2. Assert the tx_digitalreset, rx_analogreset, and rx_digitalreset signals.

3. As soon as write_all is asserted, the dynamic reconfiguration controller starts to execute its operation, as indicated by the assertion of the busy signal (marker 2).

4. Wait for the assertion of the channel_reconfig_done signal (marker 4), which indicates the completion of dynamic reconfiguration in this mode.
5. After assertion of the channel_reconfig_done signal, de-assert tx_digitalreset (marker 5) and wait for at least five parallel clock cycles to de-assert the rx_analogreset signal (marker 6).

6. Finally, wait for the rx_freqlocked signal to go high. After rx_freqlocked goes high (marker 7), wait for 4 µs to de-assert the rx_digitalreset signal (marker 8). At this point, the receiver is ready for data traffic.

**Hot Socketing Reset Sequence**

Use the example hot socketing reset sequence shown in Figure 4–13 when you are using hot socketing.

**Figure 4–13. Reset Sequence for Hot Socketing**

As shown in Figure 4–13, perform the following reset procedure when using hot socketing to change the configuration of the transceiver channel:

1. After power up, assert pll_powerdown for a minimum period of 1 µs.

2. When busy is asserted, the RX signal detect block is enabled.

3. When busy is asserted, rx_signaldetect is asserted if the RX input is valid or de-asserted if the RX input is invalid.

4. If rx_freqlocked is asserted when rx_signaldetect is still low, keep rx_digitalreset asserted.

5. Wait until rx_signaldetect is asserted, then re-assert rx_analogreset for at least two parallel clock cycle.

6. De-assert rx_digitalreset when rx_freqlocked and rx_signaldetect are asserted for ≥ 4 µs.

7. If the RX input is invalid, rx_signaldetect is de-asserted.
8. When $rx_{\text{signal detect}}$ de-asserts, assert $rx_{\text{digital reset}}$ to avoid RX from receiving corrupted data.

9. Monitor $rx_{\text{signal detect}}$ until $rx_{\text{signal detect}}$ is asserted.

10. If the RX input is not floating or the far end TX input is not in electrical idle, $rx_{\text{signal detect}}$ is asserted when the RX input is valid.

11. When $rx_{\text{signal detect}}$ is asserted, assert the $rx_{\text{analog reset}}$ for two parallel clock cycles.

12. The $rx_{\text{freq locked}}$ and $rx_{\text{signal detect}}$ is de-asserted when $rx_{\text{analog reset}}$ is asserted.

For other functional modes that do not have the $rx_{\text{signal detect}}$ output port but require hot socketing, you have several options available to determine the link status. You can logically AND together any combinations of these options and use them alternatively as a loss-of-link status indicator.

- Option 1: Create or use the Receiver Monitor Block to observe the rx word alignment status and monitor the received data at the upper layer. Repeat step 4 of the “Hot Socketing Reset Sequence” on page 4–19 if the receiver loses word alignment synchronization or detects an error in the received data.

- Option 2: Implement a Digital Loss of Signal detector by enabling the Run Length Violation option in the Word Aligner to detect bit transitions in a predetermined sliding window length.

- Option 3: Implement a parts per million (PPM) detector in the device core to detect loss-of-link status.

- Option 4: Use the Receiver Phase Compensation FIFO overflow/underflow status port to detect loss-of-link status.

**Power Down**

The Quartus II software automatically selects the power down channel feature, which takes effect when you configure the Arria II GX or GZ device. All unused transceiver channels and blocks are powered down to reduce overall power consumption.

The $gxb\_powerdown$ signal is an optional transceiver block signal. It powers down all the blocks in the transceiver block. The minimum pulse width for this signal is $1 \mu s$. 
After power up, if you use the `gxb_powerdown` signal, wait for de-assertion of the `busy` signal, then assert the `gxb_powerdown` signal for a minimum of 1 µs. To finish, follow the sequence shown in Figure 4–14.

**Simulation Requirements**

The following are simulation requirements:

- The `gxb_powerdown` port is optional. In simulation, if the `gxb_powerdown` port is not instantiated, you must assert the `tx_digitalreset`, `rx_digitalreset`, and `rx_analogreset` signals appropriately for correct simulation behavior.

- If the `gxb_powerdown` port is instantiated, and the other reset signals are not used, you must assert the `gxb_powerdown` signal for at least one parallel clock cycle for correct simulation behavior.

- You can de-assert the `rx_digitalreset` signal immediately after the `rx_freqlocked` signal goes high to reduce the simulation run time. It is not necessary to wait 4 µs (as suggested in the actual reset sequence).

- The `busy` signal is de-asserted after approximately 20 parallel `reconfig_clk` clock cycles in order to reduce the simulation run time. For silicon behavior in the hardware, follow the reset sequences described in this chapter.

- In PCIe mode simulation, you must assert the `tx_forceelecide` signal for at least one parallel clock cycle before transmitting normal data for correct simulation behavior.
Document Revision History

Table 4–4 lists the revision history for this chapter.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>July 2013</td>
<td>3.2</td>
<td>Updated Figure 4–13.</td>
</tr>
<tr>
<td>June 2011</td>
<td>3.1</td>
<td>■ Updated Table 4–1 and Table 4–2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added “Hot Socketing Reset Sequence” section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Minor text edits.</td>
</tr>
<tr>
<td>December 2010</td>
<td>3.0</td>
<td>■ Updated to add Arria II GZ information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Minor text edits.</td>
</tr>
<tr>
<td>July 2010</td>
<td>2.0</td>
<td>■ Updated Figure 4–4, Figure 4–5, and Figure 4–12.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ updated the “Blocks Affected by Reset and Power-Down Signals” section.</td>
</tr>
<tr>
<td>March 2009</td>
<td>1.1</td>
<td>Added the “Dynamic Reconfiguration Reset Sequences” section.</td>
</tr>
<tr>
<td>February 2009</td>
<td>1.0</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>
This chapter provides additional information about the *Arria II Device Handbook* and Altera.

**About this Handbook**

This handbook provides comprehensive information about the Altera® Arria® II GX family of devices.

**How to Contact Altera**

To locate the most up-to-date information about Altera products, refer to the following table.

<table>
<thead>
<tr>
<th>Contact (1)</th>
<th>Contact Method</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical support</td>
<td>Website</td>
<td><a href="http://www.altera.com/support">www.altera.com/support</a></td>
</tr>
<tr>
<td>Technical training</td>
<td>Website</td>
<td><a href="http://www.altera.com/training">www.altera.com/training</a></td>
</tr>
<tr>
<td></td>
<td>Email</td>
<td><a href="mailto:custrain@altera.com">custrain@altera.com</a></td>
</tr>
<tr>
<td>Product literature</td>
<td>Website</td>
<td><a href="http://www.altera.com/literature">www.altera.com/literature</a></td>
</tr>
<tr>
<td>Non-technical support (General)</td>
<td>Email</td>
<td><a href="mailto:nacomp@altera.com">nacomp@altera.com</a></td>
</tr>
<tr>
<td>(Software Licensing)</td>
<td>Email</td>
<td><a href="mailto:authorization@altera.com">authorization@altera.com</a></td>
</tr>
</tbody>
</table>

**Note to Table:**
(1) You can also contact your local Altera sales office or sales representative.

**Typographic Conventions**

The following table shows the typographic conventions this document uses.

<table>
<thead>
<tr>
<th>Visual Cue</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold Type with Initial Capital Letters</strong></td>
<td>Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, Save As dialog box. For GUI elements, capitalization matches the GUI.</td>
</tr>
<tr>
<td><strong>bold type</strong></td>
<td>Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, \qdesigns directory, D: drive, and chiptrip.pdf file.</td>
</tr>
<tr>
<td><strong>Italic Type with Initial Capital Letters</strong></td>
<td>Indicate document titles. For example, Stratix IV Design Guidelines.</td>
</tr>
<tr>
<td><strong>italic type</strong></td>
<td>Indicates variables. For example, ( n + 1 ). Variable names are enclosed in angle brackets (&lt; &gt;). For example, &lt;file name&gt;&gt; and &lt;project name&gt;.pof\ file.</td>
</tr>
<tr>
<td>Initial Capital Letters</td>
<td>Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.</td>
</tr>
<tr>
<td>“Subheading Title”</td>
<td>Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, “Typographic Conventions.”</td>
</tr>
<tr>
<td>Visual Cue</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>Courier type</td>
<td>Indicates signal, port, register, bit, block, and primitive names. For example, data1, tdi, and input. The suffix n denotes an active-low signal. For example, resetn. Indicates command line commands and anything that must be typed exactly as it appears. For example, c:\qdesigns\tutorial\chiptrip.gdf. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword SUBDESIGN), and logic function names (for example, TRI).</td>
</tr>
<tr>
<td>➤</td>
<td>An angled arrow instructs you to press the Enter key.</td>
</tr>
<tr>
<td>1., 2., 3., and a., b., c., and so on</td>
<td>Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.</td>
</tr>
<tr>
<td>■ ■ ■</td>
<td>Bullets indicate a list of items when the sequence of the items is not important.</td>
</tr>
<tr>
<td>❔</td>
<td>The hand points to information that requires special attention.</td>
</tr>
<tr>
<td>❔</td>
<td>A question mark directs you to a software help system with related information.</td>
</tr>
<tr>
<td>❔</td>
<td>The feet direct you to another document or website with related information.</td>
</tr>
<tr>
<td>CAUTION</td>
<td>A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.</td>
</tr>
<tr>
<td>WARNING</td>
<td>A warning calls attention to a condition or possible situation that can cause you injury.</td>
</tr>
<tr>
<td>💌</td>
<td>The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.</td>
</tr>
</tbody>
</table>