Rocket I/O™ Transceiver User Guide

XILINX ®

Rocket I/O™ Transceiver User Guide

UG024 (v1.2) February 25, 2002

The following table shows the revision history for this document.

Date	Version	Revision
11/20/01	1.0	Initial Xilinx release.
01/23/02	1.1	Updated for typographical and other errors found during review.
02/25/02	1.2	Part of Virtex-II Pro™ Developer's Kit (March 2002 Release)

Contents

Chapter 4: Analog Design Considerations

Chapter 5: Simulation and Implementation

Appendix A: Rocket I/O™ Cell Models

Figures

Chapter 1: Introduction

Chapter 2: Rocket I/O™ Transceiver Overview

Chapter 3: Digital Design Considerations

Chapter 4: Analog Design Considerations

Chapter 5: Simulation and Implementation

Appendix A: Rocket I/O™ Cell Models

Tables

Chapter 1: Introduction

Chapter 2: Rocket I/O™ Transceiver Overview

Chapter 3: Digital Design Considerations

Chapter 4: Analog Design Considerations

Chapter 5: Simulation and Implementation

Appendix A: Rocket I/O™ Cell Models

![XILINX®]

Chapter 1

Introduction

> **IMPORTANT NOTE:**
>
> This document assumes use of **ISE v4.2.x**. If running ISE v4.1.x, the following modifications must be made:
>
> 1. *Remove* the port ENMCOMMAALIGN and replace its function by *adding* the attribute MCOMMA_ALIGN.
>
> 2. *Remove* the port ENPCOMMAALIGN and replace its function by *adding* the attribute PCOMMA_ALIGN.
>
> 3. Where a High is indicated for a removed port, set the corresponding attribute to TRUE; where a Low is indicated, set the corresponding attribute to FALSE.

Rocket I/O Features

The Rocket I/O™ transceiver's flexible, programmable features allow a multi-gigabit serial transceiver to be easily integrated into any Virtex-II Pro design:

- Variable speed full-duplex transceiver, allowing 500 Mb/s to 3.125 Gb/s baud transfer rates
- Monolithic clock synthesis and clock recovery system, eliminating the need for external components
- Automatic lock-to-reference function
- Serial output differential swing can be programmed at five levels from 800 mV to 1600 mV (peak-peak), allowing compatibility with other serial system voltage levels.
- Four levels of programmable pre-emphasis
- AC and DC coupling
- Programmable on-chip termination of 50Ω or 75Ω (eliminating the need for external termination resistors)
- Serial and parallel TX to RX internal loopback modes for testing operability
- Programmable comma detection to allow for any protocol and detection of any 10-bit character.

In This User Guide

The Rocket I/O *Transceiver User Guide* contains these sections:

- Chapter 1, **Introduction** — This chapter.
- Chapter 2, **Rocket I/O™ Transceiver Overview** — An overview of the transceiver's capabilities and how it works.

- Chapter 3, **Digital Design Considerations** — Ports and attributes for the six provided communications protocol primitives; VHDL/Verilog code examples for clocking and reset schemes; transceiver instantiation; 8B/10B encoding; CRC; channel bonding.
- Chapter 4, **Analog Design Considerations** — Rocket I/O serial overview; pre-emphasis; jitter; clock/data recovery; PCB design requirements.
- Chapter 5, **Simulation and Implementation** — Simulation models; implementation tools; debugging and diagnostics.
- Appendix A, **Rocket I/O™ Cell Models** — Verilog module declarations associated with each of the sixteen Rocket I/O communication standard implementations.

Naming Conventions

Input and output ports of the Rocket I/O transceiver primitives are denoted in upper-case letters. Attributes of the Rocket I/O transceiver are denoted in upper-case letters with underscores. Trailing numbers in primitive names denote the byte width of the data path. These values are preset and not modifiable. When assumed to be the same frequency, RXUSRCLK and TXUSRCLK are referred to as USRCLK and can be used interchangeably. This also holds true for RXUSRCLK2, TXUSRCLK2, and USRCLK2.

Comma Definition

A comma is a "K" character used by the transceiver to align the serial data on a byte/half-word boundary (depending on the protocol used), so that the serial data is correctly decoded into parallel data.

For More Information

For a complete menu of online information resources available on the Xilinx website, visit **http://www.xilinx.com/virtex2pro/**.

For a comprehensive listing of available tutorials and resources on network technologies and communications protocols, visit **http://www.iol.unh.edu/training/**.

Further Reading

The Virtex-II Pro™ Developer's Kit contains a wealth of valuable information that will assist you in your design efforts. The documentation contained within the eight volumes is organized to assist you in quickly finding relevant materials. To obtain the most recent revision of this documentation, please see **http://support.xilinx.com/xlnx/xil_tt_product.jsp?sProduct=Virtex-II+Pro**.

Documentation Provided by Xilinx

Virtex-II Pro™ Platform FPGA Handbook

Virtex-II Pro™ Platform FPGA Developer's Kit Documentation:

Volume 1, *Advance Product Specification*

Volume 2, *PowerPC® 405 Processor*

Volume 3, *Rocket I/O™ Transceiver User Guide*

Volume 4, *Design Flow*

Volume 5, *Software Development Tools*

Volume 6, *Reference Systems*

Volume 7, *Hardware IP Specifications*

Volume 8, *Software IP and Applications*

IBM® CoreConnect™ Documentation

The Virtex-II Pro Developer's Kit integrates seamlessly with the IBM CoreConnect Toolkit. This toolkit is not included with the Developer 's Kit, but is required if bus functional simulation is desired. The toolkit provides a number of features which enhance design productivity and allow you to get the most from the Developer's Kit. To obtain the toolkit, you must be a licensee of the IBM CoreConnect Bus Architecture. Licensing CoreConnect provides access to a wealth of documentation, Bus Functional Models, Hardware IP, and the toolkit.

Xilinx provides a Web-based licensing mechanism that allows you to obtain the CoreConnect toolkit from our website. To license CoreConnect, use an Internet browser to access **http://www.xilinx.com/ipcenter/processor_central/register_coreconnect.htm**. Once your request has been approved (typically within 24 hours), you will receive an e-mail granting access to a protected website. You may then download the toolkit.

If you prefer, you can also license CoreConnect directly from IBM.

If you would like further information on CoreConnect Bus Architecture, please see IBM's CoreConnect website at **http://www.ibm.com/chips/products/coreconnect**.

Once you have licensed the CoreConnect toolkit, and installed it with the Developer's Kit, the following documents will be available to you in the following locations:

IBM CoreConnect Bus Architecture Specifications

IBM CoreConnect Processor Local Bus (PLB) Architecture Specification
see **$CORECONNECT/published/corecon/64bitPlbBus.pdf**

IBM CoreConnect On-chip Peripheral Bus (OPB) Architecture Specification
see **$CORECONNECT/published/corecon/OpbBus.pdf**

IBM CoreConnect Device Control Register (DCR) Bus Architecture Specification
see **$CORECONNECT/published/corecon/DcrBus.pdf**

IBM CoreConnect Toolkit Documentation

PLB Bus Functional Model Toolkit - User's Manual
see **$CORECONNECT/published/corecon/PlbToolkit.pdf**

OPB Bus Functional Model Toolkit - User's Manual
see **$CORECONNECT/published/corecon/OpbToolkit.pdf**

DCR Bus Functional Model Toolkit - User's Manual
see **$CORECONNECT/published/corecon/DcrToolkit.pdf**

CoreConnect Test Generator - User's Manual
see **$CORECONNECT/published/corecon/ctg.pdf**

Note: $CORECONNECT is an environment variable that is created when installing the Developer's Kit or CoreConnect Toolkit.

Software Development Documentation

There are many sources of documentation available for those who wish to learn more about Software Development. It is recommended that a web search be conducted using a favorite search engine for keywords such a "PowerPC+Software+Devlopment". Alternatively, a technical bookstore should be able to provide many valuable resources. The books listed below are a very small fraction of those available.

Books About Programming in C

There are many good books about the C and C++ programming languages. A few of these are listed below:

Books about C

Kernigham Brian, Ritchie Dennis. 1988. *The C Programming Language*. Prentice Hall

Wang Paul. 1992. *An Introduction to ANSI C on Unix*. Wadsworth.

Kumar Ram, Agrawal Rakesh. 1992. *Programming in ANSI C*. West Publishing Company.

Books about C++

Stroustrup Bjarne. 1991. *The C++ Programming Language*. Addison-Wesley.

Lafore Robert. 1995. *Object-Oriented Programming in C++*. Waite Group Press.

Online Documents About Programming in C

The Internet offers plenty of documentation about how to program in C and C++. One of the best approaches to finding documentation online is to use a search engine (such as Google, **http://www.google.com**) and search on "introduction to programming in C."

Rocket I/O™ Transceiver Overview

Basic Architecture and Capabilities

The Rocket I/O transceiver is based on Mindspeed's SkyRail™ technology. Figure 2-1, page 16, depicts an overall block diagram of the transceiver. Up to 16 transceiver modules are available on a single Virtex-II Pro FPGA, depending on the part being used. Table 2-1 shows the Rocket I/O cores available by device.

Table 2-1: **Rocket I/O Cores**

Device	Rocket I/O Cores
XC2VP2	4
XC2VP4	4
XC2VP7	8
XC2VP20	8
XC2VP50	16

The transceiver module is designed to operate at any serial bit rate in the range of 500 Mb/s to 3.125 Gb/s per channel, including the specific bit rates used by the communications standards listed in Table 2-2. The serial bit rate need not be configured in the transceiver, as the operating frequency is implied by the received data, the reference clock applied, and the SERDES_10B attribute (Table 2-3, page 16).

Table 2-2: **Communications Standards Supported by Rocket I/O Transceiver**

Mode	Channels (Lanes)[1]	I/O Bit Rate (Gb/s)
Fibre Channel	1	1.06
		2.12
Gbit Ethernet	1	1.25
XAUI (10-Gbit Ethernet)	4	3.125
Infiniband	1, 4, 12	2.5
Aurora (Xilinx protocol)	1, 2, 3, 4, ...	0.5 – 3.125
Custom Mode	1, 2, 3, 4, ...	0.5 – 3.125

Notes:
1. One channel is considered to be one transceiver.

Table 2-3: **Serial Baud Rates and the SERDES_10B Attribute**

SERDES_10B	Serial Baud Rate
False	800 Mb/s – 3.125 Gb/s
True	500 Mb/s – 1.0 Gb/s
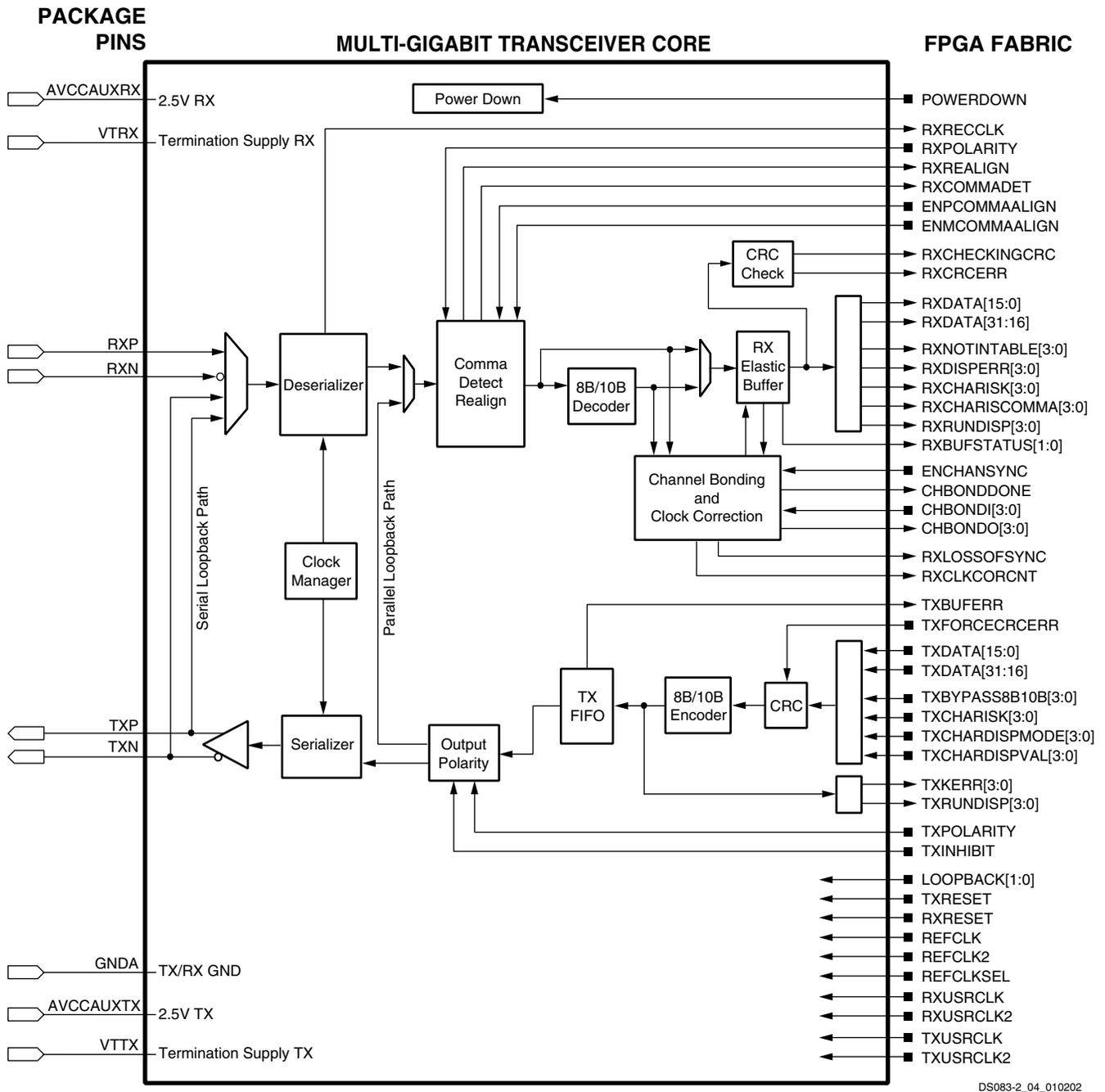
Figure 2-1: **Rocket I/O Transceiver Block Diagram**

Table 2-4 lists the 16 gigabit transceiver primitives provided. These primitives carry attributes set to default values for the communications protocols listed in Table 2-2. Data widths of one, two, and four bytes are selectable for each protocol.

Table 2-4: **Supported Rocket I/O Transceiver Primitives**

Primitives	Description	Primitive	Description
GT_CUSTOM	Fully customizable by user	GT_XAUI_2	10-Gb Ethernet, 2-byte data path
GT_FIBRE_CHAN_1	Fibre Channel, 1-byte data path	GT_XAUI_4	10-Gb Ethernet, 4-byte data path
GT_FIBRE_CHAN_2	Fibre Channel, 2-byte data path	GT_INFINIBAND_1	Infiniband, 1-byte data path
GT_FIBRE_CHAN_4	Fibre Channel, 4-byte data path	GT_INFINIBAND_2	Infiniband, 2-byte data path
GT_ETHERNET_1	Gigabit Ethernet, 1-byte data path	GT_INFINIBAND_4	Infiniband, 4-byte data path
GT_ETHERNET_2	Gigabit Ethernet, 2-byte data path	GT_AURORA_1	Xilinx protocol, 1-byte data path
GT_ETHERNET_4	Gigabit Ethernet, 4-byte data path	GT_AURORA_2	Xilinx protocol, 2-byte data path
GT_XAUI_1	10-Gb Ethernet, 1-byte data path	GT_AURORA_4	Xilinx protocol, 4-byte data path

There are two ways to modify the Rocket I/O transceiver:

•   Static properties can be set through attributes in the HDL code. Use of attributes are covered in detail in **Primitive Attributes**, page 29.

•   Dynamic changes can be made by the ports of the primitives

The Rocket I/O transceiver consists of the Physical Media Attachment (PMA) and Physical Coding Sublayer (PCS). The PMA contains the serializer/deserializer (SERDES), TX and RX buffers, clock generator, and clock recovery circuitry. The PCS contains the 8B/10B encoder/decoder and the elastic buffer supporting channel bonding and clock correction. The PCS also handles Cyclic Redundancy Check (CRC). Refer again to Figure 2-1, showing the Rocket I/O transceiver top-level block diagram and FPGA interface signals.

Clock Synthesizer

Synchronous serial data reception is facilitated by a clock/data recovery circuit. This circuit uses a fully monolithic Phase-Locked Loop (PLL), which does not require any external components. The clock/data recovery circuit extracts both phase and frequency from the incoming data stream. The recovered clock is presented on output RXRECCLK at 1/20 of the serial received data rate.

The gigabit transceiver multiplies the reference frequency provided on the reference clock input (REFCLK) by 20.

No fixed phase relationship is assumed between REFCLK, RXRECCLK, and/or any other clock that is not tied to either of these clocks. When the 4-byte or 1-byte receiver data path is used, RXUSRCLK and RXUSRCLK2 have different frequencies (1:2), and each edge of the slower clock is aligned to a falling edge of the faster clock. The same relationships apply to TXUSRCLK and TXUSRCLK2. See the section entitled **Clocking**, page 38, for details.

Clock and Data Recovery

The clock/data recovery (CDR) circuits lock to the reference clock automatically if the data is not present. For proper operation, frequency variations of REFCLK, TXUSRCLK, RXUSRCLK, and the incoming stream (RXRECCLK) must not exceed ±100 ppm.

It is critical to keep power supply noise low in order to minimize common and differential noise modes into the clock/data recovery circuitry. See **PCB Design Requirements, page 81**, for more details.

Transmitter

FPGA Transmit Interface

The FPGA can send either one, two, or four characters of data to the transmitter. Each character can be either 8 bits or 10 bits wide. If 8-bit data is applied, the additional inputs become control signals for the 8B/10B encoder. When the 8B/10B encoder is bypassed, the 10-bit character order is:

    TXCHARDISPMODE[0]
    TXCHARDISPVAL[0]
    TXDATA[7:0]

8B/10B Encoder

A bypassable 8B/10B encoder is included. The encoder uses the same 256 data characters and 12 control characters that are used for Gigabit Ethernet, XAUI, Fibre Channel, and InfiniBand.

The encoder accepts 8 bits of data along with a K-character signal for a total of 9 bits per character applied. If the K-character signal is High, the data is encoded into one of the 12 possible K-characters available in the 8B/10B code. If the K-character input is Low, the 8 bits are encoded as standard data. If the K-character input is High, and a user applies other than one of the 12 possible combinations, TXKERR indicates the error.

Disparity Control

The 8B/10B encoder is initialized with a negative running disparity.

TXRUNDISP signals the transmitter's current running disparity.

Bits TXCHARDISPMODE and TXCHARDISPVAL control the generation of running disparity before each byte, as shown in Table 2-5.

Table 2-5: **Running Disparity Control**

{txchardispmode, txchardispval}	Function
00	Maintain running disparity normally
01	Invert normally generated running disparity before encoding this byte
10	Set negative running disparity before encoding this byte
11	Set positive running disparity before encoding this byte

For example, the transceiver can generate the sequence

    K28.5+ K28.5+ K28.5– K28.5–

or

    K28.5– K28.5– K28.5+ K28.5+

by specifying inverted running disparity for the second and fourth bytes.

Transmit FIFO

Proper operation of the circuit is only possible if the FPGA clock (TXUSRCLK) is frequency-locked to the reference clock (REFCLK). Phase variations up to one clock cycle are allowable. The FIFO has a depth of four. Overflow or underflow conditions are detected and signaled at the interface.

Serializer

The multi-gigabit transceiver multiplies the reference frequency provided on the reference clock input (REFCLK) by 20. Data is converted from parallel to serial format and transmitted on the TXP and TXN differential outputs. Bit 0 is transmitted first and bit 19 is transmitted last.

The electrical polarity of TXP and TXN can be interchanged through the TXPOLARITY port. This option can either be programmed or controlled by an input at the FPGA core TX interface. This facilitates recovery from situations where printed circuit board traces have been reversed.

Transmit Termination

On-chip termination is provided at the transmitter, eliminating the need for external termination. Programmable options exist for 50Ω (default) and 75Ω termination.

Pre-emphasis Circuit and Swing Control

Four selectable levels of pre-emphasis, including default pre-emphasis, are available. Optimizing this setting allows the transceiver to drive up to 20 inches of FR4 at the maximum baud rate.

The programmable output swing control can adjust the differential output level between 400 mV and 800 mV (peak-to-peak) in four increments of 100 mV.

Receiver

Deserializer

The Rocket I/O transceiver core accepts serial differential data on its RXP and RXN inputs. The clock/data recovery circuit extracts clock phase and frequency from the incoming data stream and re-times incoming data to this clock. The recovered clock is presented on output RXRECCLK at 1/20 of the received serial data rate.

The receiver is capable of handling either transition-rich 8B/10B streams or scrambled streams, and can withstand a string of up to 75 non-transitioning bits without an error.

Word alignment is dependent on the state of comma detect bits. If comma detect is enabled, the transceiver recognizes up to two 10-bit preprogrammed characters. Upon detection of the character or characters, the comma detect output is driven High and the data is synchronously aligned. If a comma is detected and the data is aligned, no further alignment alteration takes place. If a comma is received and realignment is necessary, the data is realigned and an indication is given at the RX FPGA interface. The realignment indicator is a distinct output. The transceiver continuously monitors the data for the presence of the 10-bit character(s). Upon each occurrence of the 10-bit character, the data is checked for word alignment. If comma detect is disabled, the data is not aligned to any particular pattern. The programmable option allows a user to align data on comma+, comma–, both, or a unique user-defined and programmed sequence.

The electrical polarity of RXP and RXN can be interchanged through the RXPOLARITY port. This can be useful in the event that printed circuit board traces have been reversed.

Receiver Termination

On-chip termination is provided at the receiver, eliminating the need for external termination. The receiver includes programmable on-chip termination circuitry for 50Ω (default) or 75Ω impedance.

8B/10B Decoder

An optional 8B/10B decoder is included. A programmable option allows the decoder to be bypassed. (See **HDL Code Examples: Transceiver Bypassing of 8B/10B Encoding, page 69**.) When the 8B/10B decoder is bypassed, the 10-bit character order is:

RXCHARISK[0]
RXRUNDISP[0]
RXDATA[7:0]

The decoder uses the same table that is used for Gigabit Ethernet, Fibre Channel and InfiniBand. In addition to decoding all data and K-characters, the decoder has several extra features. The decoder separately detects both "disparity errors" and "out-of-band" errors. A *disparity error* occurs when a 10-bit character is received that exists within the 8B/10B table, but has an incorrect disparity. An *out-of-band error* occurs when a 10-bit character is recived that does not exist within the 8B/10B table. It is possible to obtain an out-of-band error without having a disparity error. The proper disparity is always computed for both legal and illegal characters. The current running disparity is available at the RXRUNDISP signal.

The 8B/10B decoder performs a unique operation if out-of-band data is detected. If out-of-band data is detected, the decoder signals the error and passes the illegal 10-bits through and places them on the outputs. This can be used for debugging purposes if desired.

The decoder also signals reception of one of the 12 valid K-characters. In addition, a programmable comma detect is included. The comma detect signal registers a comma on the receipt of any comma+, comma−, or both. Since the comma is defined as a 7-bit character, this includes several out-of-band characters. Another option allows the decoder to detect only the three defined commas (K28.1, K28.5, and K28.7) as comma+, comma−, or both. In total, there are six possible options, three for valid commas and three for "any comma".

Note that all bytes (1, 2, or 4) at the RX FPGA interface each have their own individual 8B/10B indicators (K-character, disparity error, out-of-band error, current running disparity, and comma detect).

Loopback

To facilitate testing without having the need to either apply patterns or measure data at GHz rates, two programmable loopback features are available.

One option, serial loopback, places the gigabit transceiver into a state where transmit data is directly fed back to the receiver. An important point to note is that the feedback path is at the output pads of the transmitter. This tests the entirety of the transmitter and receiver.

The second loopback path is a parallel path that checks the digital circuitry. When the parallel option is enabled, the serial loopback path is disabled. However, the transmitter outputs remain active and data is transmitted over a link. If TXINHIBIT is asserted, TXN is forced to 1 and TXP is forced to 0 until TXINHIBIT is de-asserted.

The two loopback options are shown in Table 2-6.

Table 2-6: **Loopback Options**

LOOPBACK[1:0]	Description
LOOPBACK[1]	External serial loopback
LOOPBACK[0]	Internal parallel loopback

Elastic and Transmitter Buffers

Both the transmitter and the receiver include buffers (FIFOs) in the data path. This section gives the reasons for including the buffers and outlines their operation.

Receiver Buffer

The receiver buffer is required for two reasons:

- To accommodate the slight difference in frequency between the recovered clock RXRECCLK and the internal FPGA core clock RXUSRCLK (clock correction)
- To allow realignment of the input stream to ensure proper alignment of data being read through multiple transceivers (channel bonding)

The receiver uses an *elastic buffer*, where "elastic" refers to the ability to modify the read pointer for clock correction and channel bonding.

Clock Correction

Clock RXRECCLK (the recovered clock) reflects the data rate of the incoming data. Clock RXUSRCLK defines the rate at which the FPGA core consumes the data. Ideally, these rates are identical. However, since the clocks typically have different sources, one of the clocks is faster than the other. The receiver buffer accommodates this difference between the clock rates. See Figure 2-2.

Nominally, the buffer is always half full. This is shown in the top buffer, Figure 2-2, where the shaded area represents buffered data not yet read. Received data is inserted via the write pointer under control of RXRECCLK. The FPGA core reads data via the read pointer under control of RXUSRCLK. The half full/half empty condition of the buffer gives a cushion for the differing clock rates. This operation continues indefinitely, regardless of whether or not "meaningful" data is being received. When there is no meaningful data to be received, the incoming data consists of IDLE characters or other padding.
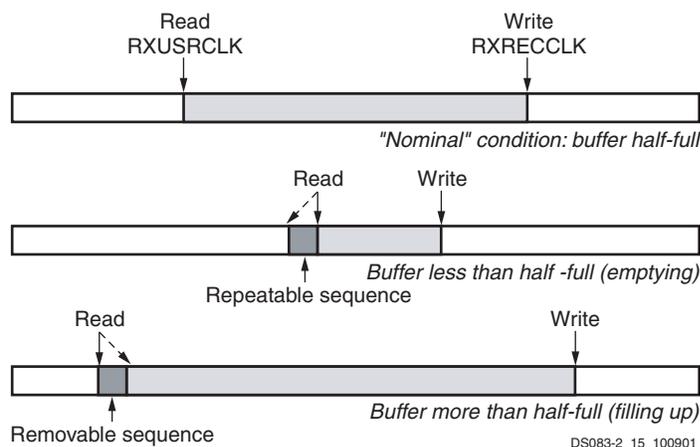
Figure 2-2: **Clock Correction in Receiver**

If RXUSRCLK is faster than RXRECCLK, the buffer becomes more empty over time. The clock correction logic corrects for this by decrementing the read pointer to reread a repeatable byte sequence. This is shown in the middle buffer, Figure 2-2, where the solid read pointer decrements to the value represented by the dashed pointer. By decrementing the read pointer instead of incrementing it in the usual fashion, the buffer is partially refilled. The transceiver inserts a single repeatable byte sequence when necessary to refill a buffer. If the byte sequence length is greater than one, and if attribute CLK_COR_REPEAT_WAIT is 0, then the transceiver can repeat the same sequence multiple times until the buffer is refilled to the half-full condition.

Similarly, if RXUSRCLK is slower than RXRECCLK, the buffer fills up over time. The clock correction logic corrects for this by incrementing the read pointer to skip over a removable byte sequence that need not appear in the final FPGA core byte stream. This is shown in the bottom buffer, Figure 2-2, where the solid read pointer increments to the value represented by the dashed pointer. This accelerates the emptying of the buffer, preventing its overflow. The transceiver design skips a single byte sequence, when necessary, to partially empty a buffer. If attribute CLK_COR_REPEAT_WAIT is 0, the transceiver can also skip two consecutive removable byte sequences in one step, to further empty the buffer, when necessary.

These operations require the clock correction logic to recognize a byte sequence that can be freely repeated or omitted in the incoming data stream. This sequence is generally an IDLE sequence, or other sequence comprised of special values that occur in the gaps separating packets of meaningful data. These gaps are required to occur sufficiently often to facilitate the timely execution of clock correction.

Channel Bonding

Some gigabit I/O standards such as Infiniband specify the use of multiple transceivers in parallel for even higher data rates. Words of data are split into bytes, with each byte sent over a separate channel (transceiver). See Figure 2-3.
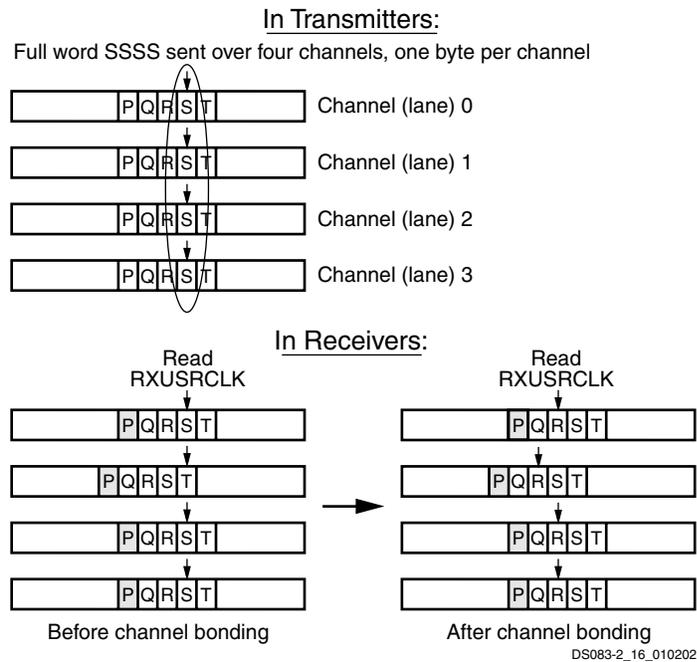
Figure 2-3: **Channel Bonding (Alignment)**

The top half of the figure shows the transmission of words split across four transceivers (channels or lanes). PPPP, QQQQ, RRRR, SSSS, and TTTT represent words sent over the four channels.

The bottom-left portion of the figure shows the initial situation in the FPGA's receivers at the other end of the four channels. Due to variations in transmission delay—especially if the channels are routed through repeaters—the FPGA core might not correctly assemble the bytes into complete words. The bottom-left illustration shows the incorrect assembly of data words PQPP, QRQQ, RSRR, etc.

To support correction of this misalignment, the data stream includes special byte sequences that define corresponding points in the several channels. In the bottom half of Figure 2-3, the shaded "P" bytes represent these special characters. Each receiver

recognizes the "P" channel bonding character, and remembers its location in the buffer. At some point, one transceiver designated as the master instructs all the transceivers to align to the channel bonding character "P" (or to some location relative to the channel bonding character). After this operation, the words transmitted to the FPGA core are properly aligned: RRRR, SSSS, TTTT, etc., as shown in the bottom-right portion of Figure 2-3. To ensure that the channels remain properly aligned following the channel bonding operation, the master transceiver must also control the clock correction operations described in the previous section for all channel-bonded transceivers.

Transmitter Buffer

The transmitter buffer's write pointer (TXUSRCLK) is frequency-locked to its read pointer (REFCLK). Therefore, clock correction and channel bonding are not required. The purpose of the transmitter's buffer is to accommodate a phase difference between TXUSRCLK and REFCLK. A simple FIFO suffices for this purpose. A FIFO depth of four permits reliable operation with simple detection of overflow or underflow, which might occur if the clocks are not frequency-locked.

CRC

The Rocket I/O transceiver CRC logic supports the 32-bit invariant CRC calculation used by Infiniband, FibreChannel, and Gigabit Ethernet.

On the transmitter side, the CRC logic recognizes where the CRC bytes should be inserted and replaces four placeholder bytes at the tail of a data packet with the computed CRC. For Gigabit Ethernet and FibreChannel, transmitter CRC can adjust certain trailing bytes to generate the required running disparity at the end of the packet.

On the receiver side, the CRC logic verifies the received CRC value, supporting the same standards as above.

The CRC logic also supports a user mode, with a simple data packet structure beginning and ending with user-defined SOP and EOP characters.

There are limitations to the CRC support provided by the Rocket I/O transceiver core:

- It is for single-channel use only. Computation and byte-striping of CRC across multiple bonded channels is not supported. For that usage, the CRC logic can be implemented in the FPGA fabric.
- The Rocket I/O transceiver does not compute the 16-bit variant CRC used for Infiniband. Therefore, Rocket I/O CRC does not fulfill the Infiniband CRC requirement. Infiniband CRC can be computed in the FPGA fabric.

Reset/Power Down

The receiver and transmitter have their own synchronous reset inputs. The transmitter reset recenters the transmission FIFO and resets all transmitter registers and the 8B/10B encoder. The receiver reset recenters the receiver elastic buffer and resets all receiver registers and the 8B/10B decoder. Neither reset signal has any effect on the PLLs.

Additional reset and power control descriptions are given in Table 2-7 and Table 2-8.

Table 2-7: **Reset and Power Control Descriptions**

Ports	Description
RXRESET	Synchronous receive system reset recenters the receiver elastic buffer, and resets the 8B/10B decoder, comma detect, channel bonding, clock correction logic, and other receiver registers. The PLL is unaffected.
TXRESET	Synchronous transmit system reset recenters the transmission FIFO, and resets the 8B/10B encoder and other transmission registers. The PLL is unaffected.
POWERDOWN	Shuts down the transceiver (both RX and TX sides) and sets TXP and TXN outputs to high-impedance state

Table 2-8: **Power Control Descriptions**

POWERDOWN	Transceiver Status
0	Transceiver in operation
1	Transceiver temporarily powered down

Chapter 3

Digital Design Considerations

List of Available Ports

The Rocket I/O transceiver primitives contain 50 ports, with the exception of the 46-port GT_ETHERNET and GT_FIBRE_CHAN primitives. The differential serial data ports (RXN, RXP, TXN, and TXP) are connected directly to external pads; the remaining 46 ports are all accessible from the FPGA logic (42 ports for GT_ETHERNET and GT_FIBRE_CHAN).

Table 3-1 contains the port descriptions of all primitives.

Table 3-1: **GT_CUSTOM[1], GT_AURORA, GT_FIBRE_CHAN[2], GT_ETHERNET[2], GT_INFINIBAND, and GT_XAUI Primitive Ports**

Port	I/O	Port Size[3]	Definition
CHBONDDONE[2]	O	1	Indicates a receiver has successfully completed channel bonding when asserted High.
CHBONDI[2]	I	4	The channel bonding control that is used only by "slaves" which is driven by a transceiver's CHBONDO port.
CHBONDO[2]	O	4	Channel bonding control that passes channel bonding and clock correction control to other transceivers.
CONFIGENABLE	I	1	Reconfiguration enable input (unused)
CONFIGIN	I	1	Data input for reconfiguring transceiver (unused)
CONFIGOUT	O	1	Data output for configuration readback (unused)
ENCHANSYNC[2]	I	1	Comes from the core to the transceiver and enables the transceiver to perform channel bonding
ENMCOMMAALIGN	I	1	Selects realignment of incoming serial bitstream on minus-comma. High realigns serial bitstream byte boundary when minus-comma is detected.
ENPCOMMAALIGN	I	1	Selects realignment of incoming serial bitstream on plus-comma. High realigns serial bitstream byte boundary when plus-comma is detected.
LOOPBACK	I	2	Selects the two loopback test modes. Bit 1 is for serial loopback and bit 0 is for internal parallel loopback.
POWERDOWN	I	1	Shuts down both the receiver and transmitter sides of the transceiver when asserted High. This decreases the power consumption while the transceiver is shut down.

Table 3-1: **GT_CUSTOM**[1]**, GT_AURORA, GT_FIBRE_CHAN**[2]**, GT_ETHERNET**[2]**, GT_INFINIBAND, and GT_XAUI Primitive Ports** *(Continued)*

Port	I/O	Port Size[3]	Definition
REFCLK	I	1	High-quality reference clock driving transmission (reading TX FIFO, and multiplied for parallel/serial conversion) and clock recovery. REFCLK frequency is accurate to ± 100 ppm. This clock originates off the device, is routed through fabric interconnect, and is selected by the REFCLKSEL.
REFCLK2	I	1	An alternative to REFCLK. Can be selected by the REFCLKSEL.
REFCLKSEL	I	1	Selects the reference clock to use REFCLK or REFCLK2. Deasserted is REFCLK. Asserted is REFCLK2.
RXBUFSTATUS	O	2	Receiver elastic buffer status. Bit 1 indicates if an overflow/underflow error has occurred when asserted High. Bit 0 indicates if the buffer is at least half-full when asserted High.
RXCHARISCOMMA	O	1, 2, 4	Similar to RXCHARISK except that the data is a comma.
RXCHARISK	O	1, 2, 4	If 8B/10B decoding is enabled, it indicates that the received data is a "K" character when asserted High. Included in Byte-mapping. If 8B/10B decoding bypassed, it becomes the 10th bit of the 10-bit encoded data.
RXCHECKINGCRC	O	1	CRC status for the receiver. Asserts High to indicate that the receiver has recognized the end of a data packet. Only meaningful if RX_CRC_USE = TRUE.
RXCLKCORCNT	O	3	Status that denotes occurrence of clock correction or channel bonding. This status is synchronized on the incoming RXDATA. See **Clock Correction Count**, page 55.
RXCOMMADET	O	1	Signals that a comma has been detected in the data stream.
RXCRCERR	O	1	Indicates if the CRC code is incorrect when asserted High. Only meaningful if RX_CRC_USE = TRUE.
RXDATA	O	8,16,32	Up to four bytes of decoded (8B/10B encoding) or encoded (8B/10B bypassed) receive data.
RXDISPERR	O	1, 2, 4	If 8B/10B encoding is enabled it indicates whether a disparity error has occurred on the serial line. Included in Byte-mapping scheme.
RXLOSSOFSYNC	O	2	Status related to byte-stream synchronization (RX_LOSS_OF_SYNC_FSM) If RX_LOSS_OF_SYNC_FSM = TRUE, this outputs the state of the FSM. Bit 1 signals a loss of sync. Bit 0 indicates a resync state. If RX_LOSS_OF_SYNC_FSM = FALSE, this indicates if received data is invalid (Bit 1) and if the channel bonding sequence is recognized (Bit 0).
RXN[4]	I	1	Serial differential port (FPGA external)
RXNOTINTABLE	O	1, 2, 4	Status of encoded data when the data is not a valid character when asserted High. Applies to the byte-mapping scheme.
RXP[4]	I	1	Serial differential port (FPGA external)
RXPOLARITY	I	1	Similar to TXPOLARITY, but for RXN and RXP. When deasserted, assumes regular polarity. When asserted, reverses polarity.

Table 3-1: **GT_CUSTOM[(1)], GT_AURORA, GT_FIBRE_CHAN[(2)], GT_ETHERNET[(2)], GT_INFINIBAND, and GT_XAUI Primitive Ports** *(Continued)*

Port	I/O	Port Size[(3)]	Definition
RXREALIGN	O	1	Signal from the PMA denoting that the byte alignment with the serial data stream changed due to a comma detection. Asserted High when alignment occurs.
RXRECCLK	O	1	Recovered clock that is divided by 20.
RXRESET	I	1	Synchronous RX system reset that "recenters" the receive elastic buffer. It also resets 8B/10B decoder, comma detect, channel bonding, clock correction logic, and other internal receive registers. It does not reset the receiver PLL.
RXRUNDISP	O	1, 2, 4	Signals the running disparity (negative/positive) in the received serial data. If 8B/10B encoding bypassed, it becomes the 9th bit of the 10-bit encoded data.
RXUSRCLK	I	1	Clock from a DCM that is used for reading the RX elastic buffer. It also clocks CHBONDI and CHBONDO in and out of the transceiver. Typically, the same as TXUSRCLK.
RXUSRCLK2	I	1	Clock output from a DCM that clocks the receiver data and status between the transceiver and the FPGA core. Typically the same as TXUSRCLK2. The relationship between RXUSRCLK and RXUSRCLK2 depends on the width of the RXDATA.
TXBUFERR	O	1	Provides status of the transmission FIFO. If asserted High, an overflow/underflow has occurred. When this bit becomes set, it can only be reset by asserting TXRESET.
TXBYPASS8B10B	I	1, 2, 4	This control signal determines whether the 8B/10B encoding is enabled or bypassed. If the signal is asserted High, the encoding is bypassed. This creates a 10-bit interface to the FPGA core. See the 8B/10B section for more details.
TXCHARDISPMODE	I	1, 2, 4	If 8B/10B encoding is enabled, this bus determines what mode of disparity is to be sent. When 8B/10B is bypassed, this becomes the 10th bit of the 10-bit encoded TXDATA bus for each byte specified by the byte-mapping section.
TXCHARDISPVAL	I	1, 2, 4	If 8B/10B encoding is enabled, this bus determines what type of disparity is to be sent. When 8B/10B is bypassed, this becomes the 9th bit of the 10-bit encoded TXDATA bus for each byte specified by the byte-mapping section.
TXCHARISK	I	1, 2, 4	If 8B/10B encoding is enabled, this control bus determines if the transmitted data is a "K" character or a Data character. A logic High indicating a K-character.
TXDATA	I	8,16,32	Transmit data that can be 1, 2, or 4 bytes wide, depending on the primitive used. TXDATA [7:0] is always the last byte transmitted. The position of the first byte depends on selected TX data path width.
TXFORCECRCERR	I	1	Specifies whether to insert error in computed CRC. When TXFORCECRCERR = TRUE, the transmitter corrupts the correctly computed CRC value by XORing with the bits specified in attribute TX_CRC_FORCE_VALUE. This input can be used to test detection of CRC errors at the receiver.
TXINHIBIT	I	1	If a logic High, the TX differential pairs are forced to be a constant 1/0. TXN = 1, TXP = 0

Table 3-1: **GT_CUSTOM[1], GT_AURORA, GT_FIBRE_CHAN[2], GT_ETHERNET[2], GT_INFINIBAND, and GT_XAUI Primitive Ports** *(Continued)*

Port	I/O	Port Size[3]	Definition
TXKERR	O	1, 2, 4	If 8B/10B encoding is enabled, this signal indicates (asserted High) when the "K" character to be transmitted is not a valid "K" character. Bits correspond to the byte-mapping scheme.
TXN[4]	O	1	Transmit differential port (FPGA external)
TXP[4]	O	1	Transmit differential port (FPGA external)
TXPOLARITY	I	1	Specifies whether or not to invert the final transmitter output. Able to reverse the polarity on the TXN and TXP lines. Deasserted sets regular polarity. Asserted reverses polarity.
TXRESET	I	1	Synchronous TX system reset that "recenters" the transmit elastic buffer. It also resets 8B/10B encoder and other internal transmission registers. It does not reset the transmission PLL.
TXRUNDISP	O	1, 2, 4	Signals the running disparity after this byte is encoded. Zero equals negative disparity and positive disparity for a one.
TXUSRCLK	I	1	Clock output from a DCM that is clocked with the REFCLK (or other reference clock). This clock is used for writing the TX buffer and is frequency-locked to the REFCLK.
TXUSRCLK2	I	1	Clock output from a DCM that clocks transmission data and status and reconfiguration data between the transceiver an the FPGA core. The ratio between the TXUSRCLK and TXUSRCLK2 depends on the width of the TXDATA.

Notes:
1. The GT_CUSTOM ports are always the maximum port size.
2. GT_FIBRE_CHAN and GT_ETHERNET ports do not have the three CHBOND** or ENCHANSYNC ports.
3. The port sizes change with relation to the primitive selected and also correlate to the byte mapping.
4. External ports only accessible from package pins.

Primitive Attributes

The primitives also contain attributes set by default to specific values controlling each specific primitive's protocol parameters. Included are channel-bonding settings (for primitives supporting channel bonding), clock correction sequences, and CRC. Table 3-2 shows a brief description of each attribute. Table 3-3 and Table 3-4 have the default values of each primitive.

Table 3-2:   **Rocket I/O Transceiver Attributes**

Attribute	Description
ALIGN_COMMA_MSB	True/False controls the alignment of detected commas within the transceivers 2-byte wide data path.
	False: Align commas within a 10-bit alignment range. As a result the comma is aligned to either RXDATA[15:8} byte or RXDATA [7:0] byte in the transceivers internal data path.
	True: Aligns comma with 20-bit alignment range.
	As a result aligns on the RXDATA[15:8] byte.
	NOTE: If protocols (like Gigabit Ethernet) are oriented in byte pairs with commas always in even (first) byte formation, this can be set to True. Otherwise, it should be set to False.
CHAN_BOND_LIMIT	Integer 1-31 that defines maximum number of bytes a slave receiver can read following a channel bonding sequence and still successfully align to that sequence.
CHAN_BOND_MODE	STRING OFF, MASTER, SLAVE_1_HOP, SLAVE_2_HOPS
	OFF: No channel bonding involving this transceiver.
	MASTER: This transceiver is master for channel bonding. Its CHBONDO port directly drives CHBONDI ports on one or more SLAVE_1_HOP transceivers.
	SLAVE_1_HOP: This transceiver is a slave for channel bonding. SLAVE_1_HOP's CHBONDI is directly driven by a MASTER transceiver CHBONDO port. SLAVE_1_HOP's CHBONDO port can directly drive CHBONDI ports on one or more SLAVE_2_HOPS transceivers.
	SLAVE_2_HOPS: This transceiver is a slave for channel bonding. SLAVE_2_HOPS CHBONDI is directly driven by a SLAVE_1_HOP CHBONDO port.
CHAN_BOND_OFFSET	Integer 0-15 that defines offset (in bytes) from channel bonding sequence for realignment. It specifies the first elastic buffer read address that all channel-bonded transceivers have immediately after channel bonding. CHAN_BOND_WAIT specifies the number of bytes that the master transceiver passes to RXDATA, starting with the channel bonding sequence, before the transceiver executes channel bonding (alignment) across all channel-bonded transceivers.
	CHAN_BOND_OFFSET specifies the first elastic buffer read address that all channel-bonded transceivers have immediately after channel bonding (alignment), as a positive offset from the beginning of the matched channel bonding sequence in each transceiver.
	For optimal performance of the elastic buffer, CHAN_BOND_WAIT and CHAN_BOND_OFFSET should be set to the same value (typically 8).

Table 3-2: **Rocket I/O Transceiver Attributes** *(Continued)*

Attribute	Description
CHAN_BOND_ONE_SHOT	True/False that controls repeated execution of channel bonding. **False:** Master transceiver initiates channel bonding whenever possible (whenever channel-bonding sequence is detected in the input) as long as input ENCHANSYNC is High and RXRESET is Low. **True:** Master transceiver initiates channel bonding only the first time it is possible (channel bonding sequence is detected in input) following negated RXRESET and asserted ENCHANSYNC. After channel-bonding alignment is done, it does not occur again until RXRESET is asserted and negated, or until ENCHANSYNC is negated and reasserted. Slave transceivers should always have CHAN_BOND_ONE_SHOT set to False.
CHAN_BOND_SEQ_*_*	11-bit vectors that define the channel bonding sequence. The usage of these vectors also depends on CHAN_BOND_SEQ_LEN and CHAN_BOND_SEQ_2_USE. See **Receiving Vitesse Channel Bonding Sequence**, page 59, for format.
CHAN_BOND_SEQ_2_USE	Controls use of second channel bonding sequence. **False:** Channel bonding uses only one channel bonding sequence defined by CHAN_BOND_SEQ_1_1..4. **True:** Channel bonding uses two channel bonding sequences defined by: CHAN_BOND_SEQ_1_1..4 and CHAN_BOND_SEQ_2_1..4 as further constrained by CHAN_BOND_SEQ_LEN.
CHAN_BOND_SEQ_LEN	Integer 1-4 defines length in bytes of channel bonding sequence. This defines the length of the sequence the transceiver matches to detect opportunities for channel bonding.
CHAN_BOND_WAIT	Integer 1-15 that defines the length of wait (in bytes) after seeing channel bonding sequence before executing channel bonding.
CLK_COR_INSERT_IDLE_FLAG	True/False controls whether RXRUNDISP input status denotes running disparity or inserted-idle flag. **False:** RXRUNDISP denotes running disparity when RXDATA is decoded data. **True:** RXRUNDISP is raised for the first byte of each inserted (repeated) clock correction ("Idle") sequence (when RXDATA is decoded data).
CLK_COR_KEEP_IDLE	True/False controls whether or not the final byte stream must retain at least one clock correction sequence. **False:** Transceiver can remove all clock correction sequences to further re-center the elastic buffer during clock correction. **True:** In the final RXDATA stream, the transceiver must leave at least one clock correction sequence per continuous stream of clock correction sequences.
CLK_COR_REPEAT_WAIT	Integer 0 - 31 controls frequency of repetition of clock correction operations. This attribute specifies the minimum number of RXUSRCLK cycles without clock correction that must occur between successive clock corrections. If this attribute is zero, no limit is placed on how frequently clock correction can occur.

Table 3-2: **Rocket I/O Transceiver Attributes** *(Continued)*

Attribute	Description
CLK_COR_SEQ_*_*	11-bit vectors that define the sequence for clock correction. The attribute used depends on the CLK_COR_SEQ_LEN and CLK_COR_SEQ_2_USE.
CLK_COR_SEQ_2_USE	True/False Control use of second clock correction sequence. **False:** Clock correction uses only one clock correction sequence defined by CLK_COR_SEQ_1_1..4. **True:** Clock correction uses two clock correction sequences defined by: CLK_COR_SEQ_1_1..4 and CLK_COR_SEQ_2_1..4 as further constrained by CLK_COR_SEQ_LEN.
CLK_COR_SEQ_LEN	Integer that defines the length of the sequence the transceiver matches to detect opportunities for clock correction. It also defines the size of the correction, since the transceiver executes clock correction by repeating or skipping entire clock correction sequences.
CLK_CORRECT_USE	True/False controls the use of clock correction logic. **False:** Permanently disable execution of clock correction (rate matching). Clock RXUSRCLK must be frequency-locked with RXRECCLK in this case. **True:** Enable clock correction (normal mode).
COMMA_10B_MASK	This 10-bit vector defines the mask that is ANDed with the incoming serial-bit stream before comparison against PCOMMA_10B_VALUE and MCOMMA_10B_VALUE.
CRC_END_OF_PKT	K28_0, K28_1, K28_2, K28_3, K28_4, K28_5, K28_6, K28_7, K23_7, K27_7, K29_7, K30_7 End-of-packet (EOP) K-character for USER_MODE CRC. Must be one of the 12 legal K-character values.
CRC_FORMAT	ETHERNET, INFINIBAND, FIBRE_CHAN, USER_MODE CRC algorithm selection. Modifiable only for GT_AURORA_n, GT_XAUI_n, and GT_CUSTOM. USER_MODE allows user definition of start-of-packet and end-of-packet K-characters.
CRC_START_OF_PKT	K28_0, K28_1, K28_2, K28_3, K28_4, K28_5, K28_6, K28_7, K23_7, K27_7, K29_7, K30_7 Start-of-packet (SOP) K-character for USER_MODE CRC. Must be one of the 12 legal K-character values.
DEC_MCOMMA_DETECT	True/False controls the raising of per-byte flag RXCHARISCOMMA on minus-comma.
DEC_PCOMMA_DETECT	True/False controls the raising of per-byte flag RXCHARISCOMMA on plus-comma.

Table 3-2: **Rocket I/O Transceiver Attributes** *(Continued)*

Attribute	Description
DEC_VALID_COMMA_ONLY	True/False controls the raising of RXCHARISCOMMA on an invalid comma.
	False: Raise RXCHARISCOMMA on:
	`0011111xxx` (if DEC_PCOMMA_DETECT is TRUE)
	and/or on:
	`1100000xxx` (if DEC_MCOMMA_DETECT is TRUE)
	regardless of the settings of the xxx bits.
	True: Raise RXCHARISCOMMA only on valid characters that are in the 8B/10B translation.
MCOMMA_10B_VALUE	This 10-bit vector defines minus-comma for the purpose of raising RXCOMMADET and realigning the serial bit stream byte boundary. This definition does not affect 8B/10B encoding or decoding. Also see COMMA_10B_MASK.
MCOMMA_DETECT	True/False indicates whether to raise or not raise the RXCOMMADET when minus-comma is detected.
PCOMMA_10B_VALUE	This 10-bit vector defines plus-comma for the purpose of raising RXCOMMADET and realigning the serial bit stream byte boundary. This definition does not affect 8B/10B encoding or decoding. Also see COMMA_10B_MASK.
PCOMMA_DETECT	True/False indicates whether to raise or not raise the RXCOMMADET when plus-comma is detected.
RX_BUFFER_USE	Always set to True.
RX_CRC_USE, TX_CRC_USE	True/False determines if CRC is used or not.
RX_DATA_WIDTH, TX_DATA_WIDTH	Integer (1, 2, or 4). Relates to the data width of the FPGA fabric interface.
RX_DECODE_USE	This determines if the 8B/10B decoding is bypassed. False denotes that it is bypassed.
RX_LOS_INVALID_INCR	Power of two in a range of 1 to 128 that denotes the number of valid characters required to "cancel out" appearance of one invalid character for loss of sync determination.
RX_LOS_THRESHOLD	Power of two in a range of 4 to 512. When divided by RX_LOS_INVALID_INCR, denotes the number of invalid characters required to cause FSM transition to "sync lost" state.
RX_LOSS_OF_SYNC_FSM	True/False denotes the nature of RXLOSSOFSYNC output. **True:** RXLOSSOFSYNC outputs the state of the FSM bit. See **RXLOSSOFSYNC, page 26**, for details.
SERDES_10B	Denotes whether the reference clock runs at 1/20 or 1/10 the serial bit rate. True denotes 1/10 and False denotes 1/20. False supports a serial bitstream range of 800 Mb/s to 3.125 Gb/s. True supports a range of 500 Mb/s to 1.0 Gb/s.

Table 3-2: **Rocket I/O Transceiver Attributes** *(Continued)*

Attribute	Description
TERMINATION_IMP	Integer (50 or 75). Termination impedance of either 50Ω or 75Ω. Refers to both the RX and TX.
TX_BUFFER_USE	Always set to True.
TX_CRC_FORCE_VALUE	8-bit vector. Value to corrupt TX CRC computation when input TXFORCECRCERR is high. This value is XORed with the correctly computed CRC value, corrupting the CRC if TX_CRC_FORCE_VALUE is nonzero. This can be used to test CRC error detection in the receiver downstream.
TX_DIFF_CTRL	An integer value (400, 500, 600, 700, or 800) representing 400 mV, 500 mV, 600 mV, 700 mV, or 800 mV of voltage difference between the differential lines. Twice this value is the peak-peak voltage.
TX_PREEMPHASIS	An integer value (0-3) that sets the output driver pre-emphasis to improve output waveform shaping for various load conditions. Larger value denotes stronger pre-emphasis. See pre-emphasis values in Table 4-2, page 76.

Modifiable Primitives

As shown in Table 3-3 and Table 3-4, only certain attributes are modifiable for any primitive. These attributes help to define the protocol used by the primitive. Only the GT_CUSTOM primitive allows the user to modify all of the attributes to a protocol not supported by another transceiver primitive. This allows for complete flexibility. The other primitives allow modification of the analog attributes of the serial data lines and several channel-bonding values.

Table 3-3: **Default Attribute Values: GT_AURORA, GT_CUSTOM, GT_ETHERNET**

Attribute	Default GT_AURORA	Default GT_CUSTOM[1]	Default GT_ETHERNET
ALIGN_COMMA_MSB	False	False	False
CHAN_BOND_LIMIT	16	16	1
CHAN_BOND_MODE	OFF[2]	OFF	OFF
CHAN_BOND_OFFSET	8	8	0
CHAN_BOND_ONE_SHOT	Falso[2]	False	True
CHAN_BOND_SEQ_1_1	00101111100	00000000000	00000000000
CHAN_BOND_SEQ_1_2	00000000000	00000000000	00000000000
CHAN_BOND_SEQ_1_3	00000000000	00000000000	00000000000
CHAN_BOND_SEQ_1_4	00000000000	00000000000	00000000000
CHAN_BOND_SEQ_2_1	00000000000	00000000000	00000000000
CHAN_BOND_SEQ_2_2	00000000000	00000000000	00000000000
CHAN_BOND_SEQ_2_3	00000000000	00000000000	00000000000
CHAN_BOND_SEQ_2_4	00000000000	00000000000	00000000000
CHAN_BOND_SEQ_2_USE	False	False	False
CHAN_BOND_SEQ_LEN	1	1	1
CHAN_BOND_WAIT	8	8	7
CLK_COR_INSERT_IDLE_FLAG	False[2]	False	False[2]
CLK_COR_KEEP_IDLE	False[2]	False	False[2]
CLK_COR_REPEAT_WAIT	1[2]	1	1[2]
CLK_COR_SEQ_1_1	00100011100	00000000000	00110111100
CLK_COR_SEQ_1_2	00100011100[4]	00000000000	00001010000
CLK_COR_SEQ_1_3	00100011100[5]	00000000000	00000000000
CLK_COR_SEQ_1_4	00100011100[5]	00000000000	00000000000
CLK_COR_SEQ_2_1	00000000000	00000000000	00000000000
CLK_COR_SEQ_2_2	00000000000	00000000000	00000000000
CLK_COR_SEQ_2_3	00000000000	00000000000	00000000000
CLK_COR_SEQ_2_4	00000000000	00000000000	00000000000
CLK_COR_SEQ_2_USE	False	False	False

Table 3-3: **Default Attribute Values: GT_AURORA, GT_CUSTOM, GT_ETHERNET** *(Continued)*

Attribute	Default GT_AURORA	Default GT_CUSTOM[1]	Default GT_ETHERNET
CLK_COR_SEQ_LEN	N[3]	1	2
CLK_CORRECT_USE	True	True	True
COMMA_10B_MASK	1111111111	1111111000	1111111000
CRC_END_OF_PKT	K29_7	K29_7	K29_7
CRC_FORMAT	USER_MODE	USER_MODE	ETHERNET
CRC_START_OF_PKT	K27_7	K27_7	K27_7
DEC_MCOMMA_DETECT	True	True	True
DEC_PCOMMA_DETECT	True	True	True
DEC_VALID_COMMA_ONLY	True	True	True
MCOMMA_10B_VALUE	1100000101	1100000000	1100000000
MCOMMA_DETECT	True	True	True
PCOMMA_10B_VALUE	0011111010	0011111000	0011111000
PCOMMA_DETECT	True	True	True
RX_BUFFER_USE	True	True	True
RX_CRC_USE	False[2]	False	False[2]
RX_DATA_WIDTH	N[3]	2	N[3]
RX_DECODE_USE	True	True	True
RX_LOS_INVALID_INCR	1[2]	1	1[2]
RX_LOS_THRESHOLD	4[2]	4	4[2]
RX_LOSS_OF_SYNC_FSM	True[2]	True	True[2]
SERDES_10B	False[2]	False	False[2]
TERMINATION_IMP	50[2]	50	50[2]
TX_BUFFER_USE	True	True	True
TX_CRC_FORCE_VALUE	11010110[2]	11010110	11010110[2]
TX_CRC_USE	False[2]	False	False[2]
TX_DATA_WIDTH	N[3]	2	N[3]
TX_DIFF_CTRL	500[2]	500	500[2]
TX_PREEMPHASIS	0[2]	0	0[2]

Notes:
1. All GT_CUSTOM attributes are modifiable.
2. Modifiable attribute for specific primitives.
3. Depends on primitive used: either 1, 2, or 4.
4. Attribute value only when RX_DATA_WIDTH is 2 or 4. When RX_DATA_WIDTH is 1, attribute value is 0.
5. Attribute value only when RX_DATA_WIDTH is 4. When RX_DATA_WIDTH is 1 or 2, attribute value is 0.

Table 3-4:   Default Attribute Values: GT_FIBRE_CHAN, GT_INFINIBAND, and GT_XAUI

Attribute	Default GT_FIBRE_CHAN	Default GT_INFINIBAND	Default GT_XAUI
ALIGN_COMMA_MSB	False	False	False
CHAN_BOND_LIMIT	1	16	16
CHAN_BOND_MODE	OFF	OFF[1]	OFF[1]
CHAN_BOND_OFFSET	0	8	8
CHAN_BOND_ONE_SHOT	True	False[1]	False[1]
CHAN_BOND_SEQ_1_1	00000000000	00110111100	00101111100
CHAN_BOND_SEQ_1_2	00000000000	Lane ID (Modify with Lane ID)	00000000000
CHAN_BOND_SEQ_1_3	00000000000	00001001010	00000000000
CHAN_BOND_SEQ_1_4	00000000000	00001001010	00000000000
CHAN_BOND_SEQ_2_1	00000000000	00110111100	00000000000
CHAN_BOND_SEQ_2_2	00000000000	Lane ID (Modify with Lane ID)	00000000000
CHAN_BOND_SEQ_2_3	00000000000	00001000101	00000000000
CHAN_BOND_SEQ_2_4	00000000000	00001000101	00000000000
CHAN_BOND_SEQ_2_USE	False	True	False
CHAN_BOND_SEQ_LEN	1	4	1
CHAN_BOND_WAIT	7	8	8
CLK_COR_INSERT_IDLE_FLAG	False[1]	False[1]	False[1]
CLK_COR_KEEP_IDLE	False[1]	False[1]	False[1]
CLK_COR_REPEAT_WAIT	2[1]	1[1]	1[1]
CLK_COR_SEQ_1_1	00110111100	00100011100	00100011100
CLK_COR_SEQ_1_2	00010010101	00000000000	00000000000
CLK_COR_SEQ_1_3	00010110101	00000000000	00000000000
CLK_COR_SEQ_1_4	00010110101	00000000000	00000000000
CLK_COR_SEQ_2_1	00000000000	00000000000	00000000000
CLK_COR_SEQ_2_2	00000000000	00000000000	00000000000
CLK_COR_SEQ_2_3	00000000000	00000000000	00000000000
CLK_COR_SEQ_2_4	00000000000	00000000000	00000000000
CLK_COR_SEQ_2_USE	False	False	False
CLK_COR_SEQ_LEN	4	1	1
CLK_CORRECT_USE	True	True	True
COMMA_10B_MASK	1111111000	1111111000	1111111000
CRC_END_OF_PKT	K29_7	Note (3)	K29_7[1]

Table 3-4: **Default Attribute Values: GT_FIBRE_CHAN, GT_INFINIBAND, and GT_XAUI** *(Continued)*

Attribute	Default GT_FIBRE_CHAN	Default GT_INFINIBAND	Default GT_XAUI
CRC_FORMAT	FIBRE_CHAN	INFINIBAND	USER_MODE[1]
CRC_START_OF_PKT	K27_7	Note (3)	K27_7[1]
DEC_MCOMMA_DETECT	True	True	True
DEC_PCOMMA_DETECT	True	True	True
DEC_VALID_COMMA_ONLY	True	True	True
Lane ID(INFINBAND ONLY)	NA	00000000000[1]	NA
MCOMMA_10B_VALUE	1100000000	1100000000	1100000000
MCOMMA_DETECT	True	True	True
PCOMMA_10B_VALUE	0011111000	0011111000	0011111000
PCOMMA_DETECT	True	True	True
RX_BUFFER_USE	True	True	True
RX_CRC_USE	False[1]	False[1]	False[1]
RX_DATA_WIDTH	N[2]	N[2]	N[2]
RX_DECODE_USE	True	True	True
RX_LOS_INVALID_INCR	1[1]	1[1]	1[1]
RX_LOS_THRESHOLD	4[1]	4[1]	4[1]
RX_LOSS_OF_SYNC_FSM	True[1]	True[1]	True[1]
SERDES_10B	False[1]	False[1]	False[1]
TERMINATION_IMP	50[1]	50[1]	50[1]
TX_BUFFER_USE	True	True	True
TX_CRC_FORCE_VALUE	11010110[1]	11010110[1]	11010110[1]
TX_CRC_USE	False[1]	False[1]	False[1]
TX_DATA_WIDTH	N[2]	N[2]	N[2]
TX_DIFF_CTRL	500[1]	500[1]	500[1]
TX_PREEMPHASIS	0[1]	0[1]	0[1]

Notes:
1. Modifiable attribute for specific primitives.
2. Depends on primitive used: either 1, 2, or 4.
3. CRC_EOP and CRC_SOP are not applicable for this primitive.

Byte Mapping

Most of the 4-bit wide status and control buses correlate to a specific byte of the TXDATA or RXDATA. This scheme is shown in Table 3-5. This creates a way to tie all the signals together regardless of the data path width needed for the GT_CUSTOM. All other primitives with specific data width paths and all byte-mapped ports are affected by this situation. For example, a 1-byte wide data path has only 1-bit control and status bits (TXKERR[0]) correlating to the data bits TXDATA[7:0]. Note 3 in Table 3-1 shows the ports that use byte mapping.

Table 3-5: **Control/Status Bus Association to Data Bus Byte Paths**

Control/Status Bit	Data Bits
[0]	[7:0]
[1]	[15:8]
[2]	[23:16]
[3]	[31:24]

Clocking

Clock Signals

There are five clock inputs into each Rocket I/O transceiver instantiation (Table 3-6). REFCLK is a clock generated from an external source. REFCLK is connected to the REFCLK of the Rocket I/O transceiver. It also clocks a Digital Clock Manager (DCM) to generate all of the other clocks for the gigabit transceiver. Typically, TXUSRCLK = RXUSRCLK and TXUSRCLK2 = RXUSRCLK2. The transceiver uses one or two clocks generated by the DCM. As an example, the USRCLK and USRCLK2 clocks run at the same speed if the 2-byte data path is used. The USRCLK must always be frequency-locked to the reference clock, REFCLK of the Rocket I/O transceiver.

NOTE: The REFCLK must be at least 40 MHz with a duty cycle between 45% and 55%, and should have a frequency stability of 100 ppm or better, with jitter as low as possible. Module 3 of the Virtex-II Pro data sheet gives further details.

Table 3-6: **Clock Ports**

Clock	I/Os	Description
RXRECCLK	Output	Recovered clock (from serial data stream) divided by 20
REFCLK	Input	Reference clock used to read the TX FIFO and multiplied by 20 for parallel-to-serial conversion (20X)
REFCLK2	Input	Reference clock used to read the TX FIFO and multiplied by 20 for parallel-to-serial conversion (20X)
REFCLKSEL	Input	Selects which reference clock is used. 0 selects REFCLK; 1 selects REFCLK2.
RXUSRCLK	Input	Clock from FPGA used for reading the RX Elastic Buffer. Clock signals CHBONDI and CHBONDO into and out of the transceiver. This clock is typically the same as TXUSRCLK.
TXUSRCLK	Input	Clock from FPGA used for writing the TX Buffer. This clock must be frequency locked to REFCLK for proper operation.
RXUSRCLK2	Input	Clock from FPGA used to clock RX data and status between the transceiver and FPGA fabric. The relationship between RXUSRCLK2 and RXUSRCLK depends on the width of the receiver data path. RXUSRCLK2 is typically the same as TXUSRCLK2.
TXUSRCLK2	Input	Clock from FPGA used to clock TX data and status between the transceiver and FPGA fabric. The relationship between TXUSRCLK2 and TXUSRCLK depends on the width of the transmission data path.

Clock Ratio

USRCLK2 clocks the data buffers. The ability to send parallel data to the transceiver at three different widths requires the user to change the frequency of USRCLK2. This creates a frequency ratio between USRCLK and USRCLK2. The falling edges of the clocks must align. Finally, for a 4-byte data path, the 1-byte data path creates a clocking scheme where USRCLK2 is phase-shifted 180° and at twice the rate of USRCLK.

Table 3-7:   **Data Width Clock Ratios**

Data Width	Frequency Ratio of USRCLK\USRCLK2
1 byte	1:2[1]
2 byte	1:1
4 byte	2:1[1]

Notes:
1.   Each edge of slower clock must align with falling edge of faster clock

Digital Clock Manager (DCM) Examples

With at least three different clocking schemes possible on the transceiver, a DCM is the best way to create these schemes.

Table 3-8 shows typical DCM connections for several transceiver clocks. REFCLK is the input reference clock for the DCM. The other clocks are generated by the DCM. The DCM establishes a desired phase relationship between RXUSRCLK, TXUSRCLK, etc. in the FPGA core and REFCLK at the pad.

Table 3-8:   **DCM Outputs for Different DATA_WIDTHs**

SERDES_10B	TX_DATA_WIDTH RX_DATA_WIDTH	REFCLK	TXUSRCLK RXUSRCLK	TXUSRCLK2 RXUSRCLK2
False	1	CLKIN	CLK0	CLK2X180
False	2	CLKIN	CLK0	CLK0
False	4	CLKIN	CLK180[1]	CLKDV (divide by 2)
True	1	CLKIN	CLKDV (divide by 2)	CLK180[1]
True	2	CLKIN	CLKDV (divide by 2)	CLKDV (divide by 2)
True	4	CLKIN	CLKFX180 (divide by 2)	CLKDV (divide by 4)

Notes:
1.   Since CLK0 is needed for feedback, it can be used instead of CLK180 to clock USRCLK or USRCLK2 of the transceiver with the use of the transceiver's local inverter, saving a global buffer (BUFG).

Example 1: Two-Byte Clock

The following HDL codes are examples of a simple clock scheme using 2-byte data with both USRCLK and USRCLK2 at the same frequency. USRCLK_M is the input for both USRCLK and USRCLK2.

Clocks for 2-Byte Data Path   MGT + DCM for 2-Byte Data Path
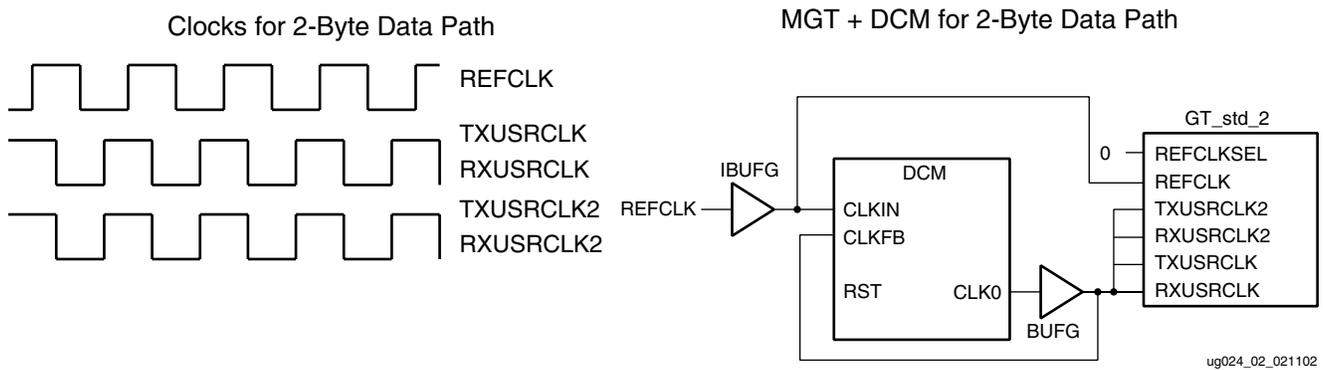
Figure 3-1: **Two-Byte Clock**

VHDL Template

```
-- Module:         TWO_BYTE_CLK
-- Description:    VHDL submodule
--                 DCM for 2-byte GT
--
-- Device:         Virtex-II Pro Family
--------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on
--
entity TWO_BYTE_CLK is
  port (
        REFCLKIN  : in std_logic;
        RST       : in std_logic;
        USRCLK_M  : out std_logic;
        REFCLK    : out std_logic;
        LOCK      : out std_logic
        );
end TWO_BYTE_CLK;
--
architecture TWO_BYTE_CLK_arch of TWO_BYTE_CLK is
--
-- Components Declarations:
component BUFG
  port (
        I : in std_logic;
        O : out std_logic
        );
end component;
--
component IBUFG
  port (
        I : in std_logic;
        O : out std_logic
```

```
                    );
           end component;
           --
           component DCM
            port (
                   CLKIN     : in std_logic;
                   CLKFB     : in std_logic;
                   DSSEN     : in std_logic;
                   PSINCDEC  : in std_logic;
                   PSEN      : in std_logic;
                   PSCLK     : in std_logic;
                   RST       : in std_logic;
                   CLK0      : out std_logic;
                   CLK90     : out std_logic;
                   CLK180    : out std_logic;
                   CLK270    : out std_logic;
                   CLK2X     : out std_logic;
                   CLK2X180  : out std_logic;
                   CLKDV     : out std_logic;
                   CLKFX     : out std_logic;
                   CLKFX180  : out std_logic;
                   LOCKED    : out std_logic;
                   PSDONE    : out std_logic;
                   STATUS    : out std_logic_vector ( 7 downto 0 )
                   );
           end component;
           --
           -- Signal Declarations:
           --
           signal GND       : std_logic;
           signal CLK0_W    : std_logic;
           signal CLK1X_W   : std_logic;


           begin

           GND    <= '0';
           --
           CLK1X  <= CLK1X_W;
           --
           -- DCM Instantiation
           U_DCM: DCM
            port map (
                   CLKIN      =>   REFCLK,
                   CLKFB      =>   USRCLK_M,
                   DSSEN      =>   GND,
                   PSINCDEC   =>   GND,
                   PSEN       =>   GND,
                   PSCLK      =>   GND,
                   RST        =>   RST,
                   CLK0       =>   CLK0_W,
                   LOCKED     =>   LOCK
                   );
           --
           -- BUFG Instantiation
           U_BUFG: IBUFG
            port map (
                   I  => REFCLKIN,
                   O  => REFCLK
                   );
```

```
U2_BUFG: BUFG
  port map (
            I  => CLK0_W,
            O  => USRCLK_M
           );


    end TWO_BYTE_CLK_arch;
```

Verilog Template

```
//Module:        TWO_BYTE_CLK
//Description:   Verilog Submodule
//               DCM for 2-byte GT
//
// Device:       Virtex-II Pro Family

module TWO_BYTE_CLK (
                     REFCLKIN,
                     REFCLK,
                     USRCLK_M,
                     DCM_LOCKED
                    );

  input   REFCLKIN;
  output  REFCLK;
  output  USRCLK_M;
  output  DCM_LOCKED;

  wire    REFCLKIN;
  wire    REFCLK;
  wire    USRCLK_M;
  wire    DCM_LOCKED;
  wire    REFCLKINBUF;
  wire    clk_i;

DCM dcm1 (
         .CLKFB     ( USRCLK_M ),
         .CLKIN     ( REFCLKINBUF ),.DSSEN( 1'b0 ),
         .PSCLK     ( 1'b0 ),
         .PSEN      ( 1'b0 ),
         .PSINCDEC  ( 1'b0 ),
         .RST       ( 1'b0 ),
         .CLK0      ( clk_i ),
         .CLK90     (  ),
         .CLK180    (  ),
         .CLK270    (  ),
         .CLK2X     (  ),
         .CLK2X180  (  ),
         .CLKDV     (  ),
         .CLKFX     (  ),
         .CLKFX180  (  ),
         .LOCKED    ( DCM_LOCKED ),
         .PSDONE    (  ),
         .STATUS    (  )
         );

BUFG buf1 (
         .I ( clk_i ),
         .O ( USRCLK_M )
         );

IBUFG buf2(
         .I ( REFCLKIN ),
```

```
                .O ( REFCLKINBUF ));

        endmodule
```

Example 2: Four-Byte Clock

If a 4-byte or 1-byte data path is chosen, the ratio between USRCLK and USRCLK2 changes. The time it take for the SERDES to serialize the parallel data requires the change in ratios.

The DCM example (Figure 3-2) is detailed for a 4-byte data path. If 3.125 Gb/s is required, the REFCLK is 156 MHz and the USRCLK2_M only runs at 78 MHz including the clocking for any interface logic. Both USRCLK and USRCLK2 are aligned on the falling edge since the USRCLK_M is 180° out of phase when using local inverters with the transceiver.

Clocks for 4-Byte Data Path

MGT + DCM for 4-Byte Data Path

UG024_03_020802
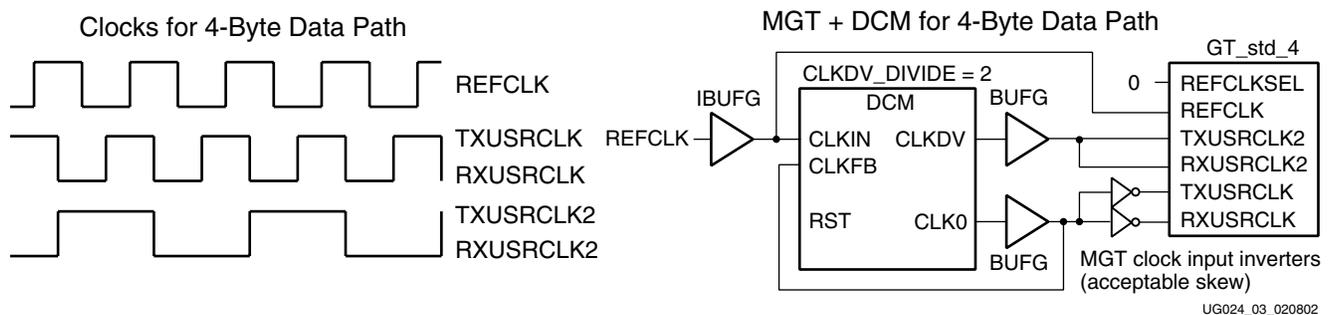
Figure 3-2: **Four-Byte Clock**

VHDL Template

```
-- Module:          FOUR_BYTE_CLK
-- Description:     VHDL submodule
--                  DCM for 4-byte GT
--
-- Device:          Virtex-II Pro Family
------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on
--
entity FOUR_BYTE_CLK is
  port (
        REFCLKIN    : in std_logic;
        RST         : in std_logic;
        USRCLK_M    : out std_logic;
        USRCLK2_M   : out std_logic;
        REFCLK      : out std_logic;
        LOCK        : out std_logic
        );
end FOUR_BYTE_CLK;
--
architecture FOUR_BYTE_CLK_arch of FOUR_BYTE_CLK is
--
-- Components Declarations:
component BUFG
  port (
        I : in std_logic;
```

```
            O : out std_logic
            );
end component;
--
component IBUFG
  port (
        I : in std_logic;
        O : out std_logic
        );
end component;
--
component DCM
  port (
        CLKIN     : in std_logic;
        CLKFB     : in std_logic;
        DSSEN     : in std_logic;
        PSINCDEC  : in std_logic;
        PSEN      : in std_logic;
        PSCLK     : in std_logic;
        RST       : in std_logic;
        CLK0      : out std_logic;
        CLK90     : out std_logic;
        CLK180    : out std_logic;
        CLK270    : out std_logic;
        CLK2X     : out std_logic;
        CLK2X180  : out std_logic;
        CLKDV     : out std_logic;
        CLKFX     : out std_logic;
        CLKFX180  : out std_logic;
        LOCKED    : out std_logic;
        PSDONE    : out std_logic;
        STATUS    : out std_logic_vector ( 7 downto 0 )
        );
end component;
--
-- Signal Declarations:
--
signal GND      : std_logic;
signal CLK0_W   : std_logic;
signal CLKDV_W  : std_logic;

begin

GND    <= '0';
-- DCM Instantiation
U_DCM: DCM
  port map (
        CLKIN     =>   REFCLK,   CLKFB     =>   CLK0_W,
        DSSEN     =>   GND,
        PSINCDEC  =>   GND,
        PSEN      =>   GND,
        PSCLK     =>   GND,
        RST       =>   RST,
        CLK0      =>   CLK0_W,
        CLKDV     =>   CLKDV_W,
        LOCKED    =>   LOCK
        );

-- BUFG Instantiation
U_BUFG: IBUFG
  port map (
        I => REFCLKIN,
```

```
                            O => REFCLK     );


            U2_BUFG: BUFG
              port map (
                        I => CLK0_W,
                        O => USRCLK_M
                        );

            U3_BUFG: BUFG
              port map (
                        I => CLKDV_W,
                        O => USRCLK2_M
                        );


            end FOUR_BYTE_CLK_arch;
```

Verilog Template

```verilog
// Module:       FOUR_BYTE_CLK
// Description:  Verilog Submodule
//               DCM for 4-byte GT
//
// Device:       Virtex-II Pro Family

module FOUR_BYTE_CLK(
                        REFCLKIN,
                        REFCLK,
                        USRCLK_M,
                        USRCLK2_M,
                        DCM_LOCKED
                         );

   input     REFCLKIN;
   output    REFCLK;
   output    USRCLK_M;
   output    USRCLK2_M;
   output    DCM_LOCKED;

   wire      REFCLKIN;
   wire      REFCLK;
   wire      USRCLK_M;
   wire      USRCLK2_M;
   wire      DCM_LOCKED;
   wire      REFCLKINBUF;
   wire      clkdv2;
   wire      clk_i;

            DCM dcm1 (
                     .CLKFB      ( USRCLK_M ),
                     .CLKIN      ( REFCLKINBUF ) ,   .DSSEN        (
1'b0 ),
                     .PSCLK      ( 1'b0 ),
                     .PSEN       ( 1'b0 ),
                     .PSINCDEC   ( 1'b0 ),
                     .RST        ( 1'b0 ),
                     .CLK0       ( clk_i ),
                     .CLK90      (   ),
                     .CLK180     (   ),
                     .CLK270     (   ),
                     .CLK2X      (   ),
                     .CLK2X180   (   ),
                     .CLKDV      ( clkdv2 ),
```

```
                              .CLKFX        (  ),
                              .CLKFX180     (  ),
                              .LOCKED       ( DCM_LOCKED ),
                              .PSDONE       (  ),
                              .STATUS       (  )
                              );

                      BUFG buf1 (
                              .I ( clkdv2 ),
                              .O ( USRCLK2_M )
                              );

                      BUFG buf2 (
                              .I ( clk_i ),
                              .O ( USRCLK_M )
                              );

                      IBUFG buf3(
                              .I ( REFCLKIN ),
                              .O ( REFCLKINBUF )      );

              endmodule
```

Example 3: One-Byte Clock

This is the 1-byte wide data path clocking scheme example. USRCLK2_M is twice as fast as USRCLK_M. It is also phase-shifted 180° for falling edge alignment.
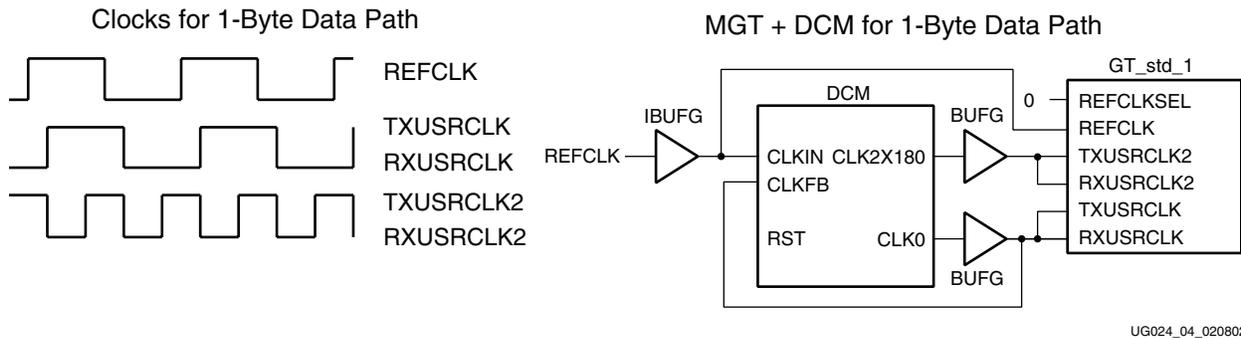
Figure 3-3: **One-Byte Clock**

VHDL Template

```
-- Module:          ONE_BYTE_CLK
-- Description:     VHDL submodule
--                  DCM for 1-byte GT
--
-- Device:          Virtex-II Pro Family
----------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on
--
entity ONE_BYTE_CLK is
  port (
        REFCLKIN    : in std_logic;
```

```
                RST          : in std_logic;
                USRCLK_M     : out std_logic;
                USRCLK2_M    : out std_logic;
                REFCLK       : out std_logic;
                LOCK         : out std_logic
                );
        end ONE_BYTE_CLK;
        --
        architecture ONE_BYTE_CLK_arch of ONE_BYTE_CLK is
        --
        -- Components Declarations:
        component BUFG
          port (
                I   : in std_logic;
                O   : out std_logic
                );
        end component;
        --
        component IBUFG
          port (
                I   : in std_logic;
                O   : out std_logic
                );
        end component;
        --
        component DCM
          port (
                CLKIN        : in std_logic;
                CLKFB        : in std_logic;
                DSSEN        : in std_logic;
                PSINCDEC     : in std_logic;
                PSEN         : in std_logic;
                PSCLK        : in std_logic;
                RST          : in std_logic;
                CLK0         : out std_logic;
                CLK90        : out std_logic;
                CLK180       : out std_logic;
                CLK270       : out std_logic;
                CLK2X        : out std_logic;
                CLK2X180     : out std_logic;
                CLKDV        : out std_logic;
                CLKFX        : out std_logic;
                CLKFX180     : out std_logic;
                LOCKED       : out std_logic;
                PSDONE       : out std_logic;
                STATUS       : out std_logic_vector ( 7 downto 0 )
                );
        end component;
        --
        -- Signal Declarations:
        --
        signal GND        : std_logic;
        signal CLK0_W     : std_logic;
        signal CLK1X_W    : std_logic;
        signal CLK2X180_W : std_logic;

        begin

        GND    <= '0';
        --
        CLK1X  <= CLK1X_W;
        --
```

```
-- DCM Instantiation
U_DCM: DCM
  port map (
          CLKIN      =>   REFCLK,

          CLKFB      =>   USRCLK_M,
          DSSEN      =>   GND,
          PSINCDEC   =>   GND,
          PSEN       =>   GND,
          PSCLK      =>   GND,
          RST        =>   RST,
          CLK0       =>   CLK0_W,
          CLK2X180   =>   CLK2X180_W,
          LOCKED     =>   LOCK
          );
-- BUFG Instantiation
U_BUFG: IBUFG
  port map (
          I => REFCLKIN,
          O => REFCLK

          );

U2_BUFG: BUFG
  port map (
          I => CLK0_W,
          O => USRCLK_M
          );

U4_BUFG: BUFG
  port map (
          I => CLK2X180_W,
          O => USRCLK2
          );

end ONE_BYTE_CLK_arch;
```

Verilog Template

```
// Module:      ONE_BYTE_CLK
// Description: Verilog Submodule
//              DCM for 1-byte GT
//
// Device:      Virtex-II Pro Family

module ONE_BYTE_CLK (
                    REFCLKIN,
                    REFCLK,
                    USRCLK_M,
                    USRCLK2_M,
                    DCM_LOCKED
                    );

  input    REFCLKIN;
  output   REFCLK;
  output   USRCLK_M;
  output   USRCLK2_M;
  output   DCM_LOCKED;

  wire     REFCLKIN;
  wire     REFCLK;
  wire     USRCLK_M;
```

```
wire      USRCLK2_M;
wire      DCM_LOCKED;
wire      REFCLKINBUF;
wire      clk_i;
wire      clk_2x_180;

DCM dcm1 (
          .CLKFB      ( USRCLK_M ),
          .CLKIN      ( REFCLKINBUF),

          .DSSEN      ( 1'b0 ),
          .PSCLK      ( 1'b0 ),
          .PSEN       ( 1'b0 ),
          .PSINCDEC   ( 1'b0 ),
          .RST        ( 1'b0 ),
          .CLK0       ( clk_i ),
          .CLK90      (   ),
          .CLK180     (   ),
          .CLK270     (   ),
          .CLK2X      (   ),
          .CLK2X180   ( clk2x_180 ),
          .CLKDV      (   ),
          .CLKFX      (   ),
          .CLKFX180   (   ),
          .LOCKED     ( DCM_LOCKED ),
          .PSDONE     (   ),
          .STATUS     (   )
          );

  BUFG buf1 (
              .I ( clk2x_180 ),
              .O ( USRCLK2_M )
              );

  BUFG buf2 (
              .I ( clk_i ),
              .O ( USRCLK_M )
              );

  IBUFGbuf3 (
              .I ( REFCLKIN ),
              .O ( REFCLKINBUF )

              );

endmodule
```

Multiplexed Clocking Scheme

Following configuration of the FPGA, some applications might need to change the frequency of its REFCLK depending on the protocol used. Figure 3-4 shows how the design can use two different reference clocks connected to two different DCMs. The clocks are then multiplexed before input into the Rocket I/O transceiver.

User logic can be designed to determine during autonegotiation if the reference clock used for the transceiver is incorrect. If so, the transceiver must then be reset and another reference clock selected.
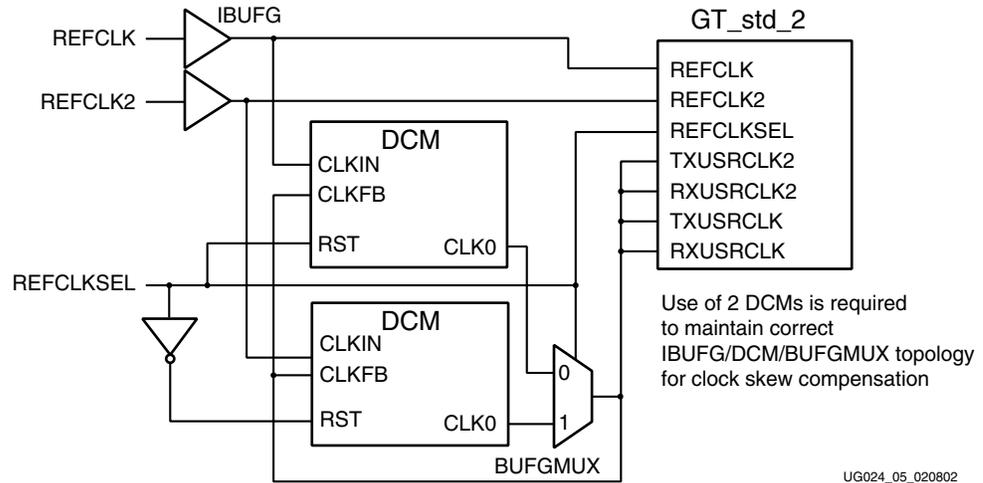
Figure 3-4: **Multiplexed REFCLK**

Clock Dependency

All signals used by the FPGA fabric to interact between user logic and the transceiver depend on an edge of USRCLK2. These signals all have setup and hold times with respect to this clock. For specific timing values, see Module 3 of the Virtex-II Pro data sheet.

The timing relationships are illustrated.

Table 3-9, **Parameters Relative to the RX User Clock (RXUSRCLK)**, page 50
Table 3-10, **Parameters Relative to the RX User Clock2 (RXUSRCLK2)**, page 50
Table 3-11, **Parameters Relative to the TX User Clock2 (TXUSRCLK2)**, page 51
Table 3-12, **Miscellaneous Clock Parameters**, page 52

Table 3-9: **Parameters Relative to the RX User Clock (RXUSRCLK)**

Parameter	Function	Signals
Setup/Hold:		
T_{GCCK}_CHBI/T_{GCKC}_CHBI	Control inputs	CHBONDI[3:0]
Clock to Out:		
T_{GCKCO}_CHBO	Control outputs	CHBONDO[3:0]
Clock:		
T_{RXPWH}	Clock pulse width, High state	RXUSRCLK
T_{RXPWL}	Clock pulse width, Low state	RXUSRCLK

Table 3-10: **Parameters Relative to the RX User Clock2 (RXUSRCLK2)**

Parameter	Function	Signals
Setup/Hold:		
T_{GCCK}_RRST/T_{GCKC}_RRST	Control input	RXRESET
T_{GCCK}_RPOL/T_{GCKC}_RPOL	Control input	RXPOLARITY
T_{GCCK}_ECSY/T_{GCKC}_ECSY	Control input	ENCHANSYNC
Clock to Out:		

Table 3-10: **Parameters Relative to the RX User Clock2 (RXUSRCLK2)** *(Continued)*

Parameter	Function	Signals
$T_{GCKST\_}RNIT$	Status outputs	RXNOTINTABLE[3:0]
$T_{GCKST\_}RDERR$	Status outputs	RXDISPERR[3:0]
$T_{GCKST\_}RCMCH$	Status outputs	RXCHARISCOMMA[3:0]
$T_{GCKST\_}ALIGN$	Status output	RXREALIGN
$T_{GCKST\_}CMDT$	Status output	RXCOMMADET
$T_{GCKST\_}RLOS$	Status outputs	RXLOSSOFSYNC[1:0]
$T_{GCKST\_}RCCCNT$	Status outputs	RXCLKCORCNT[2:0]
$T_{GCKST\_}RBSTA$	Status outputs	RXBUFSTATUS[1:0]
$T_{GCKST\_}RCCRC$	Status output	RXCHECKINGCRC
$T_{GCKST\_}RCRCE$	Status output	RXCRCERR
$T_{GCKST\_}CHBD$	Status output	CHBONDDONE
$T_{GCKST\_}RKCH$	Status outputs	RXCHARISK[3:0]
$T_{GCKST\_}RRDIS$	Status outputs	RXRUNDISP[3:0]
$T_{GCKDO\_}RDAT$	Data outputs	RXDATA[31:0]
Clock:		
T_{RX2PWH}	Clock pulse width, High state	RXUSRCLK2
T_{RX2PWH}	Clock pulse width, Low state	RXUSRCLK2

Table 3-11: **Parameters Relative to the TX User Clock2 (TXUSRCLK2)**

Parameter	Function	Signals
Setup/Hold:		
$T_{GCCK\_}CFGEN/T_{GCKC\_}CFGEN$	Control inputs	CONFIGENABLE
$T_{GCCK\_}TBYP/T_{GCKC\_}TBYP$	Control inputs	TXBYPASS8B10B[3:0]
$T_{GCCK\_}TCRCE/T_{GCKC\_}TCRCE$	Control inputs	TXFORCECRCERR
$T_{GCCK\_}TPOL/T_{GCKC\_}TPOL$	Control inputs	TXPOLARITY
$T_{GCCK\_}TINH/T_{GCKC\_}TINH$	Control inputs	TXINHIBIT
$T_{GCCK\_}LBK/T_{GCKC\_}LBK$	Control inputs	LOOPBACK[1:0]
$T_{GCCK\_}TRST/T_{GCKC\_}TRST$	Control inputs	TXRESET
$T_{GCCK\_}TKCH/T_{GCKC\_}TKCH$	Control inputs	TXCHARISK[3:0]
$T_{GCCK\_}TCDM/T_{GCKC\_}TCDM$	Control inputs	TXCHARDISPMODE[3:0]
$T_{GCCK\_}TCDV/T_{GCKC\_}TCDV$	Control inputs	TXCHARDISPVAL[3:0]
$T_{GDCK\_}CFGIN/T_{GCKD\_}CFGIN$	Data inputs	CONFIGIN
$T_{GDCK\_}TDAT/T_{GCKD\_}TDAT$	Data inputs	TXDATA[31:0]
Clock to Out:		
$T_{GCKST\_}TBERR$	Status outputs	TXBUFERR
$T_{GCKST\_}TKERR$	Status outputs	TXKERR[3:0]

Table 3-11: **Parameters Relative to the TX User Clock2 (TXUSRCLK2)** *(Continued)*

Parameter	Function	Signals
$T_{GCKDO\_}$TRDIS	Data outputs	TXRUNDISP[3:0]
$T_{GCKDO\_}$CFGOUT	Data outputs	CONFIGOUT
Clock:		
T_{TX2PWH}	Clock pulse width, High state	TXUSRCLK2
T_{TX2PWH}	Clock pulse width, Low state	TXUSRCLK2

Table 3-12: **Miscellaneous Clock Parameters**

Parameter	Function	Signals
Clock:		
T_{REFPWH}	Clock pulse width, High state	REFCLK[1]
T_{REFPWL}	Clock pulse width, Low state	REFCLK[1]
T_{TXPWH}	Clock pulse width, High state	TXUSRCLK[2]
T_{TXPWL}	Clock pulse width, Low state	TXUSRCLK[2]

Notes:
1. REFCLK is not synchronous to any Rocket I/O signals.
2. TXUSRCLK is not synchronous to any Rocket I/O signals.
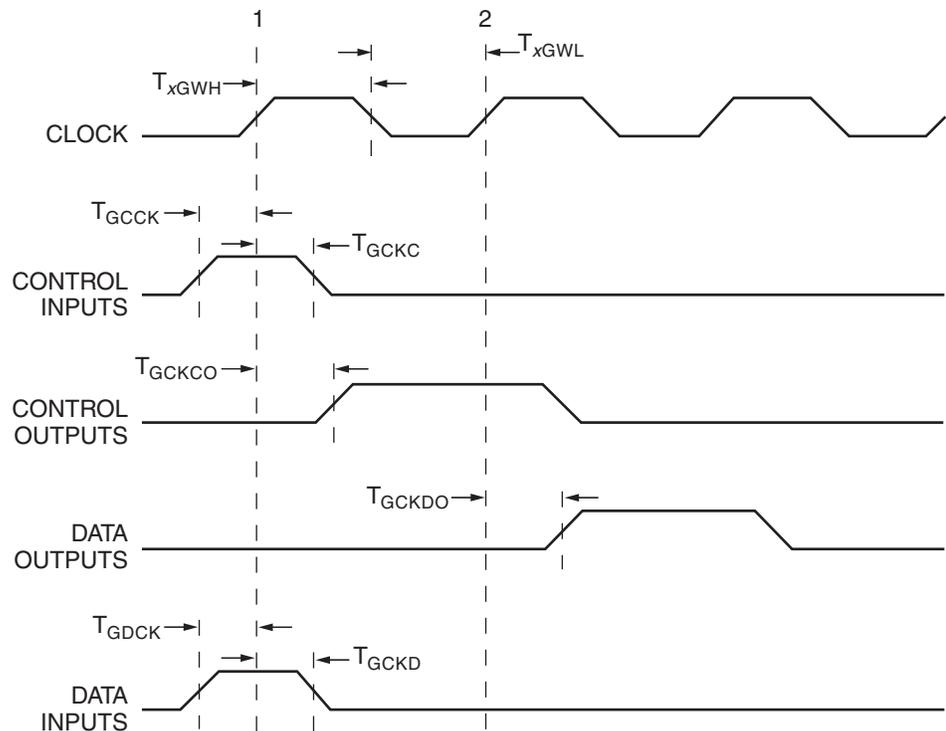
Figure 3-5: **Rocket I/O Timing Relative to Clock Edge**

Resets

There are two reset signals, one each for the transmit and receive sections of each gigabit transceiver. After the DCM locked signal is asserted, the resets can be asserted. The resets must be asserted for two USRCLK2 cycles to ensure correct initialization of the FIFOs. Although both the transmit and receive resets can be attached to the same signal, separate signals are preferred. This allows the elastic buffer to be cleared in case of an over/underflow without affecting the ongoing TX transmission. The following example is an implementation to reset all three data-width transceivers.

VHDL Template

```vhdl
-- Module: gt_reset
-- Description: VHDL submodule
-- reset for GT
--
-- Device: Virtex-II Pro Family
----------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.Numeric_STD.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on
--
entity gt_reset is
port (
USRCLK2_M : in std_logic;
LOCK      : in std_logic;
REFCLK    : out std_logic;
DCM_LOCKED: in std_logic;
RST       : out std_logic);
end gt_reset;
--
architecture RTL of gt_reset is
--
  signal startup_count : std_logic_vector (7 downto 0);
begin
  process (USRCLK2_M, DCM_LOCKED)

  begin
    if (USRCLK2_M' event and USRCLK2_M = '1') then
      if(DCM_LOCKED = '0') then
        startup_count <= "00000000";
      elsif (DCM_LOCKED = '1') then
        startup_count <= startup_count + "00000001";
      end if;
    end if;

if (USRCLK2_M'  event and USRCLK2_M = '1') then
  if(DCM_LOCKED = '0') then
    RST <= '1';
  elsif (startup_count = "00000010") then
    RST <= '0';
  end if;
end if;

end process;
```

```
                        end RTL;
```

Verilog Template

```verilog
// Module:      gt_reset
// Description: Verilog Submodule
//              reset for4-byte GT
//
// Device:      Virtex-II Pro Family

module gt_reset(
                USRCLK2_M,
                DCM_LOCKED,
                RST
              );

input       USRCLK2_M;
input       DCM_LOCKED;
input       RST;

wire        USRCLK2_M;
wire        DCM_LOCKED;
reg         RST;
reg [7:0]   startup_counter;

always @ ( posedge USRCLK2_M )
    if ( !DCM_LOCKED )
        startup_counter <= 8'h0;
    else if ( startup_counter != 8'h02 )
        startup_counter <= startup_counter + 1;

always @ ( posedge USRCLK2_M or negedge DCM_LOCKED )
    if ( !DCM_LOCKED )
        RST <= 1'b1;
    else
        RST <= ( startup_counter != 8'h02 );

endmodule
```

Rocket I/O Transceiver Instantiations

For the different clocking schemes, several things must change, including the clock frequency for USRCLK and USRCLK2 discussed in the DCM section above. The data and control ports for GT_CUSTOM must also reflect this change in data width by concatenating zeros onto inputs and wires for outputs for Verilog designs, and by setting outputs to open and concatenating zeros on unused input bits for VHDL designs.

HDL Code Examples

Availability of downloadable GT_CUSTOM code examples with 1-, 2-, and 4-byte data widths is planned for a later date.

PLL Operation and Clock Recovery

The clock correction sequence is a special sequence to accommodate frequency differences between the received data (as reflected in RXRECCLK) and RXUSRCLK. Most of the primitives have these defaulted to the respective protocols. Only the GT_CUSTOM allows this sequence to be set to any specific protocol. The sequence contains 11 bits including the 10 bits of serial data. The 11th bit has two different formats. The typical usage is:

```
        0, disparity error required, char is K, 8-bit data value (after
    8B/10B decoding)
        1, 10 bit data value (without 8B/10B decoding)
```

Table 3-13 is an example of data 11-bit attribute setting, the character value, CHARISK value, and the parallel data interface, and how each corresponds with the other.

Table 3-13: **Clock Correction Sequence / Data Correlation for 16-Bit Data Port**

Attribute Setting	Character	CHARISK	TXDATA (hex)
CLK_COR_SEQ_1_1 = 00110111100	K28.5	1	BC
CLK_COR_SEQ_1_2 = 00010010101	D21.4	0	95
CLK_COR_SEQ_1_3 = 00010110101	D21.5	0	B5
CLK_COR_SEQ_1_4 = 00010110101	D21.5	0	B5

The GT_CUSTOM transceiver examples use the previous sequence for clock correction.

Clock Correction Count

The clock correction count signal (RXCLKCORCNT) is a three-bit signal. It signals if the clock correction has occurred and whether the elastic buffer realigned the data by skipping or repeating data in the buffer. It also signals if channel bonding has occurred (Table 3-14).

Table 3-14: **RXCLKCORCNT Definition**

RXCLKCORCNT[2:0]	Significance
000	No channel bonding or clock correction occurred for current RXDATA
001	Elastic buffer skipped one clock correction sequence for current RXDATA
010	Elastic buffer skipped two clock correction sequence for current RXDATA
011	Elastic buffer skipped three clock correction sequence for current RXDATA
100	Elastic buffer skipped four clock correction sequence for current RXDATA
101	Elastic buffer executed channel bonding for current RXDATA
110	Elastic buffer repeated two clock correction sequences for current RXDATA
111	Elastic buffer repeated one clock correction sequences for current RXDATA

RX_LOSS_OF_SYNC_FSM

The transceivers FSM is driven by RXRECLK and uses status from the data stream prior to the elastic buffer. This is intended to give early warning of possible problems well before corrupt data appears on RXDATA. There are three states.

SYNC_ACQUIRED (RXLOSSOFSYNC = 00)

In this state, a counter is decremented by 1 (but not past 0) for a valid received symbol and incremented by RX_LOS_INVALID_INCR for an invalid symbol. If the count reaches or exceeds RX_LOS_THRESHOLD, the FSM moves to state LOSS_OF_SYNC. Otherwise, if a

channel bonding (alignment) sequence has just been written into the elastic buffer, or if a comma realignment has just occurred, the FSM moves to state RESYNC. Otherwise, the FSM remains in state SYNC_ACQUIRED.

RESYNC (RXLOSSOFSYNC = 01)

The FSM waits in this state for four RXRECCLK cycles and then goes to state SYNC_ACQUIRED, unless an invalid symbol is received, in which case the FSM goes to state LOSS_OF_SYNC.

LOSS_OF_SYNC (RXLOSSOFSYNC = 10)

The FSM remains in this state until a comma is received, at which time it goes to state RESYNC. The FSM state appears on RXLOSSOFSYNC if RX_LOSS_OF_SYNC_FSM is TRUE. Otherwise, RXLOSSOFSYNC[1] is high if an invalid byte (not in table, or with disparity error) is received, and RXLOSSOFSYNC[0] is High when a channel bonding (alignment) sequence has just been written into the elastic buffer.

8B/10B Operation

The Rocket I/O transceiver has the ability to encode eight bits into a 10-bit serial stream using standard 8B/10B encoding. This guarantees a DC-balanced, edge-rich serial stream, facilitating DC- or AC-coupling and clock recovery. If the 8B/10B encoding is disabled, the data is sent through in 10-bit blocks. The 8B/10B-bypass signal is set to `1111` and the RX_DECODE_USE attribute must be set to FALSE. If the 8B/10B encoding is needed, the bypass signal must be set to `0000` and the RX_DECODE_USE must be set to TRUE. If this pair is not matched, the data is not received correctly. Figure 3-6 shows the encoding/decoding blocks of the transceiver and how the data passes through these blocks. Table 3-15, page 57, shows the significance of 8B/10B ports that change purpose depending on whether 8B/10B is bypassed or enabled.
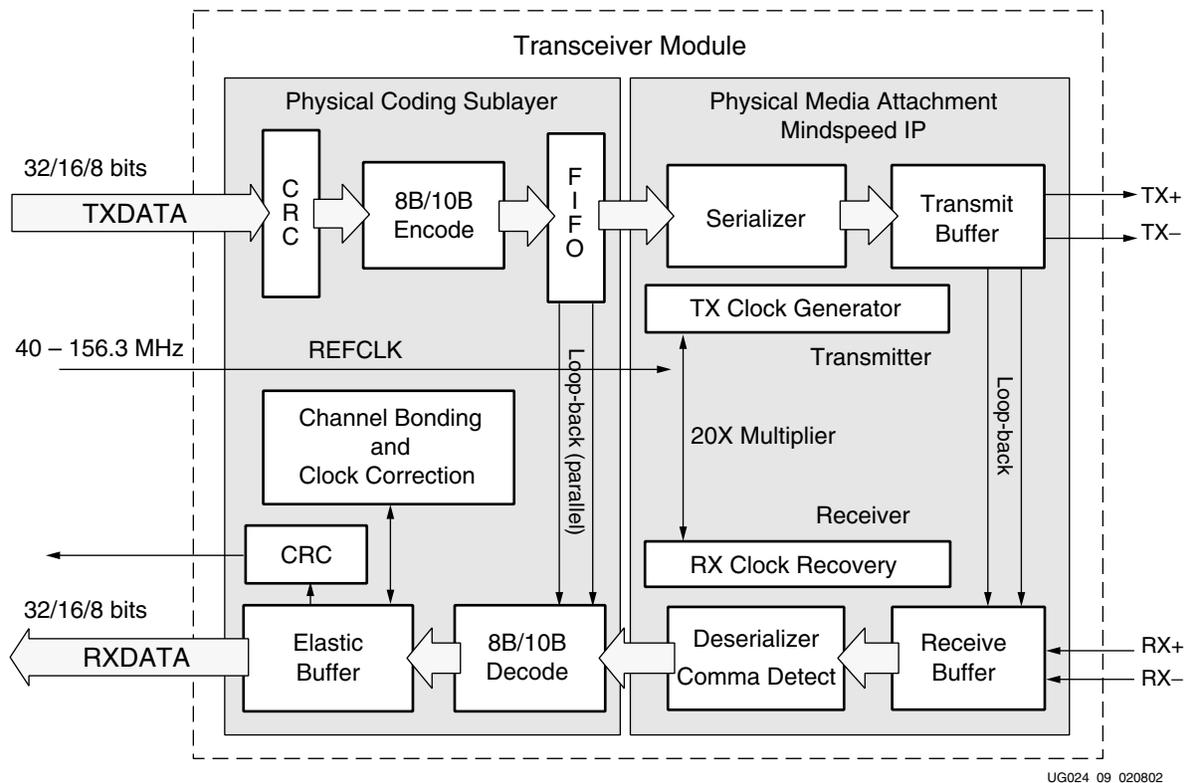
Figure 3-6: **8B/10B Data Flow**

Table 3-15: **8B/10B Bypassed Signal Significance**

		8B/10B Enabled	8B/10B Bypassed
TXBYPASS8B10B	1	N/A	Bypassed 8B/10B encoding
	0	Enabled 8B/10B	N/A
TXCHARISK	1	Transmitted byte is a K-character	Unused
	0	Transmitted byte is a data character	Unused
TXCHARDISPMODE		Disparity generation control	Part of 10-bit encoded byte: TXCHARDISPMODE[1], TXCHARDISPVAL[1], TXDATA[15:8], similar for other bytes
TXCHARDISPVAL		Disparity generation control	Part of 10-bit encoded byte: TXCHARDISPMODE[1], TXCHARDISPVAL[1], TXDATA[15:8], similar for other bytes
RXCHARISK		Indicates if character is a K-character	Part of 10-bit encoded byte: RXCHARISK[1], RXRUNDISP[1], RXDATA[15:8], similar for other bytes
RXRUNDISP		Indicates running disparity	Part of 10-bit encoded byte: RXCHARISK[1], RXRUNDISP[1], RXDATA[15:8], similar for other bytes

While the 8B/10B is enabled, the disparity of the serial transmission can be controlled with TXCHARDISPMODE and TXCHARDISPVAL. This is explained in Table 3-16. During the bypassing of the 8B/10B encoding, these ports become part of the 10-bit encoded data that the transceiver must transmit. See Figure 3-7 and Figure 3-8 for TX and RX data maps during 8B/10B bypass.

Table 3-16: **Running Disparity Modes with 8B/10B Enabled**

{TXCHARDISPMODE, TXCHARDISPVAL}	Function
00	Maintain running disparity normally.
01	Invert the normally generated running disparity before encoding this byte.
10	Set negative running disparity before encoding this byte.
11	Set positive running disparity before encoding this byte.

TXCHARDISPMODE[0]
TXCHARDISPVAL[0]
TXDATA[7] . . .   . . . TXDATA[0]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

First transmitted   Last transmitted

UG024_10a_021102
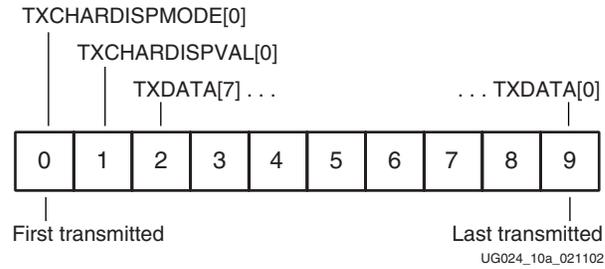
Figure 3-7: **10-Bit TX Data Map with 8B/10B Bypassed**

RXCHARISK[0]
RXRUNDISP[0]
RXDATA[7] . . .   . . . RXDATA[0]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

First received   Last received

UG024_10b_021102
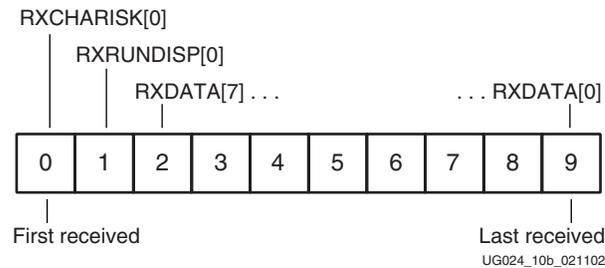
Figure 3-8: **10-Bit RX Data Map with 8B/10B Bypassed**

Vitesse Disparity Example

To support other protocols, the transceiver can affect the disparity mode of the serial data transmitted. For example, Vitesse channel-to-channel alignment protocol sends out:

```
K28.5+ K28.5+ K28.5- K28.5-
```

or

```
K28.5- K28.5- K28.5+ K28.5+
```

Instead of:

```
K28.5+ K28.5- K28.5+ K28.5-
```

or

```
K28.5- K28.5+ K28.5- K28.5+
```

The logic must assert TXCHARDISPVAL to cause the serial data to send out two negative running disparity characters.

Transmitting Vitesse Channel Bonding Sequence

```
TXBYPASS8B10B
| TXCHARISK
| | TXCHARDISPMODE
| | | TXCHARDISPVAL
| | | | TXDATA
| | | | |
0 1 0 0 10111100    K28.5+ (or K28.5-)
0 1 0 1 10111100    K28.5+ (or K28.5-)
0 1 0 0 10111100    K28.5- (or K28.5+)
0 1 0 1 10111100    K28.5- (or K28.5+)
```

The Rocket I/O core receives this data but must have the CHAN_BOND_SEQ set with the `disp_err` bit set High for the cases when TXCHARDISPVAL is set High during data transmission.

Receiving Vitesse Channel Bonding Sequence

On the RX side, the definition of the channel bonding sequence uses the `disp_err` bit to specify the flipped disparity.

```
                                    10-bit literal value
                                    | disp_err
                                    | | char_is_k
                                    | | | 8-bit_byte_value
                                    | | | |
                    CHAN_BOND_SEQ_1_1 = 0 0 1 10111100    matches K28.5+ (or K28.5-)
                    CHAN_BOND_SEQ_1_2 = 0 1 1 10111100    matches K28.5+ (or K28.5-)
                    CHAN_BOND_SEQ_1_3 = 0 0 1 10111100    matches K28.5- (or K28.5+)
                    CHAN_BOND_SEQ_1_4 = 0 1 1 10111100    matches K28.5- (or K28.5+)
                    CHAN_BOND_SEQ_LEN = 4
                    CHAN_BOND_SEQ_2_USE = False
```

Status Signals

Whether the 8B/10B encoding is enabled or disabled, there are several status signals for error indication. If an invalid K-character is sent to the transceiver, the TXKERR transitions High. This can produce several receive errors. These receive errors are RXNOTINTABLE or RUNDISPERR. RUNDISPERR transitions High if the incorrect disparity is received. RXNOTINTABLE determines if the incoming character is a valid character. These signals are meant to detect errors in the transmission data from incorrect framing. The **CRC Operation** section covers transmission data error detection caused by noise.

8B/10B Encoding

8B/10B encoding includes a set of Data characters and K-characters. Eight-bit values are coded into 10-bit values keeping the serial line DC balanced. K-characters are special Data characters designated with a CHARISK. K-characters are used for specific informative designations. Table 3-17 and Table 3-18 show the Data and K tables of valid characters.

Table 3-17:   **Valid Data Characters**

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D0.0	000 00000	100111 0100	011000 1011
D1.0	000 00001	011101 0100	100010 1011
D2.0	000 00010	101101 0100	010010 1011
D3.0	000 00011	110001 1011	110001 0100
D4.0	000 00100	110101 0100	001010 1011
D5.0	000 00101	101001 1011	101011 0100
D6.0	000 00110	011001 1011	011001 0100
D7.0	000 00111	111000 1011	000111 0100
D8.0	000 01000	111001 0100	000110 1011
D9.0	000 01001	100101 1011	011010 0100
D10.0	000 01010	010101 1011	010101 0100
D11.0	000 01011	110100 1011	110100 0100

Table 3-17: **Valid Data Characters** *(Continued)*

Data Byte Name	Bits HGF EDCBA	Current RD − abcdei fghj	Current RD + abcdei fghj
D12.0	000 01100	001101 1011	001101 0100
D13.0	000 01101	101100 1011	101100 0100
D14.0	000 01110	011100 1011	011100 0100
D15.0	000 01111	010111 0100	101000 1011
D16.0	000 10000	011011 0100	100100 1011
D17.0	000 10001	100011 1011	100011 0100
D18.0	000 10010	010011 1011	010011 0100
D19.0	000 10011	110010 1011	110010 0100
D20.0	000 10100	001011 1011	001011 0100
D21.0	000 10101	101010 1011	101010 0100
D22.0	000 10110	011010 1011	011010 0100
D23.0	000 10111	111010 0100	000101 1011
D24.0	000 11000	110011 0100	001100 1011
D25.0	000 11001	100110 1011	100110 0100
D26.0	000 11010	010110 1011	010110 0100
D27.0	000 11011	110110 0100	001001 1011
D28.0	000 11100	001110 1011	001110 0100
D29.0	000 11101	101110 0100	010001 1011
D30.0	000 11110	011110 0100	100001 1011
D31.0	000 11111	101011 0100	010100 1011
D0.1	001 00000	100111 1001	011000 1001
D1.1	001 00001	011101 1001	100010 1001
D2.1	001 00010	101101 1001	010010 1001
D3.1	001 00011	110001 1001	110001 1001
D4.1	001 00100	110101 1001	001010 1001
D5.1	001 00101	101001 1001	101011 1001
D6.1	001 00110	011001 1001	011001 1001
D7.1	001 00111	111000 1001	000111 1001
D8.1	001 01000	111001 1001	000110 1001
D9.1	001 01001	100101 1001	011010 1001
D10.1	001 01010	010101 1001	010101 1001
D11.1	001 01011	110100 1001	110100 1001
D12.1	001 01100	001101 1001	001101 1001

Table 3-17: **Valid Data Characters** *(Continued)*

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D13.1	001 01101	101100 1001	101100 1001
D14.1	001 01110	011100 1001	011100 1001
D15.1	001 01111	010111 1001	101000 1001
D16.1	001 10000	011011 1001	100100 1001
D17.1	001 10001	100011 1001	100011 1001
D18.1	001 10010	010011 1001	010011 1001
D19.1	001 10011	110010 1001	110010 1001
D20.1	001 10100	001011 1001	001011 1001
D21.1	001 10101	101010 1001	101010 1001
D22.1	001 10110	011010 1001	011010 1001
D23.1	001 10111	111010 1001	000101 1001
D24.1	001 11000	110011 1001	001100 1001
D25.1	001 11001	100110 1001	100110 1001
D26.1	001 11010	010010 1001	010110 1001
D27.1	001 11011	110110 1001	001001 1001
D28.1	001 11100	001110 1001	001110 1001
D29.1	001 11101	101110 1001	010001 1001
D30.1	001 11110	011110 1001	100001 1001
D31.1	001 11111	101011 1001	010100 1001
D0.2	010 00000	100111 0101	011000 0101
D1.2	010 00001	011101 0101	100010 0101
D2.2	010 00010	101101 0101	010010 0101
D3.2	010 00011	110001 0101	110001 0101
D4.2	010 00100	110101 0101	001010 0101
D5.2	010 00101	101001 0101	101011 0101
D6.2	010 00110	011001 0101	011001 0101
D7.2	010 00111	111000 0101	000111 0101
D8.2	010 01000	111001 0101	000110 0101
D9.2	010 01001	100101 0101	011010 0101
D10.2	010 01010	010101 0101	010101 0101
D11.2	010 01011	110100 0101	110100 0101
D12.2	010 01100	001101 0101	001101 0101
D13.2	010 01101	101100 0101	101100 0101

Table 3-17: **Valid Data Characters** *(Continued)*

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D14.2	010 01110	011100 0101	011100 0101
D15.2	010 01111	010111 0101	101000 0101
D16.2	010 10000	011011 0101	100100 0101
D17.2	010 10001	100011 0101	100011 0101
D18.2	010 01010	010011 0101	010011 0101
D19.2	010 10011	110010 0101	110010 0101
D20.2	010 10100	001011 0101	001011 0101
D21.2	010 10101	101010 0101	101010 0101
D22.2	010 10110	011010 0101	011010 0101
D23.2	010 10111	111010 0101	000101 0101
D24.2	010 11000	110011 0101	001100 0101
D25.2	010 11001	100110 0101	100110 0101
D26.2	010 11010	010010 0101	010110 0101
D27.2	010 11011	110110 0101	001001 0101
D28.2	010 11100	001110 0101	001110 0101
D29.2	010 11101	101110 0101	010001 0101
D30.2	010 11110	011110 0101	100001 0101
D31.2	010 11111	101011 0101	010100 0101
D0.3	000 00000	100111 0011	011000 1100
D1.3	011 00001	011101 0011	100010 1100
D2.3	011 00010	101101 0011	010010 1100
D3.3	011 00011	110001 1100	110001 0011
D4.3	011 00100	110101 0011	001010 1100
D5.3	011 00101	101001 1100	101011 0011
D6.3	011 00110	011001 1100	011001 0011
D7.3	011 00111	111000 1100	000111 0011
D8.3	011 01000	111001 0011	000110 1100
D9.3	011 01001	100101 1100	011010 0011
D10.3	011 01010	010101 1100	010101 0011
D11.3	011 01011	110100 1100	110100 0011
D12.3	011 01100	001101 1100	001101 0011
D13.3	011 01101	101100 1100	101100 0011
D14.3	011 01110	011100 1100	011100 0011

Table 3-17: **Valid Data Characters** *(Continued)*

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D15.3	011 01111	010111 0011	101000 1100
D16.3	011 10000	011011 0011	100100 1100
D17.3	011 10001	100011 1100	100011 0011
D18.3	011 10010	010011 1100	010011 0011
D19.3	011 10011	110010 1100	110010 0011
D20.3	011 10100	001011 1100	001011 0011
D21.3	011 10101	101010 1100	101010 0011
D22.3	011 10110	011010 1100	011010 0011
D23.3	011 10111	111010 0011	000101 1100
D24.3	011 11000	110011 0011	001100 1100
D25.3	011 11001	100110 1100	100110 0011
D26.3	011 11010	010110 1100	010110 0011
D27.3	011 11011	110110 0011	001001 1100
D28.3	011 11100	001110 1100	001110 0011
D29.3	011 11101	101110 0011	010001 1100
D30.3	011 11110	011110 0011	100001 1100
D31.3	011 11111	101011 0011	010100 1100
D0.4	100 00000	100111 0010	011000 1101
D1.4	100 00001	011101 0010	100010 1101
D2.4	100 00010	101101 0010	010010 1101
D3.4	100 00011	110001 1101	110001 0010
D4.4	100 00100	110101 0010	001010 1101
D5.4	100 00101	101001 1101	101011 0010
D6.4	100 00110	011001 1101	011001 0010
D7.4	100 00111	111000 1101	000111 0010
D8.4	100 01000	111001 0010	000110 1101
D9.4	100 01001	100101 1101	011010 0010
D10.4	100 01010	010101 1101	010101 0010
D11.4	100 01011	110100 1101	110100 0010
D12.4	100 01100	001101 1101	001101 0010
D13.4	100 01101	101100 1101	101100 0010
D14.4	100 01110	011100 1101	011100 0010
D15.4	100 01111	010111 0010	101000 1101

Table 3-17: **Valid Data Characters** *(Continued)*

Data Byte Name	Bits HGF EDCBA	Current RD − abcdei fghj	Current RD + abcdei fghj
D16.4	100 10000	011011 0010	100100 1101
D17.4	100 10001	100011 1101	100011 0010
D18.4	100 10010	010011 1101	010011 0010
D19.4	100 10011	110010 1101	110010 0010
D20.4	100 10100	001011 1101	001011 0010
D21.4	100 10101	101010 1101	101010 0010
D22.4	100 10110	011010 1101	011010 0010
D23.4	100 10111	111010 0010	000101 1101
D24.4	100 11000	110011 0010	001100 1101
D25.4	100 11001	100110 1101	100110 0010
D26.4	100 11010	010010 1101	010110 0010
D27.4	100 11011	110110 0010	001001 1101
D28.4	100 11100	001110 1101	001110 0010
D29.4	100 11101	101110 0010	010001 1101
D30.4	100 11110	011110 0010	100001 1101
D31.4	100 11111	101011 0010	010100 1101
D0.5	101 00000	100111 1010	011000 1010
D1.5	101 00001	011101 1010	100010 1010
D2.5	101 00010	101101 1010	010010 1010
D3.5	101 00011	110001 1010	110001 1010
D4.5	101 00100	110101 101	001010 1010
D5.5	101 00101	101001 1010	101011 1010
D6.5	101 00110	011001 1010	011001 1010
D7.5	101 00111	111000 1010	000111 1010
D8.5	101 01000	111001 1010	000110 1010
D9.5	101 01001	100101 1010	011010 1010
D10.5	101 01010	010101 1010	010101 1010
D11.5	101 01011	110100 1010	110100 1010
D12.5	101 01100	001101 1010	001101 1010
D13.5	101 01101	101100 1010	101100 1010
D14.5	101 01110	011100 1010	011100 1010
D15.5	101 01111	010111 1010	101000 1010
D16.5	101 10000	011011 1010	100100 1010

Table 3-17: **Valid Data Characters** *(Continued)*

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D17.5	101 10001	100011 1010	100011 1010
D18.5	101 01010	010011 1010	010011 1010
D19.5	101 10011	110010 1010	110010 1010
D20.5	101 10100	001011 1010	001011 1010
D21.5	101 10101	101010 1010	101010 1010
D22.5	101 10110	011010 1010	011010 1010
D23.5	101 10111	111010 1010	000101 1010
D24.5	101 11000	110011 1010	001100 1010
D25.5	101 11001	100110 1010	100110 1010
D26.5	101 11010	010010 1010	010110 1010
D27.5	101 11011	110110 1010	001001 1010
D28.5	101 11100	001110 1010	001110 1010
D29.5	101 11101	101110 1010	010001 1010
D30.5	101 11110	011110 1010	100001 1010
D31.5	101 11111	101011 1010	010100 1010
D0.6	110 00000	100111 0110	011000 0110
D1.6	110 00001	011101 0110	100010 0110
D2.6	110 00010	101101 0110	010010 0110
D3.6	110 00011	110001 0110	110001 0110
D4.6	110 00100	110101 0110	001010 0110
D5.6	110 00101	101001 0110	101011 0110
D6.6	110 00110	011001 0110	011001 0110
D7.6	110 00111	111000 0110	000111 0110
D8.6	110 01000	111001 0110	000110 0110
D9.6	110 01001	100101 0110	011010 0110
D10.6	110 01010	010101 0110	010101 0110
D11.6	110 01011	110100 0110	110100 0110
D12.6	110 01100	001101 0110	001101 0110
D13.6	110 01101	101100 0110	101100 0110
D14.6	110 01110	011100 0110	011100 0110
D15.6	110 01111	010111 0110	101000 0110
D16.6	110 10000	011011 0110	100100 0110
D17.6	110 10001	100011 0110	100011 0110

Table 3-17: **Valid Data Characters** *(Continued)*

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D18.6	110 01010	010011 0110	010011 0110
D19.6	110 10011	110010 0110	110010 0110
D20.6	110 10100	001011 0110	001011 0110
D21.6	110 10101	101010 0110	101010 0110
D22.6	110 10110	011010 0110	011010 0110
D23.6	110 10111	111010 0110	000101 0110
D24.6	110 11000	110011 0110	001100 0110
D25.6	110 11001	100110 0110	100110 0110
D26.6	110 11010	010010 0110	010110 0110
D27.6	110 11011	110110 0110	001001 0110
D28.6	110 11100	001110 0110	001110 0110
D29.6	110 11101	101110 0110	010001 0110
D30.6	110 11110	011110 0110	100001 0110
D31.6	110 11111	101011 0110	010100 0110
D0.7	111 00000	100111 0001	011000 1110
D1.7	111 00001	011101 0001	100010 1110
D2.7	111 00010	101101 0001	010010 1110
D3.7	111 00011	110001 1110	110001 0001
D4.7	111 00100	110101 0001	001010 1110
D5.7	111 00101	101001 1110	101011 0001
D6.7	111 00110	011001 1110	011001 0001
D7.7	111 00111	111000 1110	000111 0001
D8.7	111 01000	111001 0001	000110 1110
D9.7	111 01001	100101 1110	011010 0001
D10.7	111 01010	010101 1110	010101 0001
D11.7	111 01011	110100 1110	110100 1000
D12.7	111 01100	001101 1110	001101 0001
D13.7	111 01101	101100 1110	101100 1000
D14.7	111 01110	011100 1110	011100 1000
D15.7	111 01111	010111 0001	101000 1110
D16.7	111 10000	011011 0001	100100 1110
D17.7	111 10001	100011 0111	100011 0001
D18.7	111 10010	010011 0111	010011 0001

Table 3-17: Valid Data Characters *(Continued)*

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D19.7	111 10011	110010 1110	110010 0001
D20.7	111 10100	001011 0111	001011 0001
D21.7	111 10101	101010 1110	101010 0001
D22.7	111 10110	011010 1110	011010 0001
D23.7	111 10111	111010 0001	000101 1110
D24.7	111 11000	110011 0001	001100 1110
D25.7	111 11001	100110 1110	100110 0001
D26.7	111 11010	010110 1110	010110 0001
D27.7	111 11011	110110 0001	001001 1110
D28.7	111 11100	001110 1110	001110 0001
D29.7	111 11101	101110 0001	010001 1110
D30.7	111 11110	011110 0001	100001 1110
D31.7	111 11111	101011 0001	010100 1110

Table 3-18: **Valid Control "K" Characters**

Special Code Name	Bits HGF EDCBA	Current RD − abcdei fghj	Current RD + abcdei fghj
K28.0	000 11100	001111 0100	110000 1011
K28.1	001 11100	001111 1001	110000 0110
K28.2	010 11100	001111 0101	110000 1010
K28.3	011 11100	001111 0011	110000 1100
K28.4	100 11100	001111 0010	110000 1101
K28.5	101 11100	001111 1010	110000 0101
K28.6	110 11100	001111 0110	110000 1001
K28.7	111 11100	001111 1000	110000 0111
K23.7	111 10111	111010 1000	000101 0111
K27.7	111 11011	110110 1000	001001 0111
K29.7	111 11101	101110 1000	010001 0111
K30.7	111 11110	011110 1000	100001 0111

8B/10B Serial Output Format

The 8B/10B encoding translates a 8-bit parallel data byte to be transmitted into a 10-bit serial data stream. This conversion and data alignment are shown in Figure 3-9. The serial port transmits the least significant bit of the 10-bit data "a" first and proceeds to "j". This allows data to be read and matched to the form shown in Table 3-17.

UG024_10_021102
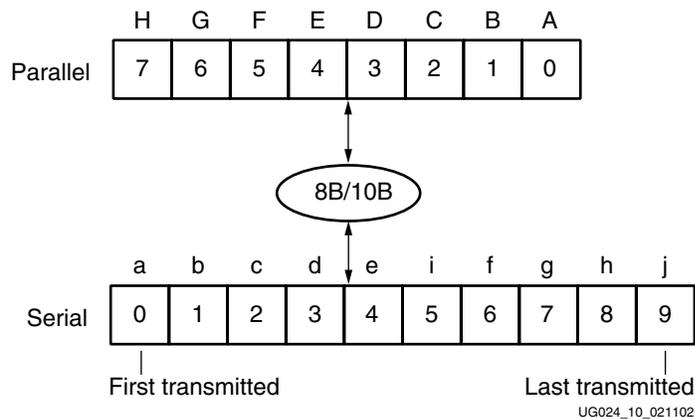
Figure 3-9: **8B/10B Parallel to Serial Conversion**

The serial data bit sequence is dependent on the width of the parallel data. The most significant byte is always sent first regardless of the whether 1-byte, 2-byte, or 4-byte paths are used. The least significant byte is always last. Figure 3-10 shows a case when the serial data corresponds to each byte of the parallel data. TXDATA [31:24] is serialized and sent out first followed by TXDATA [23:16], TXDATA [15:8], and finally TXDATA [7:0]. The 2-byte path transmits TXDATA [15:8] and then TXDATA [7:0].
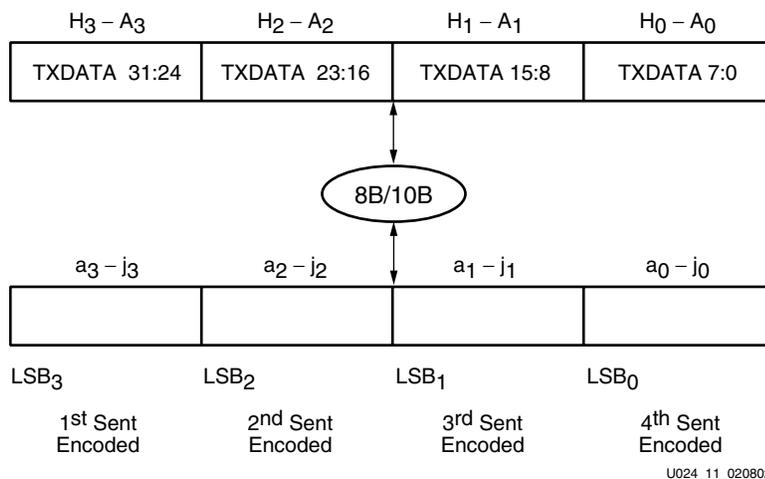
Figure 3-10: **4-Byte Serial Structure**

HDL Code Examples: Transceiver Bypassing of 8B/10B Encoding

8B/10B encoding can be bypassed by the transceiver. The TX8B10BBYPASS is set to `1111`; the RXDECODE attribute is set to "FALSE" to create the extra two bits needed for a 10-bit data bus; and TXCHARDISPMODE, TXCHARDISPVAL, RXCHARISK, and RXRUNDISP are added to the 8-bit data bus.

Availability for download of code examples with 8B/10B bypassing is planned for a later date.

CRC Operation

Cyclic Redundancy Check (CRC) is a procedure to detect errors in the received data. There are four possible CRC modes, USER_MODE, ETHERNET, INFINIBAND, and FIBRE_CHAN. These are only modifiable for the GT_XAUI and GT_CUSTOM. Each mode has a start-of-packet (SOP) and end-of-packet (EOP) setting to determine where to start and end the CRC monitoring. USER_MODE allows the user to define the SOP and EOP by setting the CRC_START_OF_PKT and CRC_END_OF_PKT to one of the valid K-characters (Table 3-18). The CRC is controlled by RX_CRC_USE and TX_CRC_USE. Whenever these attributes are set to TRUE, CRC is used. A CRC error can be "forced" with the use of TXFORCECRCERR. This causes TX_CRC_FORCE_VALUE to be XORed with the computed CRC, to test the CRC error logic.

CRC Generation

Rocket I/O transceivers support a 32-bit invariant CRC (fixed 32-bit polynomial shown below) for Gigabit Ethernet, Fibre Channel, Infiniband, and user-defined modes.

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$

The CRC recognizes the SOP (Start of Packet), EOP (End of Packet), and other packet features to identify the beginning and end of data. These SOP and EOP are defined by the

mode, except in the case where CRC_MODE is USER_DEFINED. The user-defined mode uses CRC_START_OF_PKT and CRC_END_OF_PKT to define SOP and EOP.

The transmitter computes 4-byte CRC on the packet data between the SOP and EOP (excluding the CRC placeholder bytes). The transmitter inserts the computed CRC just before the EOP. The transmitter modifies trailing Idles or EOP if necessary to generate correct running disparity for Gigabit Ethernet and FibreChannel. The receiver recomputes CRC and verifies it against the inserted CRC. Figure 3-11 shows the packet format for CRC generation. The empty boxes are only used in certain protocols (Ethernet). The user logic must create a four-byte placeholder for the CRC; otherwise, data is overwritten.

| SOP | ⋯ | Data | CRC | ⋯ | EOP | ⋯ | Idle |

4 Bytes

UG024_07_021102

Figure 3-11: **CRC Packet Format**

CRC Latency

Enabling CRC increases the transmission latency from TXDATA to TXP and TXN. The enabling of CRC does not affect the latency from RXP and RXN to RXDATA. The typical and maximum latencies, expressed in TXUSRCLK2/RXUSRCLK2 cycles, are shown in Table 3-19. For timing diagrams expressing these relationships, please see Module 3 of the **Virtex-II Pro Data Sheet**.

Table 3-19: **Effects of CRC on Transceiver Latency**

	TXDATA to TXP and TXN, in TXUSRCLK2 Cycles		RXP and RXN to RXDATA, in RXUSRCLK2 Cycles	
	Typical	**Maximum**	**Typical**	**Maximum**
CRC Disabled	8	11	25	42
CRC Enabled	14	17	25	42

CRC Limitations

There are several limitations to the Rocket I/O CRC. First, CRC is not supported in byte-striped data. If byte-striped (channel bonding) is required, CRC must be computed in CLBs prior to the byte-striping. The CRC support of Infiniband is incomplete, because the 16-bit variant CRC must be done in the CLBs making the transceiver core CRC function redundant. For this case, set TX_CRC_USE = FALSE.

CRC Modes

The Rocket I/O transceiver has four CRC modes: USER_MODE, FIBRECHANNEL, ETHERNET, and INFINIBAND. These CRC modes are briefly explained below.

USER_MODE

USER_MODE is the simplest CRC methodology. The CRC checks for the SOP and EOF, calculates CRC on the data, and leaves the four remainders directly before the EOP. The CRC form for the user-defined mode is shown in Figure 3-12, along with the timing for asserting RXCHECKINGCRC and RXCRCERR High with respect to the incoming data.

TXCRCFORCEERR and RXCRCERR are both useful during testing. When using CRC, testing the CRC logic can be done by setting CRCFORCEERR High to incorporate an error into the data that is transmitted. When the data is received in a testing mode, such as serial loopback, the data is "corrupted" and the receiver should signal an error with the use of

RXCRCERR when the RXCHECKINGCRC is asserted High. User logic determines the procedure when a RXCRCERR occurs.

FIBRECHANNEL

The Fibre Channel CRC is similar to the USER_MODE CRC (Figure 3-12) with one exception: In FibreChannel, SOP and EOP are the protocol delimiters, while USER_MODE uses the two attributes CRC_START_OF_PKT and CRC_END_OF_PKT to define SOP and EOP. Both USER_MODE and Fibre Channel, however, disregard the SOP and EOP in CRC computation.

SOP	DATA	R_0	R_1	R_2	R_3	EOP

RXCHECKINGCRC

RXCRCERR
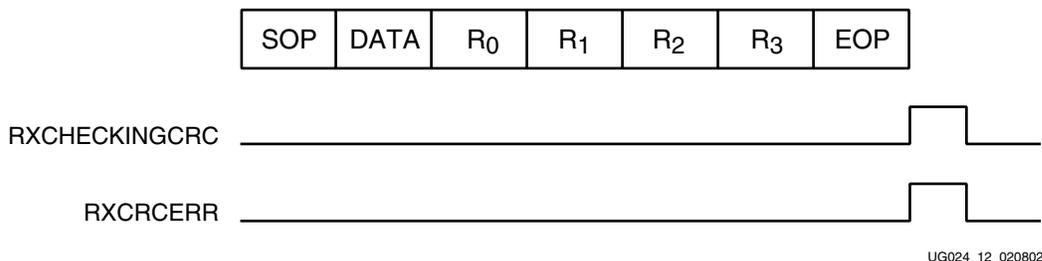
UG024_12_020802

Figure 3-12: **USER_MODE / FIBRE_CHANNEL Mode**

ETHERNET

The Ethernet CRC is more complex (Figure 3-13). The SOP, EOP, and Preamble are neglected by the CRC. The extension bytes are special "K" characters in special cases. The extension bytes are untouched by the CRC as are the Trail bits, which are added to maintain packet length.

SOP	Preamble	SOF	DATA	R_0	R_1	R_2	R_3	Extn Bits	EOP	Trail Bits

n Bytes   2 to 3 Bytes
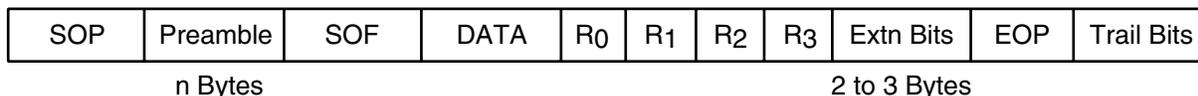
UG024_13_020802

Figure 3-13: **Ethernet Mode**

INFINIBAND

The Infiniband CRC is the most complex mode, and is not supported in the CRC generator. Infiniband CRC contains two computation types: an invariant 32-bit CRC, the same as in Ethernet protocol; and a variant 16-bit CRC, which is not supported in the hard core. Infiniband CRC must be implemented entirely in the FPGA fabric.

There are also two Infiniband Architecture (IBA) packets, a local and a global. Both of these IBA packets are shown in Figure 3-14.

Local IBA

SOP	LRH	BTH	Packet Payload	R_0	R_1	R_2	R_3	Variant CRC	EOP

Global IBA

SOP	LRH	GRH	BTH	Packet Payload	R_0	R_1	R_2	R_3	Variant CRC	EOP
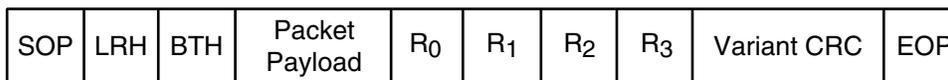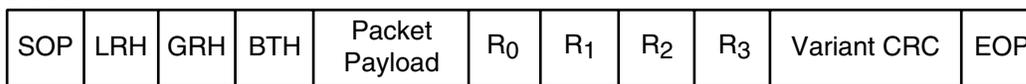
UG024_14_020802

Figure 3-14: **Infiniband Mode**

The CRC is calculated with certain bits masked in LRH and GRH, depending on whether the packet is local or global. The size of these headers is shown in Table 3-20.

Table 3-20: **Global and Local Headers**

Packet	Description	Size
LRH	Local Routing Header	8 Bytes
GRH	Global Routing Header	40 Bytes
BTH	IBA Transport Header	12 Bytes

The CRC checks the LNH (Link Next Header) of the LRH. LRH is shown in Figure 3-15, along with the bits the CRC uses to evaluate the next packet.

B_0	$B1_7 - B1_0$	B_2	B_3	B_4	B_5	B_6	B_7

$B1_1, B1_0$ 1 1 IBA Global Packet
               1 0 IBA Local Packet
               0 1 Raw Packet (CRC does not insert remainder)
               0 0 Raw Packet (CRC does not insert remainder)

UG024_15_020802
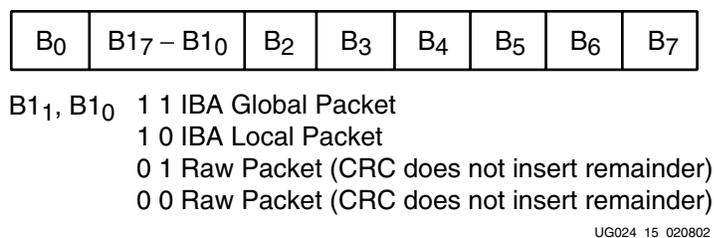
Figure 3-15: **Local Route Header**

Because of the complexity of the CRC algorithms and implementations, especially with Infiniband, a more in-depth discussion is beyond the scope of this manual.

Channel Bonding (Channel-to-Channel Alignment)

Channel bonding is the technique of tying several serial channels together to create one aggregate channel. Several channels are fed on the transmit side by one parallel bus and reproduced on the receive side as the identical parallel bus. The maximum number of serial differential pairs that can be bonded is 16. For implementation guidelines, see **Implementation Tools,** page 87.

Channel bonding allows those primitives that support it to send data over multiple "channels." Among these primitives are GT_CUSTOM, GT_INFINIBAND, GT_XAUI, and GT_AURORA. To "bond" channels together, there is always one "master." The other channels can either be a SLAVE_1_HOP or SLAVE_2_HOPs. SLAVE_1_HOP is a slave to a master that can also be daisy chained to a SLAVE_2_HOPs. A SLAVE_2_HOPs can only be a slave to a SLAVE_1_HOP and its CHBONDO does not connect to another transceiver. To designate a transceiver as a master or a slave, the attribute CHAN_BOND_MODE must be set to one of three designations: Master, SLAVE_1_HOP, or SLAVE_2_HOPs. To shut off channel bonding, set the transceiver attribute to "off." The possible values that can be used are shown in Table 3-21.

Table 3-21: **Bonded Channel Connections**

Mode	CHBONDI	CHBONDO
OFF	NA	NA
MASTER	NA	slave 1 CHBONDI
SLAVE_1_HOP	master CHBONDO	slave 2 CHBONDI
SLAVE_2_HOPS	slave 1 CHBONDO	NA

The channel bonding sequence is similar to the clock correction sequence. This sequence is set to the appropriate sequence for the primitives supporting channel bonding. The GT_CUSTOM is the only primitive allowing modification to the sequence. These sequences are comprised of one or two sequences of length up to 4 bytes each, as set by CHAN_BOND_SEQ_LEN and CHAN_BOND_SEQ_2_USE. Other control signals include the attributes:

- CHAN_BOND_WAIT
- CHAN_BOND_OFFSET
- CHAN_BOND_LIMIT
- CHAN_BOND_ONE_SHOT

Typical values for these attributes are:

CHAN_BOND_WAIT = 8
CHAN_BOND_OFFSET = CHAN_BOND_WAIT
CHAN_BOND_LIMIT = 2 x CHAN_BOND_WAIT

Lower values are not recommended. Use higher values only if channel bonding sequences are farther apart than 17 bytes.

Table 3-22 shows different settings for CHAN_BONDONE_SHOT and ENCHANSYNC in Master and Slave applications.

Table 3-22:   **Master/Slave Channel Bonding Attribute Settings**

	Master	**Slave**
CHAN_BOND_ONE_SHOT	TRUE or FALSE as desired	FALSE
ENCHANSYNC	Dynamic control as desired	Tie High

HDL Code Examples: Channel Bonding

Availability for download of code examples implementing channel bonding is planned for a later date.

Analog Design Considerations

Serial I/O Description

The Rocket I/O transceiver transmits and receives serial differential signals. This feature operates at a nominal supply voltage of 2.5 VDC. A serial differential pair consists of a true (V_P) and a complement (V_N) set of signals. The voltage difference represents the transferred data. Thus: $V_P - V_N = V_{DATA}$. Differential switching is performed at the crossing of the two complementary signals. Therefore, no separate reference level is needed.

A graphical representation of this concept is shown in Figure 4-1.

U024_06_020802
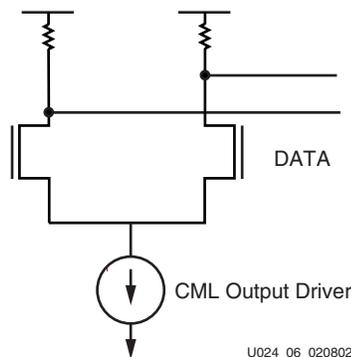
Figure 4-1:   **Differential Amplifier**

The Rocket I/O transceiver is implemented in Current Mode Logic (CML). A CML output consists of transistors configured as shown in Figure 4-1. CML uses a positive supply and offers easy interface requirements. In this configuration, both legs of the driver, V_P and V_N, sink current, with one leg always sinking more current than its complement. The CML output consists of a differential pair with 50Ω (or, optionally, 75Ω) source resistors. The signal swing is created by switching the current in a common-drain differential pair.

The differential transmitter specification is shown in Table 4-1, page 76.

Table 4-1: **Differential Transmitter Parameters**

Parameter		Min	Typ	Max	Units	Conditions
V_{OUT}	Serial output differential peak to peak (TXP/TXN)	800		1600	mV	Output differential voltage is programmable
V_{TTX}	Output termination voltage supply	1.8		2.8	V	
V_{TCM}	Common mode output voltage range	1.5		2.5	V	
V_{ISKEW}	Differential output skew			15	ps	

Pre-emphasis Techniques

In pre-emphasis, the initial differential voltage swing is boosted to create a stronger rising or falling waveform. This method compensates for high frequency loss in the transmission media that would otherwise limit the magnitude of this waveform. The effects of pre-emphasis are shown in four scope screen captures, Figure 4-2 through Figure 4-5 on the pages following. The STRONG notation in Figure 4-3 is used to show that the waveform is greater in voltage magnitude, at this point, than the LOGIC or normal level (i.e., no pre-emphasis).

A second characteristic of Rocket I/O transceiver pre-emphasis is that the STRONG level is reduced after some time to the LOGIC level, thereby minimizing the voltage swing necessary to switch the differential pair into the opposite state.

Lossy transmission lines cause the dissipation of electrical energy. This pre-emphasis technique extends the distance that signals can be driven down lossy line media and increases the signal-to-noise ratio at the receiver.

The four levels of pre-emphasis are shown in Table 4-2.

Table 4-2: **Pre-emphasis Values**

Attribute Values	Emphasis (%)
0	10
1	20
2	25
3	33
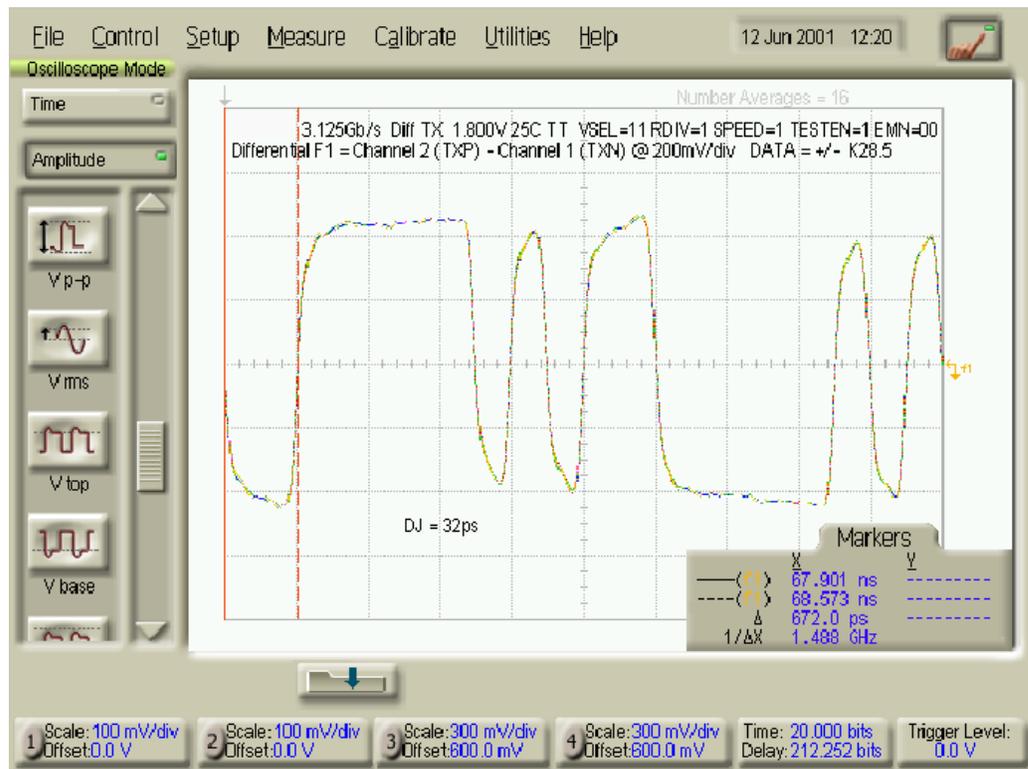
Figure 4-2: **Alternating K28.5+ with No Pre-Emphasis**
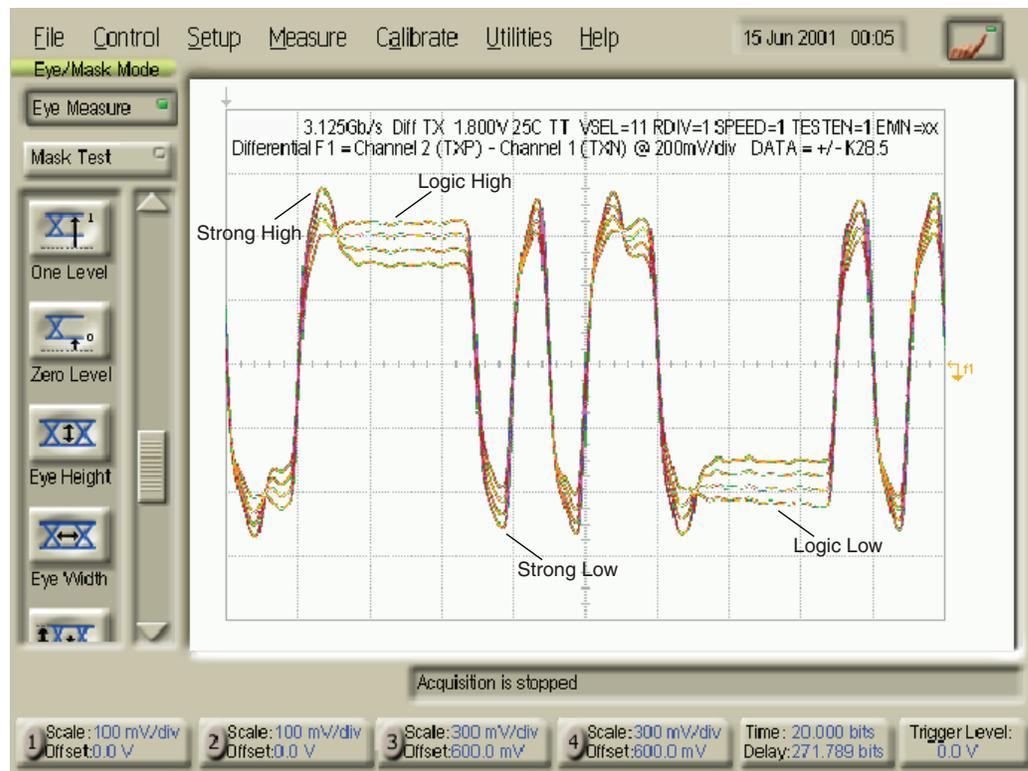
Figure 4-3: **K28.5+ with Pre-Emphasis**
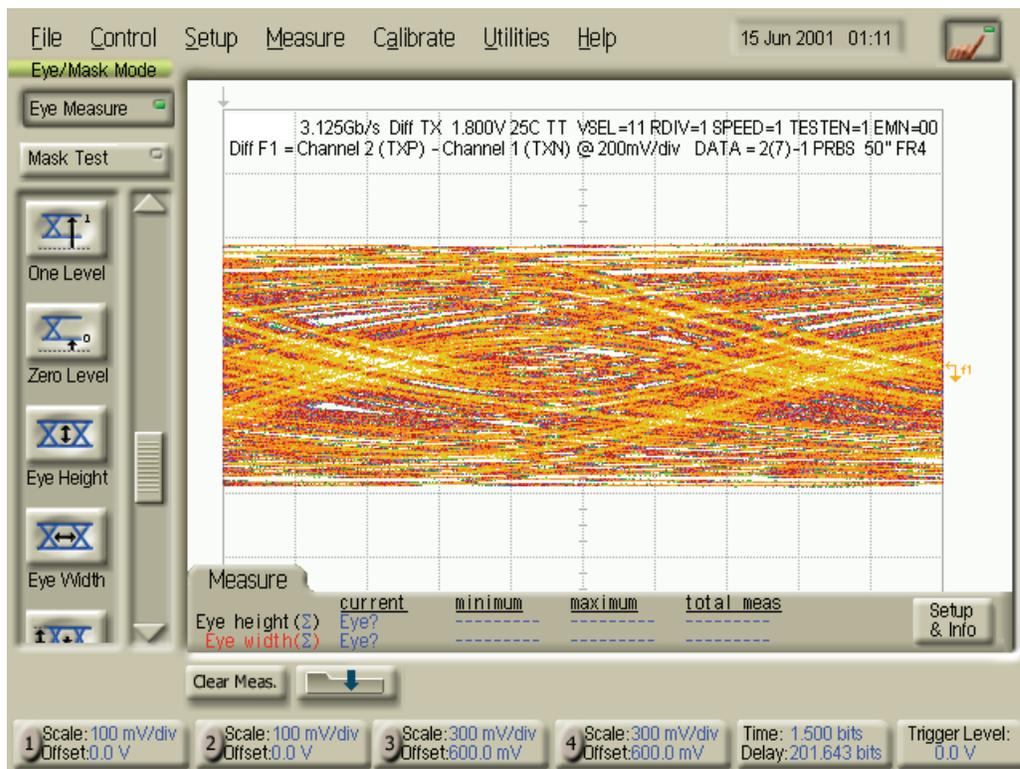
Figure 4-4: **Eye Diagram: without Pre-Emphasis**
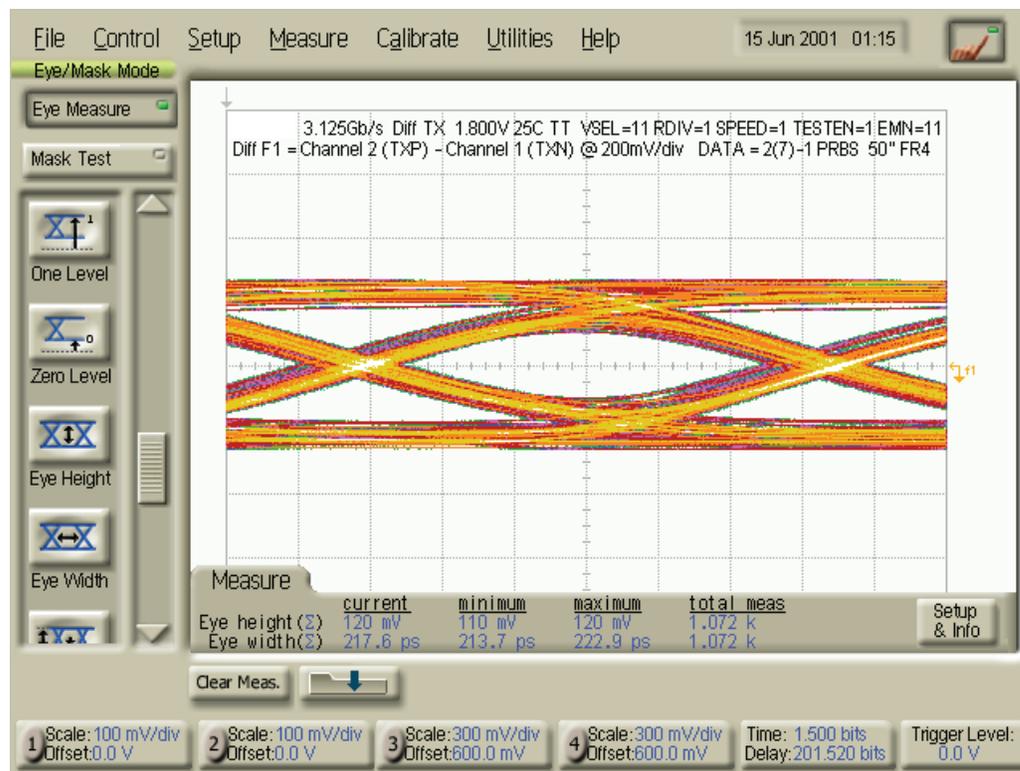
Figure 4-5: **Eye Diagram: with 30% Pre-Emphasis**

Differential Receiver

The differential receiver accepts the V_P and V_N signals, carrying out the difference calculation $V_P - V_N$ electronically.

All input data must be differential and nominally biased to a common mode voltage of 0.5 V – 2.5 V, or AC coupled. Internal terminations provide for simple 50Ω or 75Ω transmission line connection.

The differential receiver parameters are shown in Table 4-3.

Table 4-3: **Differential Receiver Parameters**

Parameter		Min	Typ	Max	Units	Conditions
V_{IN}	Serial input differential peak to peak (RXP/RXN)	175		1,000	mV	
V_{ICM}	Common mode input voltage range	500		2500	mV	
T_{ISKEW}	Differential input skew			75	ps	
T_{JTOL}	Receive data total jitter tolerance (peak to peak)			0.65	UI[1]	
T_{DJTOL}	Receive data deterministic jitter tolerance (peak to peak)			0.41	UI	

Notes:
1. UI = Unit Interval

Jitter

Jitter is defined as the short-term variations of significant instants of a signal from their ideal positions in time (ITU). Jitter is typically expressed in a decimal fraction of Unit Interval (UI), e.g. 0.3 UI.

Total Jitter (DJ + RJ)

Deterministic Jitter (DJ)

DJ is data pattern dependant jitter, attributed to a unique source (e.g., Inter Symbol Interference (ISI) due to loss effects of the media). DJ is linearly additive.

Random Jitter (RJ)

RJ is due to stochastic sources, such as substrate, power supply, etc. RJ is additive as the sum of squares, and follows a bell curve.

Clock and Data Recovery

The serial transceiver input is locked to the input data stream through Clock and Data Recovery (CDR), a built-in feature of the Rocket I/O transceiver. CDR keys off the rising and falling edges of incoming data and derives a clock that is representative of the incoming data rate.

The derived clock, RXRECCLK, is presented to the FPGA fabric at 1/20th the incoming data rate. This clock is generated and locked to as long as it remains within the specified component range. This range is shown in Table 4-4.

Table 4-4: **CDR Parameters**

Parameter		Min	Typ	Max	Units	Conditions
Frequency Range	Serial input differential (RXP/RXN)	175		1,000	MHz	Peak-to-peak
Frequency Offset					ppm	
T_{DCREF}	REFCLK duty cycle	45	50	55	%	
T_{RCLK}/T_{FCLK}	REFCLK rise and fall time (see Virtex-II Pro Data Sheet, Module 3)			75	ps	Between 20% and 80% voltage levels
T_{GJTT}	REFCLK total jitter			40	ps	Peak-to-peak
T_{LOCK}	Clock recovery frequency acquisition time		10		µs	
T_{UNLOCK}					cycles	

A sufficient number of transitions must be present in the data stream for CDR to work properly. The CDR circuit is guaranteed to work with 8B/10B encoding. Further, CDR requires approximately 5,000 transitions upon power-up to guarantee locking to the incoming data rate. Once lock is achieved, up to 75 missing transitions can be tolerated before lock to the incoming data stream is lost.

Care must be taken if a custom serial data stream is engineered so that the transition frequency rate requirement of 8B/10B encoding is met. An additional feature of CDR is its ability to accept an external precision clock, REFCLK, as an optional input. REFCLK acts either to clock incoming data or to assist in synchronizing the derived RXRECCLK.

For further clarity, the TXUSRCLK is used to clock data from the FPGA core to the TX FIFO. The FIFO depth accounts for the slight phase difference between these two clocks. If the clocks are locked in frequency, then the FIFO acts much like a pass-through buffer.

PCB Design Requirements

In order to ensure reliable operation of the Rocket I/O transceivers, certain requirements must be met by the designer. This section outlines these requirements governing power filtering networks, high-speed differential signal traces, and reference clocks. Any designs that do not adhere to these requirements will not be supported by Xilinx, Inc.

Power Filtering

Each Rocket I/O transceiver has five power supply pins, all of which are sensitive to noise. Table 4-5 summarizes the power supply pins, their names, and associated voltages.

Table 4-5:   **Transceiver Power Supplies**

Supply	2.5V	1.5V - 2.8V	Description
AVCCAUXRX	X		Analog RX supply
AVCCAUXTX	X		Analog TX supply
VTRX		X	RX termination supply
VTTX		X	TX termination supply
GNDA			Analog ground for transmit and receive analog supplies

To operate properly, the Rocket I/O transceiver requires a certain level of noise isolation from surrounding noise sources. For this reason, it is required that dedicated voltage regulators be used to power the Rocket I/O circuitry. These power supply circuits must not be shared with any other supplies (including FPGA supplies V_{CCINT}, V_{CCO}, V_{CCAUX}, and V_{REF}). Voltage regulators may be shared among transceiver power supplies of the same voltage.

The required voltage regulator is the Linear Technology LT1963-2.5 device. This regulator must be used in the circuit specified by the manufacturer. Figure 4-6 shows the schematic with values for a 2.5V supply, as would be used for AVCCAUXRX and AVCCAUXTX.

Refer to the manufacturer's Web page at **http://www.linear-tech.com** for further information about this device.
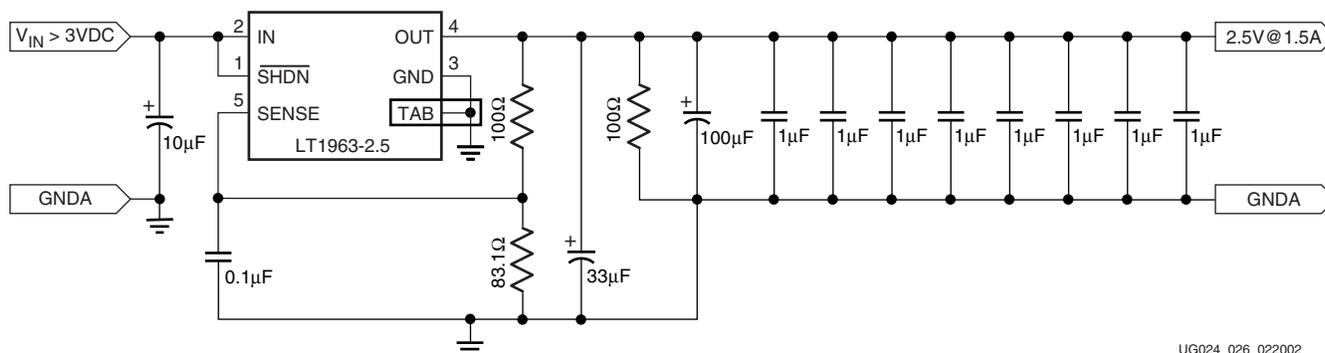
Figure 4-6:   **Power Supply Circuit Using LT1963 Regulator**

To achieve the necessary isolation from power supply noise, filter networks are required on the power supply pins. The topology of these capacitor and ferrite bead circuits is given in Figure 4-7.
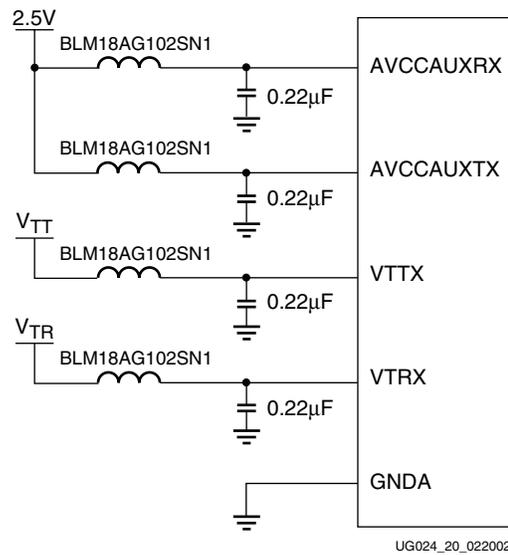
Figure 4-7: **Power Filtering Network for One Transceiver**

Each transceiver power pin requires one capacitor and one ferrite bead. The capacitors must be of value 0.22 µF in an 0603 SMT package of X7R dielectric material at 10% tolerance, rated to at least 5V. These capacitors must be placed within 1 cm of the pins they are bypassing. The ferrite bead is the Murata BLM18AG102SN1.

Figure 4-8 and Figure 4-9 show an example layout of the power filtering network for four transceivers. The device is in an ff672 package, which has eight transceivers total—four on the top edge and four on the bottom edge. Figure 4-8 shows the top PCB layer, with lands for the capacitors and ferrite beads of the VTTX and VTRX supplies. The ferrite beads are L1, L2, L3, L4, L9, L11, L12, and L21; the capacitors are C85, C90, C94, C96, C98, C100, C119, and C124. Figure 4-9 shows the bottom PCB layer, with lands for the capacitors and ferrite beads of the AVCCAUXTX and AVCCAUXRX supplies. The ferrite beads are L10, L13, L15, L16, L19, L33, and L34; the capacitors are C141, C144, C211, C221, C223, C225, C227, and C229.
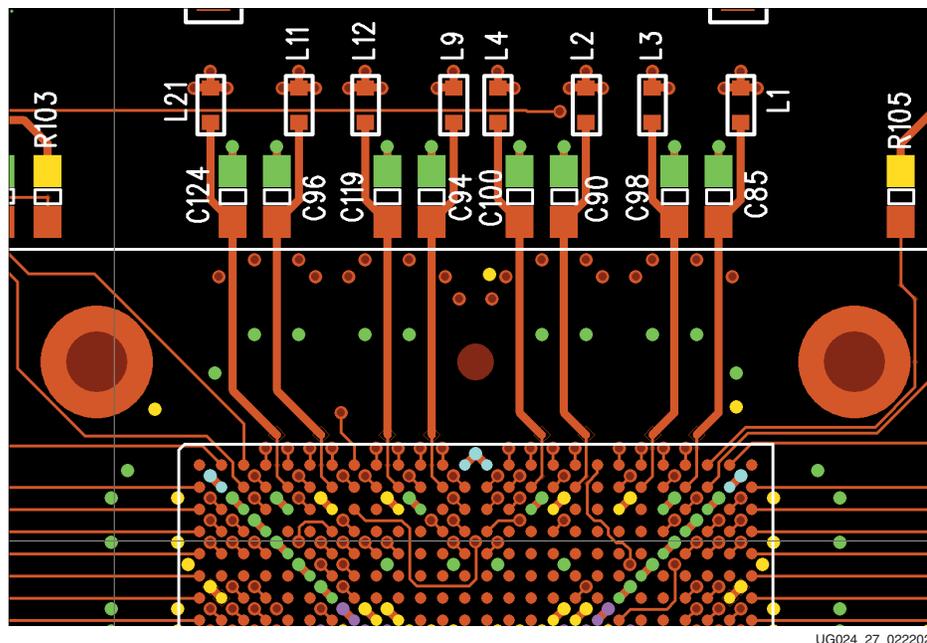
Figure 4-8: **Example Power Filtering PCB Layout for Four MGTs, Top Layer**

UG024_28_022202
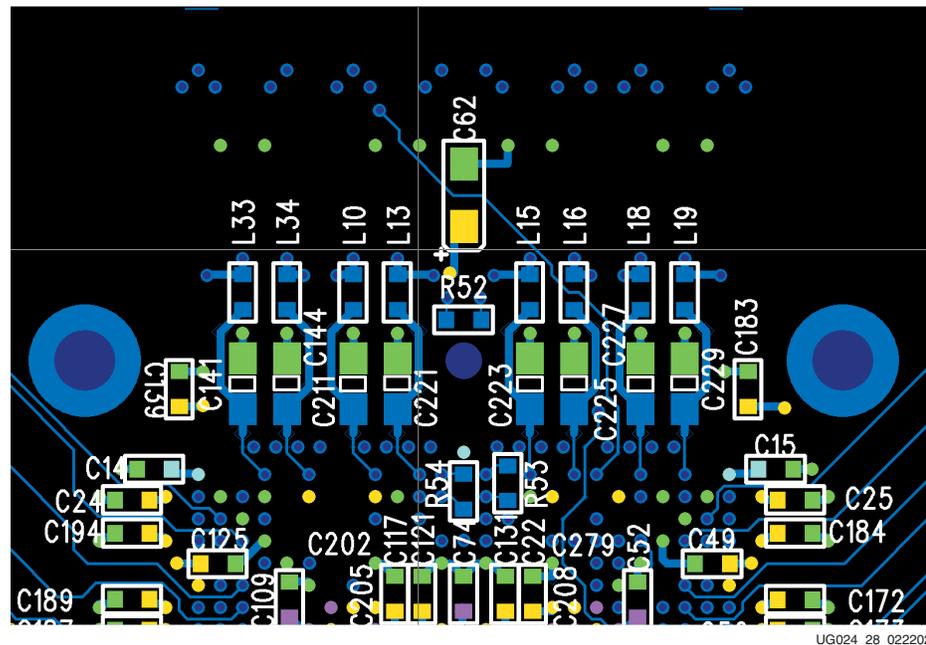
Figure 4-9: **Example Power Filtering PCB Layout for Four MGTs, Bottom Layer**

Differential impedance of traces on the finished PCB should be verified with Time Domain Reflectometry (TDR) measurements.

Power and ground paths in the PCB must be designed to have the lowest inductance possible. To this end, dedicated planes for ground must be used. When vias are a part of the power distribution path (as they are when connecting a bypass capacitor to its associated power and ground layers), multiple vias should be used to decrease the total inductance of the path. If power filtering capacitors are mounted on the same side of the board as the FPGA, designers can choose to place power layers closer to the surface of the board to shorten the path of power travel through vias.

High-Speed Serial Trace Design

Routing Serial Traces

All Rocket I/O transceiver I/Os are placed on the periphery of the BGA package to facilitate routing and inspection (since JTAG is not available on serial I/O pins). Two output/input impedance options are available in the Rocket I/O transceivers: 50Ω and 75Ω. Controlled impedance traces with a corresponding impedance should be used to connect the Rocket I/O transceiver to other compatible transceivers. In chip-to-chip PCB applications, 50Ω termination and 100Ω differential transmission lines are recommended.

When routing a differential pair, the complementary traces must be matched in length to as close a tolerance as is feasible. Length mismatches produce common mode noise and radiation. Severe length mismatches produce jitter and unpredictable timing problems at the receiver. Matching the differential traces to within 50 mils (1.27 mm) produces a robust design. Since signals propagate in FR4 PCB traces at approximately 180 ps per inch, a difference of 50 mils produces a timing skew of roughly 9 ps. Use SI CAD tools to confirm these assumptions on specific board designs.

All signal traces must have an intact reference plane beneath them. Stripline and microstrip geometries may be used. The reference plane should extend no less than five trace widths to either side of the trace to ensure predictable transmission line behavior.

Routing of a differential pair is optimally done in a point-to-point fashion, ideally remaining on the same PCB routing layer. As vias represent an impedance discontinuity, layer-to-layer changes should be avoided wherever possible. It is acceptable to traverse the

PCB stackup to reach the transmitter and receiver package pins. If serial traces must change layers, care must be taken to ensure an intact current return path. For this reason, routing of high-speed serial traces should be on signal layers that share a reference plane. If the signal layers do not share a reference plane, a capacitor of value 0.01 µF should be connected across the two reference layers close to the vias where the signals change layers. If both of the reference layers are DC coupled (if they are both ground), they can be connected with vias close to where the signals change layers.

To control crosstalk, serial differential traces should be spaced at least five trace separation widths from all other PCB routes, including other serial pairs. A larger spacing is required if the other PCB routes carry especially noisy signals, such as TTL and similar.

The Rocket I/O transceiver is designed to function at 3.125 Gb/s through 20 inches of FR4 with two high-bandwidth connectors. Longer trace lengths require either a low-loss dielectric or considerably wider serial traces.

Differential Trace Design

The characteristic impedance of a pair of differential traces depends not only on the individual trace dimensions, but also on the spacing between them. The Rocket I/O transceivers require either a 100Ω or 150Ω differential trace impedance (depending on whether the 50Ω or 75Ω termination option is selected). To achieve this differential impedance requirement, the characteristic impedance of each individual trace must be slightly higher than half of the target differential impedance. A field solver should be used to determine the exact trace geometry suited to the specific application (Figure 4-10). This task should not be left up to the PCB vendor.
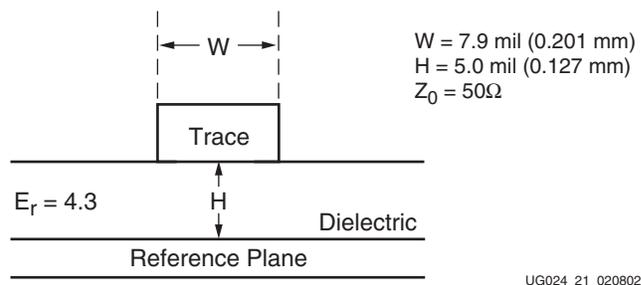
Figure 4-10: **Single-Ended Trace Geometry**

Tight coupling of differential traces is recommended. Tightly coupled traces (as opposed to loosely coupled) maintain a very close proximity to one another along their full length. Since the differential impedance of tightly coupled traces depends heavily on their proximity to each other, it is imperative that they maintain constant spacing along their full length, without deviation. If it is necessary to separate the traces in order to route through a pin field or other PCB obstacle, it can be helpful to modify the trace geometry in the vicinity of the obstacle to correct for the impedance discontinuity (increase the individual trace width where trace separation occurs).

Figure 4-11 and Figure 4-12 show examples of PCB geometries that result in 100Ω differential impedance.

W_1 = 6.29 mil (0.160 mm)
W_2 = 6.29 mil (0.160 mm)
S = 10 mil (0.254 mm)
H = 5.0 mil (0.127 mm)
Z_{01} = 55.3Ω
Z_{02} = 55.3Ω
Z_{0DIFF} = 100Ω
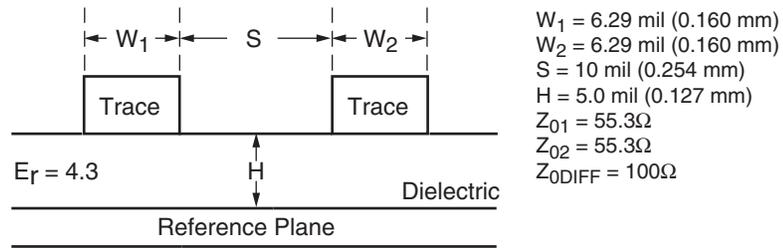
UG024_22_020802

Figure 4-11: **Microstrip Edge-Coupled Differential Pair**

W_1 = 3.0 mil (0.076 mm)
W_2 = 3.0 mil (0.076 mm)
S = 6.85 mil (0.174 mm)
H_1 = 10.0 mil (0.254 mm)
H_2 = 10.0 mil (0.254 mm)
Z_{01} = 64.8Ω
Z_{02} = 64.8Ω
Z_{0DIFF} = 100Ω
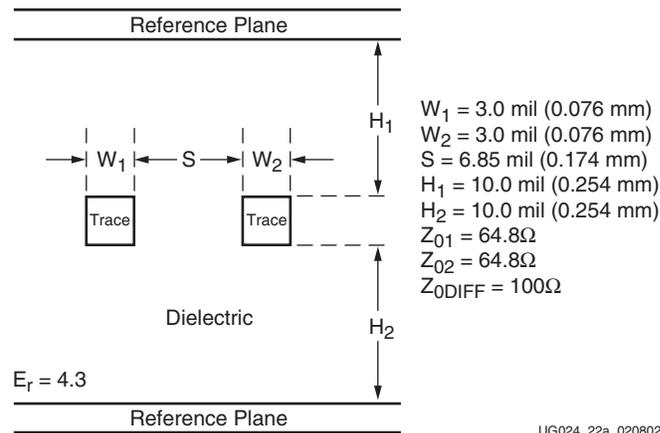
UG024_22a_020802

Figure 4-12: **Stripline Edge-Coupled Differential Pair**

AC and DC Coupling

AC coupling (use of DC blocking capacitors in the signal path) should be used in cases where transceiver differential voltages are compatible, but common mode voltages are not. Some designs require AC coupling to accommodate hot plug-in, and/or differing power supply voltages at different transceivers. This is illustrated in Figure 4-13.

DC coupling (direct connection) is preferable in cases where Rocket I/O transceivers are interfaced with other Rocket I/O transceivers or other Mindspeed transceivers that have compatible differential and common mode voltage specifications. Passive components are not required when DC coupling is used. This is illustrated in Figure 4-14.
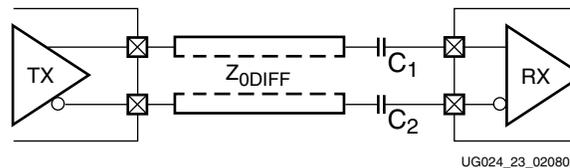
UG024_23_020802

Figure 4-13: **AC-Coupled Serial Link**
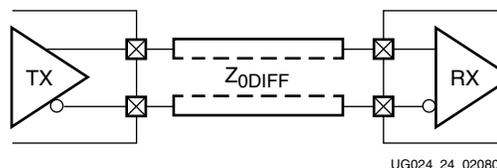
UG024_24_020802

Figure 4-14: **DC-Coupled Serial Link**

Power Consumption

The maximum power consumption per port is 350 mW at 3.125 Gb/s operation. All unused serial I/O can be turned off to consume zero power.

POWERDOWN

POWERDOWN allows "shutting off" the transceiver in case it is not needed for the design, or will not be transmitting or receiving for a long period of time. When POWERDOWN is enabled, the transceiver does not use any power. The clocks are disabled and do not propagate through the core. The 3-state TXP and TXN pins are set to high-Z, while the outputs to the fabric are frozen but *not* set to high-Z. Unused transceivers are automatically powered down to save on power consumption.

Reference Clock

A high degree of accuracy is required from the reference clock. For this reason, it is required that an EPSON EG-2121CA 2.5V oscillator be used. The power supply circuit specified by the manufacturer must be used, and the circuit in Figure 4-15 must be used to interface the LVPECL outputs of the oscillator with the LVDS inputs of the transceiver reference clock.
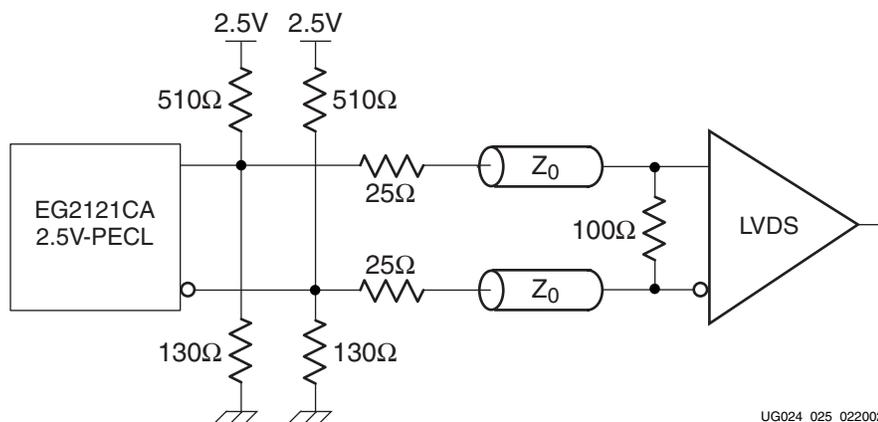
Figure 4-15: **Reference CLock Oscillator Interface**

Simulation and Implementation

Simulation Models

Smart Model

Smart models are an encrypted version to the actual HDL code. These models allow the user to simulate with the actual functionality without having access to the code itself. A simulator with smart model capability is required to use the smart models.

HSPICE

HSPICE is an analog design model that allows simulation of the RX and TX high-speed transceiver. The following HSPICE deck is an example of how to set up such a simulation.

Behavioral

Behavioral models allow for simulation without the need to upgrade the simulator to support smart models.

Implementation Tools

Synthesis

During synthesis, the transceiver is treated as a "black box." This requires that a wrapper be used that describes the modules port.

Par

For place and route, the transceiver has one restriction. This is required when channel bonding is implemented. Because of the delay limitations on the CHBONDO to CHBONDI ports, linking of the Master to a Slave_1_hop must run either in the X or Y direction, but not both.

In Figure 5-1, the two Slave_1_hops are linked to the master in only one direction. To navigate to the other slave (a Slave_2_hops), both X and Y displacement is needed. This slave needs one level of daisy-chaining, which is the basis of the Slave_2_hops setting.

Figure 5-2 shows the channel bonding mode and linking for a 2VP50, which contains more transceivers (16) per chip. To ensure the timing is met on the link between the CHBONDO

and CHBONDI ports, a constraint must be added to check the time delay. The UCF example below shows and describes this.
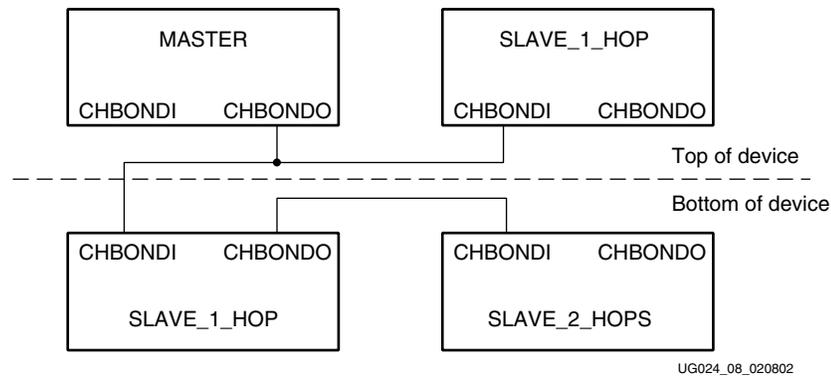
Figure 5-1: **2VP2 Implementation**
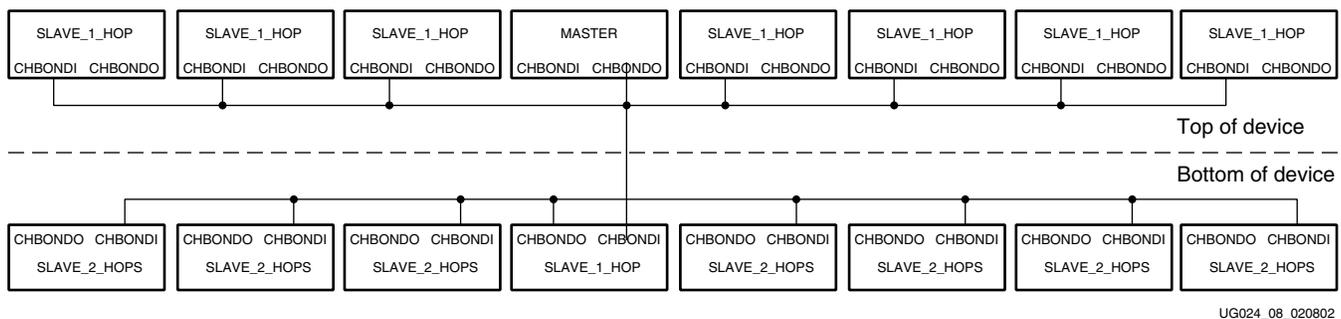
Figure 5-2: **2VP50 Implementation**

UCF Example

```
NET "chbond_*" MAXDELAY = 4.2 ns ;
```

4.2 ns is estimated as the channel bonding delay. This is based upon an RXUSRCLK of 156.25 MHz (6.4 ns period), less 0.2 ns for estimated clock skew, less 2.0 ns for estimated clock-to-out/setup-time adjustment:

6.4 ns – 0.2 ns – 2.0 ns = 4.2 ns

This design used four Rocket I/O multi-gigabit transceivers, consisting of one master, two Slave_1_hop, and one Slave_2_hops. The net `chbond_m_s01[3:0]` connects the master and two Slave_1_hop. The net `chbond_s1_s2[3:0]` connects one Slave_1_hop and one Slave_2_hops. `NET "chbond_*" MAXDELAY = 4.2 ns ;` constrains all these connections.

Implementing Clock Schemes

Sometimes certain FPGA resources are needed for specific logic. With Rocket I/O clocking schemes, the user has several resource choices. If the transceivers implemented are only at the top or bottom of the device, the REFCLK of the transceivers is not required to run through a clock tree resource. This saves this resource for other user logic. However, it does require additional I/O pins to be used (one for the DCM and one for the transceiver). Figure 3-3, page 46, shows this scenario, which is similar to Figure 3-1 minus the clock-tree resource. If transceivers from both the top and bottom of the device are used or device I/Os are at a premium, the clock tree resource is used allowing one less I/O pin used.

Diagnostic Signals

Often a diagnostic check is needed upon power-up. Rocket I/O transceivers have several inputs and outputs to run these checks.

LOOPBACK

LOOPBACK allows the user to send the data that is being transmitted directly to the receiver of the transceiver. Table 5-1 shows the three modes for loopback.

Table 5-1: **LOOPBACK Modes**

Input Value	Mode	Description
00	Normal Mode	The normal mode is selected during normal operation. The transmitted data is sent out the differential transmit ports (TXN, TXP) and are sent to another transceiver without being sent to its own receiver logic. During normal operation, the LOOPBACK should be set to "00".
01	External Serial Mode	The external serial mode is used to check that the entire transceiver is working properly. This includes testing of the 8B/10B encoding/decoding. This emulates what another transceiver would receive as data from this specific transceiver design.
10	Internal Parallel Mode	For testing of interfacing logic, the Internal Parallel Mode allows the use of linking the transmit and receive interface logic without having to go to another transceiver in the cases of 8B/10B bypassed or to reduce data latency from TXDATA to RXDATA.

Rocket I/O™ Cell Models

Summary

This appendix documents the Rocket I/O™ Multi-Gigabit Transceiver cell models. The following information lists the Verilog module declarations of the model and pins associated with each of the Rocket I/O communication standards available in the Virtex-II Pro family.

Verilog Module Declarations

GT_AURORA_1

```
module GT_AURORA_1 (
    CHBONDDONE,
    CHBONDO,
    CONFIGOUT,
    RXBUFSTATUS,
    RXCHARISCOMMA,
    RXCHARISK,
    RXCHECKINGCRC,
    RXCLKCORCNT,
    RXCOMMADET,
    RXCRCERR,
    RXDATA,
    RXDISPERR,
    RXLOSSOFSYNC,
    RXNOTINTABLE,
    RXREALIGN,
    RXRECCLK,
    RXRUNDISP,
    TXBUFERR,
    TXKERR,
    TXN,
    TXP,
    TXRUNDISP,
    CHBONDI,
    CONFIGENABLE,
    CONFIGIN,
    ENCHANSYNC,
    LOOPBACK,
    POWERDOWN,
    REFCLK,
    REFCLK2,
    REFCLKSEL,
    RXN,
```

```
            RXP,
            RXPOLARITY,
            RXRESET,
            RXUSRCLK,
            RXUSRCLK2,
            TXBYPASS8B10B,
            TXCHARDISPMODE,
            TXCHARDISPVAL,
            TXCHARISK,
            TXDATA,
            TXFORCECRCERR,
            TXINHIBIT,
            TXPOLARITY,
            TXRESET,
            TXUSRCLK,
            TXUSRCLK2
        );
```

GT_AURORA_2

```
        module GT_AURORA_2 (
            CHBONDDONE,
            CHBONDO,
            CONFIGOUT,
            RXBUFSTATUS,
            RXCHARISCOMMA,
            RXCHARISK,
            RXCHECKINGCRC,
            RXCLKCORCNT,
            RXCOMMADET,
            RXCRCERR,
            RXDATA,
            RXDISPERR,
            RXLOSSOFSYNC,
            RXNOTINTABLE,
            RXREALIGN,
            RXRECCLK,
            RXRUNDISP,
            TXBUFERR,
            TXKERR,
            TXN,
            TXP,
            TXRUNDISP,
            CHBONDI,
            CONFIGENABLE,
            CONFIGIN,
            ENCHANSYNC,
            LOOPBACK,
            POWERDOWN,
            REFCLK,
            REFCLK2,
            REFCLKSEL,
            RXN,
            RXP,
            RXPOLARITY,
            RXRESET,
            RXUSRCLK,
            RXUSRCLK2,
            TXBYPASS8B10B,
            TXCHARDISPMODE,
```

```
                              TXCHARDISPVAL,
                              TXCHARISK,
                              TXDATA,
                              TXFORCECRCERR,
                              TXINHIBIT,
                              TXPOLARITY,
                              TXRESET,
                              TXUSRCLK,
                              TXUSRCLK2
                          );
```

GT_AURORA_4

```
                  module GT_AURORA_4 (
                      CHBONDDONE,
                      CHBONDO,
                      CONFIGOUT,
                      RXBUFSTATUS,
                      RXCHARISCOMMA,
                      RXCHARISK,
                      RXCHECKINGCRC,
                      RXCLKCORCNT,
                      RXCOMMADET,
                      RXCRCERR,
                      RXDATA,
                      RXDISPERR,
                      RXLOSSOFSYNC,
                      RXNOTINTABLE,
                      RXREALIGN,
                      RXRECCLK,
                      RXRUNDISP,
                      TXBUFERR,
                      TXKERR,
                      TXN,
                      TXP,
                      TXRUNDISP,
                      CHBONDI,
                      CONFIGENABLE,
                      CONFIGIN,
                      ENCHANSYNC,
                      LOOPBACK,
                      POWERDOWN,
                      REFCLK,
                      REFCLK2,
                      REFCLKSEL,
                      RXN,
                      RXP,
                      RXPOLARITY,
                      RXRESET,
                      RXUSRCLK,
                      RXUSRCLK2,
                      TXBYPASS8B10B,
                      TXCHARDISPMODE,
                      TXCHARDISPVAL,
                      TXCHARISK,
                      TXDATA,
                      TXFORCECRCERR,
                      TXINHIBIT,
                      TXPOLARITY,
                      TXRESET,
```

```
            TXUSRCLK,
            TXUSRCLK2
        );
```

GT_CUSTOM

```
        module GT_CUSTOM (
            CHBONDDONE,
            CHBONDO,
            CONFIGOUT,
            RXBUFSTATUS,
            RXCHARISCOMMA,
            RXCHARISK,
            RXCHECKINGCRC,
            RXCLKCORCNT,
            RXCOMMADET,
            RXCRCERR,
            RXDATA,
            RXDISPERR,
            RXLOSSOFSYNC,
            RXNOTINTABLE,
            RXREALIGN,
            RXRECCLK,
            RXRUNDISP,
            TXBUFERR,
            TXKERR,
            TXN,
            TXP,
            TXRUNDISP,
            CHBONDI,
            CONFIGENABLE,
            CONFIGIN,
            ENCHANSYNC,
            LOOPBACK,
            POWERDOWN,
            REFCLK,
            REFCLK2,
            REFCLKSEL,
            RXN,
            RXP,
            RXPOLARITY,
            RXRESET,
            RXUSRCLK,
            RXUSRCLK2,
            TXBYPASS8B10B,
            TXCHARDISPMODE,
            TXCHARDISPVAL,
            TXCHARISK,
            TXDATA,
            TXFORCECRCERR,
            TXINHIBIT,
            TXPOLARITY,
            TXRESET,
            TXUSRCLK,
            TXUSRCLK2
        );
```

GT_ETHERNET_1

```verilog
module GT_ETHERNET_1 (
    CONFIGOUT,
    RXBUFSTATUS,
    RXCHARISCOMMA,
    RXCHARISK,
    RXCHECKINGCRC,
    RXCLKCORCNT,
    RXCOMMADET,
    RXCRCERR,
    RXDATA,
    RXDISPERR,
    RXLOSSOFSYNC,
    RXNOTINTABLE,
    RXREALIGN,
    RXRECCLK,
    RXRUNDISP,
    TXBUFERR,
    TXKERR,
    TXN,
    TXP,
    TXRUNDISP,
    CONFIGENABLE,
    CONFIGIN,
    LOOPBACK,
    POWERDOWN,
    REFCLK,
    REFCLK2,
    REFCLKSEL,
    RXN,
    RXP,
    RXPOLARITY,
    RXRESET,
    RXUSRCLK,
    RXUSRCLK2,
    TXBYPASS8B10B,
    TXCHARDISPMODE,
    TXCHARDISPVAL,
    TXCHARISK,
    TXDATA,
    TXFORCECRCERR,
    TXINHIBIT,
    TXPOLARITY,
    TXRESET,
    TXUSRCLK,
    TXUSRCLK2
);
```

GT_ETHERNET_2

```verilog
module GT_ETHERNET_2 (
    CONFIGOUT,
    RXBUFSTATUS,
    RXCHARISCOMMA,
    RXCHARISK,
    RXCHECKINGCRC,
    RXCLKCORCNT,
    RXCOMMADET,
```

```
        RXCRCERR,
        RXDATA,
        RXDISPERR,
        RXLOSSOFSYNC,
        RXNOTINTABLE,
        RXREALIGN,
        RXRECCLK,
        RXRUNDISP,
        TXBUFERR,
        TXKERR,
        TXN,
        TXP,
        TXRUNDISP,
        CONFIGENABLE,
        CONFIGIN,
        LOOPBACK,
        POWERDOWN,
        REFCLK,
        REFCLK2,
        REFCLKSEL,
        RXN,
        RXP,
        RXPOLARITY,
        RXRESET,
        RXUSRCLK,
        RXUSRCLK2,
        TXBYPASS8B10B,
        TXCHARDISPMODE,
        TXCHARDISPVAL,
        TXCHARISK,
        TXDATA,
        TXFORCECRCERR,
        TXINHIBIT,
        TXPOLARITY,
        TXRESET,
        TXUSRCLK,
        TXUSRCLK2
    );
```

GT_ETHERNET_4

```
    module GT_ETHERNET_4 (
        CONFIGOUT,
        RXBUFSTATUS,
        RXCHARISCOMMA,
        RXCHARISK,
        RXCHECKINGCRC,
        RXCLKCORCNT,
        RXCOMMADET,
        RXCRCERR,
        RXDATA,
        RXDISPERR,
        RXLOSSOFSYNC,
        RXNOTINTABLE,
        RXREALIGN,
        RXRECCLK,
        RXRUNDISP,
        TXBUFERR,
        TXKERR,
        TXN,
```

```
                            TXP,
                            TXRUNDISP,
                            CONFIGENABLE,
                            CONFIGIN,
                            LOOPBACK,
                            POWERDOWN,
                            REFCLK,
                            REFCLK2,
                            REFCLKSEL,
                            RXN,
                            RXP,
                            RXPOLARITY,
                            RXRESET,
                            RXUSRCLK,
                            RXUSRCLK2,
                            TXBYPASS8B10B,
                            TXCHARDISPMODE,
                            TXCHARDISPVAL,
                            TXCHARISK,
                            TXDATA,
                            TXFORCECRCERR,
                            TXINHIBIT,
                            TXPOLARITY,
                            TXRESET,
                            TXUSRCLK,
                            TXUSRCLK2
                        );
```

GT_FIBRE_CHAN_1

```
                module GT_FIBRE_CHAN_1 (
                    CONFIGOUT,
                    RXBUFSTATUS,
                    RXCHARISCOMMA,
                    RXCHARISK,
                    RXCHECKINGCRC,
                    RXCLKCORCNT,
                    RXCOMMADET,
                    RXCRCERR,
                    RXDATA,
                    RXDISPERR,
                    RXLOSSOFSYNC,
                    RXNOTINTABLE,
                    RXREALIGN,
                    RXRECCLK,
                    RXRUNDISP,
                    TXBUFERR,
                    TXKERR,
                    TXN,
                    TXP,
                    TXRUNDISP,
                    CONFIGENABLE,
                    CONFIGIN,
                    LOOPBACK,
                    POWERDOWN,
                    REFCLK,
                    REFCLK2,
                    REFCLKSEL,
                    RXN,
                    RXP,
```

```
            RXPOLARITY,
            RXRESET,
            RXUSRCLK,
            RXUSRCLK2,
            TXBYPASS8B10B,
            TXCHARDISPMODE,
            TXCHARDISPVAL,
            TXCHARISK,
            TXDATA,
            TXFORCECRCERR,
            TXINHIBIT,
            TXPOLARITY,
            TXRESET,
            TXUSRCLK,
            TXUSRCLK2
        );
```

GT_FIBRE_CHAN_2

```
        module GT_FIBRE_CHAN_2 (
            CONFIGOUT,
            RXBUFSTATUS,
            RXCHARISCOMMA,
            RXCHARISK,
            RXCHECKINGCRC,
            RXCLKCORCNT,
            RXCOMMADET,
            RXCRCERR,
            RXDATA,
            RXDISPERR,
            RXLOSSOFSYNC,
            RXNOTINTABLE,
            RXREALIGN,
            RXRECCLK,
            RXRUNDISP,
            TXBUFERR,
            TXKERR,
            TXN,
            TXP,
            TXRUNDISP,
            CONFIGENABLE,
            CONFIGIN,
            LOOPBACK,
            POWERDOWN,
            REFCLK,
            REFCLK2,
            REFCLKSEL,
            RXN,
            RXP,
            RXPOLARITY,
            RXRESET,
            RXUSRCLK,
            RXUSRCLK2,
            TXBYPASS8B10B,
            TXCHARDISPMODE,
            TXCHARDISPVAL,
            TXCHARISK,
            TXDATA,
            TXFORCECRCERR,
            TXINHIBIT,
```

```
            TXPOLARITY,
            TXRESET,
            TXUSRCLK,
            TXUSRCLK2
        );
```

GT_FIBRE_CHAN_4

```
    module GT_FIBRE_CHAN_4 (
        CONFIGOUT,
        RXBUFSTATUS,
        RXCHARISCOMMA,
        RXCHARISK,
        RXCHECKINGCRC,
        RXCLKCORCNT,
        RXCOMMADET,
        RXCRCERR,
        RXDATA,
        RXDISPERR,
        RXLOSSOFSYNC,
        RXNOTINTABLE,
        RXREALIGN,
        RXRECCLK,
        RXRUNDISP,
        TXBUFERR,
        TXKERR,
        TXN,
        TXP,
        TXRUNDISP,
        CONFIGENABLE,
        CONFIGIN,
        LOOPBACK,
        POWERDOWN,
        REFCLK,
        REFCLK2,
        REFCLKSEL,
        RXN,
        RXP,
        RXPOLARITY,
        RXRESET,
        RXUSRCLK,
        RXUSRCLK2,
        TXBYPASS8B10B,
        TXCHARDISPMODE,
        TXCHARDISPVAL,
        TXCHARISK,
        TXDATA,
        TXFORCECRCERR,
        TXINHIBIT,
        TXPOLARITY,
        TXRESET,
        TXUSRCLK,
        TXUSRCLK2
    );
```

GT_INFINIBAND_1

```
module GT_INFINIBAND_1 (
    CHBONDDONE,
    CHBONDO,
    CONFIGOUT,
    RXBUFSTATUS,
    RXCHARISCOMMA,
    RXCHARISK,
    RXCHECKINGCRC,
    RXCLKCORCNT,
    RXCOMMADET,
    RXCRCERR,
    RXDATA,
    RXDISPERR,
    RXLOSSOFSYNC,
    RXNOTINTABLE,
    RXREALIGN,
    RXRECCLK,
    RXRUNDISP,
    TXBUFERR,
    TXKERR,
    TXN,
    TXP,
    TXRUNDISP,
    CHBONDI,
    CONFIGENABLE,
    CONFIGIN,
    ENCHANSYNC,
    LOOPBACK,
    POWERDOWN,
    REFCLK,
    REFCLK2,
    REFCLKSEL,
    RXN,
    RXP,
    RXPOLARITY,
    RXRESET,
    RXUSRCLK,
    RXUSRCLK2,
    TXBYPASS8B10B,
    TXCHARDISPMODE,
    TXCHARDISPVAL,
    TXCHARISK,
    TXDATA,
    TXFORCECRCERR,
    TXINHIBIT,
    TXPOLARITY,
    TXRESET,
    TXUSRCLK,
    TXUSRCLK2
);
```

GT_INFINIBAND_2

```
module GT_INFINIBAND_2 (
    CHBONDDONE,
    CHBONDO,
    CONFIGOUT,
```

```
                RXBUFSTATUS,
                RXCHARISCOMMA,
                RXCHARISK,
                RXCHECKINGCRC,
                RXCLKCORCNT,
                RXCOMMADET,
                RXCRCERR,
                RXDATA,
                RXDISPERR,
                RXLOSSOFSYNC,
                RXNOTINTABLE,
                RXREALIGN,
                RXRECCLK,
                RXRUNDISP,
                TXBUFERR,
                TXKERR,
                TXN,
                TXP,
                TXRUNDISP,
                CHBONDI,
                CONFIGENABLE,
                CONFIGIN,
                ENCHANSYNC,
                LOOPBACK,
                POWERDOWN,
                REFCLK,
                REFCLK2,
                REFCLKSEL,
                RXN,
                RXP,
                RXPOLARITY,
                RXRESET,
                RXUSRCLK,
                RXUSRCLK2,
                TXBYPASS8B10B,
                TXCHARDISPMODE,
                TXCHARDISPVAL,
                TXCHARISK,
                TXDATA,
                TXFORCECRCERR,
                TXINHIBIT,
                TXPOLARITY,
                TXRESET,
                TXUSRCLK,
                TXUSRCLK2
            );
```

GT_INFINIBAND_4

```
            module GT_INFINIBAND_4 (
                CHBONDDONE,
                CHBONDO,
                CONFIGOUT,
                RXBUFSTATUS,
                RXCHARISCOMMA,
                RXCHARISK,
                RXCHECKINGCRC,
                RXCLKCORCNT,
                RXCOMMADET,
                RXCRCERR,
```

```
                RXDATA,
                RXDISPERR,
                RXLOSSOFSYNC,
                RXNOTINTABLE,
                RXREALIGN,
                RXRECCLK,
                RXRUNDISP,
                TXBUFERR,
                TXKERR,
                TXN,
                TXP,
                TXRUNDISP,
                CHBONDI,
                CONFIGENABLE,
                CONFIGIN,
                ENCHANSYNC,
                LOOPBACK,
                POWERDOWN,
                REFCLK,
                REFCLK2,
                REFCLKSEL,
                RXN,
                RXP,
                RXPOLARITY,
                RXRESET,
                RXUSRCLK,
                RXUSRCLK2,
                TXBYPASS8B10B,
                TXCHARDISPMODE,
                TXCHARDISPVAL,
                TXCHARISK,
                TXDATA,
                TXFORCECRCERR,
                TXINHIBIT,
                TXPOLARITY,
                TXRESET,
                TXUSRCLK,
                TXUSRCLK2
            );
```

GT_XAUI_1

```
            module GT_XAUI_1 (
                CHBONDDONE,
                CHBONDO,
                CONFIGOUT,
                RXBUFSTATUS,
                RXCHARISCOMMA,
                RXCHARISK,
                RXCHECKINGCRC,
                RXCLKCORCNT,
                RXCOMMADET,
                RXCRCERR,
                RXDATA,
                RXDISPERR,
                RXLOSSOFSYNC,
                RXNOTINTABLE,
                RXREALIGN,
                RXRECCLK,
                RXRUNDISP,
```

```
            TXBUFERR,
            TXKERR,
            TXN,
            TXP,
            TXRUNDISP,
            CHBONDI,
            CONFIGENABLE,
            CONFIGIN,
            ENCHANSYNC,
            LOOPBACK,
            POWERDOWN,
            REFCLK,
            REFCLK2,
            REFCLKSEL,
            RXN,
            RXP,
            RXPOLARITY,
            RXRESET,
            RXUSRCLK,
            RXUSRCLK2,
            TXBYPASS8B10B,
            TXCHARDISPMODE,
            TXCHARDISPVAL,
            TXCHARISK,
            TXDATA,
            TXFORCECRCERR,
            TXINHIBIT,
            TXPOLARITY,
            TXRESET,
            TXUSRCLK,
            TXUSRCLK2
        );
```

GT_XAUI_2

```
        module GT_XAUI_2 (
            CHBONDDONE,
            CHBONDO,
            CONFIGOUT,
            RXBUFSTATUS,
            RXCHARISCOMMA,
            RXCHARISK,
            RXCHECKINGCRC,
            RXCLKCORCNT,
            RXCOMMADET,
            RXCRCERR,
            RXDATA,
            RXDISPERR,
            RXLOSSOFSYNC,
            RXNOTINTABLE,
            RXREALIGN,
            RXRECCLK,
            RXRUNDISP,
            TXBUFERR,
            TXKERR,
            TXN,
            TXP,
            TXRUNDISP,
            CHBONDI,
            CONFIGENABLE,
```

```
        CONFIGIN,
        ENCHANSYNC,
        LOOPBACK,
        POWERDOWN,
        REFCLK,
        REFCLK2,
        REFCLKSEL,
        RXN,
        RXP,
        RXPOLARITY,
        RXRESET,
        RXUSRCLK,
        RXUSRCLK2,
        TXBYPASS8B10B,
        TXCHARDISPMODE,
        TXCHARDISPVAL,
        TXCHARISK,
        TXDATA,
        TXFORCECRCERR,
        TXINHIBIT,
        TXPOLARITY,
        TXRESET,
        TXUSRCLK,
        TXUSRCLK2
    );
```

GT_XAUI_4

```
    module GT_XAUI_4 (
        CHBONDDONE,
        CHBONDO,
        CONFIGOUT,
        RXBUFSTATUS,
        RXCHARISCOMMA,
        RXCHARISK,
        RXCHECKINGCRC,
        RXCLKCORCNT,
        RXCOMMADET,
        RXCRCERR,
        RXDATA,
        RXDISPERR,
        RXLOSSOFSYNC,
        RXNOTINTABLE,
        RXREALIGN,
        RXRECCLK,
        RXRUNDISP,
        TXBUFERR,
        TXKERR,
        TXN,
        TXP,
        TXRUNDISP,
        CHBONDI,
        CONFIGENABLE,
        CONFIGIN,
        ENCHANSYNC,
        LOOPBACK,
        POWERDOWN,
        REFCLK,
        REFCLK2,
        REFCLKSEL,
```

```
                        RXN,
                        RXP,
                        RXPOLARITY,
                        RXRESET,
                        RXUSRCLK,
                        RXUSRCLK2,
                        TXBYPASS8B10B,
                        TXCHARDISPMODE,
                        TXCHARDISPVAL,
                        TXCHARISK,
                        TXDATA,
                        TXFORCECRCERR,
                        TXINHIBIT,
                        TXPOLARITY,
                        TXRESET,
                        TXUSRCLK,
                        TXUSRCLK2
                    );
```