

# RocketIO™ Transceiver User Guide

UG024 (v1.5) October 16, 2002





The Xilinx logo shown above is a registered trademark of Xilinx, Inc.



The shadow X shown above is a trademark of Xilinx, Inc.

"Xilinx" and the Xilinx logo are registered trademarks of Xilinx, Inc. Any rights not expressly granted herein are reserved.

CoolRunner, RocketChips, Rocket IP, Spartan, StateBENCH, StateCAD, Virtex, XACT, XC2064, XC3090, XC4005, XC5210 are registered Trademarks of Xilinx, Inc.

ACE Controller, ACE Flash, A.K.A. Speed, Alliance Series, AllianceCORE, Bencher, ChipScope, Configurable Logic Cell, CORE Generator, CoreLINX, Dual Block, EZTag, Fast CLK, Fast CONNECT, Fast FLASH, FastMap, Fast Zero Power, Foundation, Gigabit Speeds...and Beyond!, HardWire, HDL Bencher, IRL, J Drive, JBits, LCA, LogiBLOX, Logic Cell, LogiCORE, LogicProfessor, MicroBlaze, MicroVia, MultiLINX, Nano-Blaze, PicoBlaze, PLUSASM, PowerGuide, PowerMaze, QPro, Real-PCI, RocketIO, SelectIO, SelectRAM, SelectRAM+, Silicon Xpresso, Smartguide, Smart-IP, SmartSearch, SMARTswitch, System ACE, Testbench In A Minute, TrueMap, UIM, VectorMaze, VersaBlock, VersaRing, Virtex-II Pro, Wave Table, WebFITTER, WebPACK, WebPOWERED, XABEL, XACT-Floorplanner, XACT-Performance, XACTstep Advanced, XACTstep Foundry, XAM, XAPP, X-BLOX +, XC designated products, XChecker, XDM, XEPLD, Xilinx Foundation Series, Xilinx XDTV, Xinfo, XSI, Xtrem-eDSP and ZERO+ are trademarks of Xilinx, Inc.

The Programmable Logic Company is a service mark of Xilinx, Inc.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

IBM IBM Logo PowerPC PowerPC Logo Blue Logic CoreConnect CodePack

All other trademarks are the property of their respective owners.

Xilinx does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx provides any design, code, or information shown or described herein "as is." By providing the design, code, or information as one possible implementation of a feature, application, or standard, Xilinx makes no representation that such implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of any such implementation, including but not limited to any warranties or representations that the implementation is free from claims of infringement, as well as any implied warranties of merchantability or fitness for a particular purpose. Xilinx assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

Copyright 2002 Xilinx, Inc. All Rights Reserved.

---

**RocketIO™ Transceiver User Guide**  
**UG024 (v1.5) October 16, 2002**

The following table shows the revision history for this document.

| <b>Date</b> | <b>Version</b> | <b>Revision</b>   |
|-------------|----------------|---|
| 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)   |
| 07/11/02    | 1.3            | Updated <b>PCB Design Requirements, Chapter 4</b> . Added Timing Model as Appendix A, changed Cell Models to Appendix B.  |
| 09/27/02    | 1.4            | <ul style="list-style-type: none"><li>• Added additional <b>IMPORTANT NOTES</b> regarding ISE revisions at the beginning of <b>Chapter 1</b></li><li>• Added material in section <b>CRC Operation, Chapter 3</b></li><li>• Added section <b>Other Important Design Notes, Chapter 3</b></li><li>• New pre-emphasis eye diagrams in section <b>Pre-emphasis Techniques, Chapter 4</b></li><li>• Numerous parameter additions previously shown as "TBD" in <b>MGT Package Pins, Chapter 5</b></li></ul> |
| 10/16/02    | 1.5            | <ul style="list-style-type: none"><li>• Corrected pinouts in <b>Table 5-2</b>, FF1152 package, device column 2VP20/30, LOC Constraints rows GT_X0_Y0 and GT_X0_Y1.</li><li>• Corrected section <b>CRC Latency</b> and <b>Table 3-15</b> to express latency in terms of TXUSRCLK and RXUSRCLK cycles.</li><li>• Corrected sequence of packet elements in <b>Figure 3-16</b>.</li></ul>   |



# Contents

---

|   |    |
|---|----|
| <b>Schedule of Figures</b> .....                | 9  |
| <b>Schedule of Tables</b> .....                 | 11 |
| <b>Chapter 1: Introduction</b>                  |    |
| RocketIO Features .....                         | 13 |
| In This User Guide .....                        | 14 |
| Naming Conventions .....                        | 14 |
| For More Information .....                      | 14 |
| <b>Chapter 2: RocketIO Transceiver Overview</b> |    |
| Basic Architecture and Capabilities .....       | 15 |
| Clock Synthesizer .....                         | 17 |
| Clock and Data Recovery .....                   | 18 |
| <b>Transmitter</b> .....                        | 18 |
| FPGA Transmit Interface .....                   | 18 |
| 8B/10B Encoder .....                            | 18 |
| Disparity Control .....                         | 18 |
| Transmit FIFO .....                             | 19 |
| Serializer .....                                | 19 |
| Transmit Termination .....                      | 19 |
| Pre-emphasis Circuit and Swing Control .....    | 19 |
| <b>Receiver</b> .....                           | 19 |
| Deserializer .....                              | 19 |
| Receiver Termination .....                      | 20 |
| 8B/10B Decoder .....                            | 20 |
| Loopback .....                                  | 20 |
| <b>Elastic and Transmitter Buffers</b> .....    | 21 |
| Receiver Buffer .....                           | 21 |
| Clock Correction .....                          | 21 |
| Channel Bonding .....                           | 22 |
| Transmitter Buffer .....                        | 23 |
| CRC .....                                       | 23 |
| Reset/Power Down .....                          | 23 |
| <b>Chapter 3: Digital Design Considerations</b> |    |
| List of Available Ports .....                   | 25 |
| Primitive Attributes .....                      | 29 |
| Modifiable Primitives .....                     | 33 |
| Byte Mapping .....                              | 38 |
| <b>Clocking</b> .....                           | 38 |
| Clock Signals .....                             | 38 |
| Clock Ratio .....                               | 39 |
| Digital Clock Manager (DCM) Examples .....      | 39 |
| Example 1: Two-Byte Clock .....                 | 40 |
| Example 2: Four-Byte Clock .....                | 43 |
| Example 3: One-Byte Clock .....                 | 46 |

|  |    |
|--|----|
| BREFCLK.....   | 50 |
| Half-Rate Clocking Scheme .....  | 52 |
| Multiplexed Clocking Scheme.....   | 53 |
| Data Path Latency .....  | 53 |
| <i>Transmitter Latency</i> .....   | 53 |
| <i>Receiver Latency</i> .....  | 54 |
| Clock Dependency .....   | 54 |
| <b>Resets</b> .....  | 54 |
| <b>RocketIO Transceiver Instantiations</b> .....                         | 56 |
| <i>HDL Code Examples</i> .....   | 56 |
| <b>PLL Operation and Clock Recovery</b> .....                            | 56 |
| Clock Correction Count.....  | 56 |
| RX_LOSS_OF_SYNC_FSM .....  | 57 |
| <i>SYNC_ACQUIRED (RXLOSSOFSYNC = 00)</i> .....                           | 57 |
| <i>RESYNC (RXLOSSOFSYNC = 01)</i> .....                                  | 57 |
| <i>LOSS_OF_SYNC (RXLOSSOFSYNC = 10)</i> .....                            | 57 |
| 8B/10B Operation.....  | 57 |
| Vitesse Disparity Example .....  | 59 |
| <i>Transmitting Vitesse Channel Bonding Sequence</i> .....               | 60 |
| <i>Receiving Vitesse Channel Bonding Sequence</i> .....                  | 60 |
| <b>Status Signals</b> .....  | 61 |
| <b>8B/10B Encoding</b> .....   | 61 |
| 8B/10B Serial Output Format.....   | 69 |
| <i>HDL Code Examples: Transceiver Bypassing of 8B/10B Encoding</i> ..... | 70 |
| <b>CRC Operation</b> .....   | 70 |
| CRC Generation.....  | 70 |
| CRC Latency.....   | 71 |
| CRC Limitations .....  | 71 |
| CRC Modes.....   | 71 |
| <i>USER_MODE</i> .....   | 71 |
| <i>FIBRECHANNEL</i> .....  | 72 |
| <i>ETHERNET</i> .....  | 72 |
| <i>INFINIBAND</i> .....  | 72 |
| <b>Channel Bonding (Channel-to-Channel Alignment)</b> .....              | 73 |
| <i>HDL Code Examples: Channel Bonding</i> .....                          | 74 |
| <b>Other Important Design Notes</b> .....                                | 75 |
| Receive Data Path 32-bit Alignment.....                                  | 75 |
| 32-bit Alignment Design .....  | 76 |
| <i>Verilog</i> .....   | 76 |
| <i>VHDL</i> .....  | 79 |

## Chapter 4: Analog Design Considerations

|  |    |
|--|----|
| Serial I/O Description .....           | 83 |
| Pre-emphasis Techniques.....           | 84 |
| Differential Receiver .....            | 87 |
| Jitter .....                           | 87 |
| Total Jitter (DJ + RJ).....            | 87 |
| <i>Deterministic Jitter (DJ)</i> ..... | 87 |
| <i>Random Jitter (RJ)</i> .....        | 87 |
| Clock and Data Recovery.....           | 87 |
| PCB Design Requirements .....          | 89 |
| Power Conditioning.....                | 89 |
| <i>Voltage Regulation</i> .....        | 89 |
| <i>Passive Filtering</i> .....         | 90 |

|  |    |
|--|----|
| High-Speed Serial Trace Design.....                | 91 |
| <i>Routing Serial Traces</i> .....                 | 91 |
| <i>Differential Trace Design</i> .....             | 92 |
| <i>AC and DC Coupling</i> .....                    | 93 |
| Reference Clock .....                              | 94 |
| <i>Epson EG-2121CA 2.5V (LVPECL outputs)</i> ..... | 94 |
| <i>Pletronics LV1145B (LVDS outputs)</i> .....     | 94 |
| <b>Other Important Design Notes</b> .....          | 95 |
| Powering the RocketIO Transceivers .....           | 95 |
| The POWERDOWN Port .....                           | 95 |

## Chapter 5: Simulation and Implementation

|                                   |     |
|-----------------------------------|-----|
| <b>Simulation Models</b> .....    | 97  |
| SmartModels .....                 | 97  |
| HSPICE .....                      | 97  |
| <b>Implementation Tools</b> ..... | 97  |
| Synthesis .....                   | 97  |
| Par .....                         | 97  |
| UCF Example .....                 | 98  |
| Implementing Clock Schemes .....  | 98  |
| <b>MGT Package Pins</b> .....     | 99  |
| <b>Diagnostic Signals</b> .....   | 102 |
| LOOPBACK .....                    | 102 |

## Appendix A: RocketIO Transceiver Timing Model

|  |     |
|--|-----|
| <b>Timing Parameters</b> .....                     | 104 |
| Setup/Hold Times of Inputs Relative to Clock ..... | 105 |
| Clock to Output Delays .....                       | 105 |
| Clock Pulse Width.....                             | 105 |
| <b>Timing Parameter Tables and Diagram</b> .....   | 106 |

## Appendix B: RocketIO Transceiver Cell Models

|  |     |
|--|-----|
| <b>Summary</b> .....                     | 109 |
| <b>Verilog Module Declarations</b> ..... | 109 |
| GT_AURORA_1.....                         | 109 |
| GT_AURORA_2.....                         | 110 |
| GT_AURORA_4.....                         | 111 |
| GT_CUSTOM.....                           | 112 |
| GT_ETHERNET_1.....                       | 113 |
| GT_ETHERNET_2.....                       | 114 |
| GT_ETHERNET_4.....                       | 114 |
| GT_FIBRE_CHAN_1.....                     | 115 |
| GT_FIBRE_CHAN_2.....                     | 116 |
| GT_FIBRE_CHAN_4.....                     | 117 |
| GT_INFINIBAND_1 .....                    | 118 |
| GT_INFINIBAND_2 .....                    | 119 |
| GT_INFINIBAND_4 .....                    | 120 |
| GT_XAUI_1 .....                          | 121 |
| GT_XAUI_2 .....                          | 122 |
| GT_XAUI_4 .....                          | 122 |





# Figures

---

## Chapter 1: Introduction

## Chapter 2: RocketIO Transceiver Overview

|   |    |
|---|----|
| <i>Figure 2-1: RocketIO Transceiver Block Diagram</i> ..... | 16 |
| <i>Figure 2-2: Clock Correction in Receiver</i> .....       | 21 |
| <i>Figure 2-3: Channel Bonding (Alignment)</i> .....        | 22 |

## Chapter 3: Digital Design Considerations

|  |    |
|--|----|
| <i>Figure 3-1: Two-Byte Clock</i> .....                                | 40 |
| <i>Figure 3-2: Four-Byte Clock</i> .....                               | 43 |
| <i>Figure 3-3: One-Byte Clock</i> .....                                | 46 |
| <i>Figure 3-4: REFCLK/BREFCLK Selection Logic</i> .....                | 50 |
| <i>Figure 3-5: One-Byte Data Path Clocks, SERDES_10B = TRUE</i> .....  | 52 |
| <i>Figure 3-6: Two-Byte Data Path Clocks, SERDES_10B = TRUE</i> .....  | 52 |
| <i>Figure 3-7: Four-Byte Data Path Clocks, SERDES_10B = TRUE</i> ..... | 52 |
| <i>Figure 3-8: Multiplexed REFCLK</i> .....                            | 53 |
| <i>Figure 3-9: 8B/10B Data Flow</i> .....                              | 58 |
| <i>Figure 3-10: 10-Bit TX Data Map with 8B/10B Bypassed</i> .....      | 59 |
| <i>Figure 3-11: 10-Bit RX Data Map with 8B/10B Bypassed</i> .....      | 59 |
| <i>Figure 3-12: 8B/10B Parallel to Serial Conversion</i> .....         | 69 |
| <i>Figure 3-13: 4-Byte Serial Structure</i> .....                      | 70 |
| <i>Figure 3-14: CRC Packet Format</i> .....                            | 71 |
| <i>Figure 3-15: USER_MODE / FIBRE_CHANNEL Mode</i> .....               | 72 |
| <i>Figure 3-16: Ethernet Mode</i> .....                                | 72 |
| <i>Figure 3-17: Infiniband Mode</i> .....                              | 73 |
| <i>Figure 3-18: Local Route Header</i> .....                           | 73 |
| <i>Figure 3-19: RXDATA Aligned Correctly</i> .....                     | 75 |
| <i>Figure 3-20: Realignment of RXDATA</i> .....                        | 76 |

## Chapter 4: Analog Design Considerations

|  |    |
|--|----|
| <i>Figure 4-1: Differential Amplifier</i> .....                                      | 83 |
| <i>Figure 4-2: Alternating K28.5+ with No Pre-Emphasis</i> .....                     | 85 |
| <i>Figure 4-3: K28.5+ with Pre-Emphasis</i> .....                                    | 85 |
| <i>Figure 4-5: Eye Diagram, 33% Pre-Emphasis</i> .....                               | 86 |
| <i>Figure 4-4: Eye Diagram, 10% Pre-Emphasis</i> .....                               | 86 |
| <i>Figure 4-6: Power Supply Circuit Using LT1963 Regulator</i> .....                 | 89 |
| <i>Figure 4-7: Power Filtering Network for One Transceiver</i> .....                 | 90 |
| <i>Figure 4-8: Example Power Filtering PCB Layout for Four MGTs, Top Layer</i> ..... | 91 |

---

|   |    |
|---|----|
| <i>Figure 4-9: Example Power Filtering PCB Layout for Four MGTs, Bottom Layer</i> ..... | 91 |
| <i>Figure 4-10: Single-Ended Trace Geometry</i> .....                                   | 92 |
| <i>Figure 4-11: Microstrip Edge-Coupled Differential Pair</i> .....                     | 93 |
| <i>Figure 4-12: Stripline Edge-Coupled Differential Pair</i> .....                      | 93 |
| <i>Figure 4-13: AC-Coupled Serial Link</i> .....  | 93 |
| <i>Figure 4-14: DC-Coupled Serial Link</i> .....  | 94 |
| <i>Figure 4-15: LVPECL Reference Clock Oscillator Interface</i> .....                   | 94 |
| <i>Figure 4-16: LVDS Reference Clock Oscillator Interface</i> .....                     | 94 |

## **Chapter 5: Simulation and Implementation**

|   |    |
|---|----|
| <i>Figure 5-1: 2VP2 Implementation</i> .....  | 98 |
| <i>Figure 5-2: 2VP50 Implementation</i> ..... | 98 |

## **Appendix A: RocketIO Transceiver Timing Model**

|   |     |
|---|-----|
| <i>Figure A-1: RocketIO Transceiver Block Diagram</i> .....                 | 104 |
| <i>Figure A-2: RocketIO Transceiver Timing Relative to Clock Edge</i> ..... | 108 |

## **Appendix B: RocketIO Transceiver Cell Models**

# Tables

---

## Chapter 1: Introduction

## Chapter 2: RocketIO Transceiver Overview

|  |    |
|--|----|
| <i>Table 2-1: RocketIO Cores</i> .....   | 15 |
| <i>Table 2-2: Communications Standards Supported by RocketIO Transceiver</i> ..... | 15 |
| <i>Table 2-3: Serial Baud Rates and the SERDES_10B Attribute</i> .....             | 16 |
| <i>Table 2-4: Supported RocketIO Transceiver Primitives</i> .....                  | 17 |
| <i>Table 2-5: Running Disparity Control</i> .....                                  | 18 |
| <i>Table 2-6: Loopback Options</i> .....   | 20 |
| <i>Table 2-7: Reset and Power Control Descriptions</i> .....                       | 24 |
| <i>Table 2-8: Power Control Descriptions</i> .....                                 | 24 |

## Chapter 3: Digital Design Considerations

|  |    |
|--|----|
| <i>Table 3-1: GT_CUSTOM<sup>(1)</sup>, GT_AURORA, GT_FIBRE_CHAN<sup>(2)</sup>, GT_ETHERNET<sup>(2)</sup>, GT_INFINIBAND, and GT_XAUI Primitive Ports</i> ..... | 25 |
| <i>Table 3-2: RocketIO Transceiver Attributes</i> .....  | 29 |
| <i>Table 3-3: Default Attribute Values: GT_AURORA, GT_CUSTOM, GT_ETHERNET</i> .....  | 33 |
| <i>Table 3-4: Default Attribute Values: GT_FIBRE_CHAN, GT_INFINIBAND, and GT_XAUI</i> .....  | 35 |
| <i>Table 3-5: Control/Status Bus Association to Data Bus Byte Paths</i> .....  | 38 |
| <i>Table 3-6: Clock Ports</i> .....  | 38 |
| <i>Table 3-7: Data Width Clock Ratios</i> .....  | 39 |
| <i>Table 3-8: DCM Outputs for Different DATA_WIDTHs</i> .....  | 39 |
| <i>Table 3-9: BREFCLK Pin Numbers</i> .....  | 51 |
| <i>Table 3-10: Clock Correction Sequence / Data Correlation for 16-Bit Data Port</i> .....   | 56 |
| <i>Table 3-11: RXCLKCORCNT Definition</i> .....  | 56 |
| <i>Table 3-12: 8B/10B Bypassed Signal Significance</i> .....   | 58 |
| <i>Table 3-13: Valid Data Characters</i> .....   | 61 |
| <i>Table 3-14: Valid Control "K" Characters</i> .....  | 69 |
| <i>Table 3-15: Effects of CRC on Transceiver Latency</i> .....   | 71 |
| <i>Table 3-16: Global and Local Headers</i> .....  | 73 |
| <i>Table 3-17: Bonded Channel Connections</i> .....  | 74 |
| <i>Table 3-18: Master/Slave Channel Bonding Attribute Settings</i> .....   | 74 |
| <i>Table 3-19: 32-bit RXDATA, Aligned versus Misaligned</i> .....  | 75 |

## Chapter 4: Analog Design Considerations

|   |    |
|---|----|
| <i>Table 4-1: Differential Transmitter Parameters</i> ..... | 84 |
| <i>Table 4-2: Pre-emphasis Values</i> .....                 | 84 |
| <i>Table 4-3: Differential Receiver Parameters</i> .....    | 87 |

---

|  |    |
|--|----|
| <i>Table 4-4: CDR Parameters</i> .....             | 88 |
| <i>Table 4-5: Transceiver Power Supplies</i> ..... | 89 |

## **Chapter 5: Simulation and Implementation**

|   |     |
|---|-----|
| <i>Table 5-1: LOC Grid &amp; Package Pins Correlation for FG256, FG456, and FF672</i> ..... | 99  |
| <i>Table 5-2: LOC Grid &amp; Package Pins Correlation for FF896 and FF1152</i> .....        | 99  |
| <i>Table 5-3: LOC Grid &amp; Package Pins Correlation for FF1517 and FF1704</i> .....       | 100 |
| <i>Table 5-4: LOOPBACK Modes</i> .....  | 102 |

## **Appendix A: RocketIO Transceiver Timing Model**

|   |     |
|---|-----|
| <i>Table A-1: RocketIO Clock Descriptions</i> .....                           | 103 |
| <i>Table A-2: Parameters Relative to the RX User Clock (RXUSRCLK)</i> .....   | 106 |
| <i>Table A-3: Parameters Relative to the RX User Clock2 (RXUSRCLK2)</i> ..... | 106 |
| <i>Table A-4: Parameters Relative to the TX User Clock2 (TXUSRCLK2)</i> ..... | 107 |
| <i>Table A-5: Miscellaneous Clock Parameters</i> .....                        | 107 |

## **Appendix B: RocketIO Transceiver Cell Models**

## Introduction

---

### **IMPORTANT NOTES**

This document assumes use of ISE v5.1.x.

- If running ISE v4.x.x, the following modifications must be made:
  1. Remove the ports BREFCLK and BREFCLK2.
  2. Remove the REF\_CLK\_V\_SEL attribute.
- If running ISE v4.1.x, the following *additional* 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.
  4. Change the attribute name TERMINATION\_IMP to RX\_TERM\_IMP.

## RocketIO Features

The RocketIO™ 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 622 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 RocketIO *Transceiver User Guide* contains these sections:

- **Chapter 1, Introduction** — This chapter.
- **Chapter 2, RocketIO 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** — RocketIO serial overview; pre-emphasis; jitter; clock/data recovery; PCB design requirements.
- **Chapter 5, Simulation and Implementation** — Simulation models; implementation tools; debugging and diagnostics.
- **Appendix B, RocketIO Transceiver Cell Models** — Verilog module declarations associated with each of the sixteen RocketIO communication standard implementations.

## Naming Conventions

Input and output ports of the RocketIO transceiver primitives are denoted in upper-case letters. Attributes of the RocketIO 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/>.

## RocketIO Transceiver Overview

### Basic Architecture and Capabilities

The RocketIO transceiver is based on Mindspeed's SkyRail™ technology. [Figure 2-1, page 16](#), depicts an overall block diagram of the transceiver. Up to 24 transceiver modules are available on a single Virtex-II Pro FPGA, depending on the part being used. [Table 2-1](#) shows the RocketIO cores available by device.

*Table 2-1: RocketIO Cores*

| Device  | RocketIO Cores | Device   | RocketIO Cores |
|---------|----------------|----------|----------------|
| XC2VP2  | 4              | XC2VP40  | 0 or 12        |
| XC2VP4  | 4              | XC2VP50  | 0 or 16        |
| XC2VP7  | 8              | XC2VP70  | 20             |
| XC2VP20 | 8              | XC2VP100 | 0 or 20        |
| XC2VP30 | 8              | XC2VP125 | 0, 20, or 24   |

The transceiver module is designed to operate at any serial bit rate in the range of 622 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 RocketIO Transceiver*

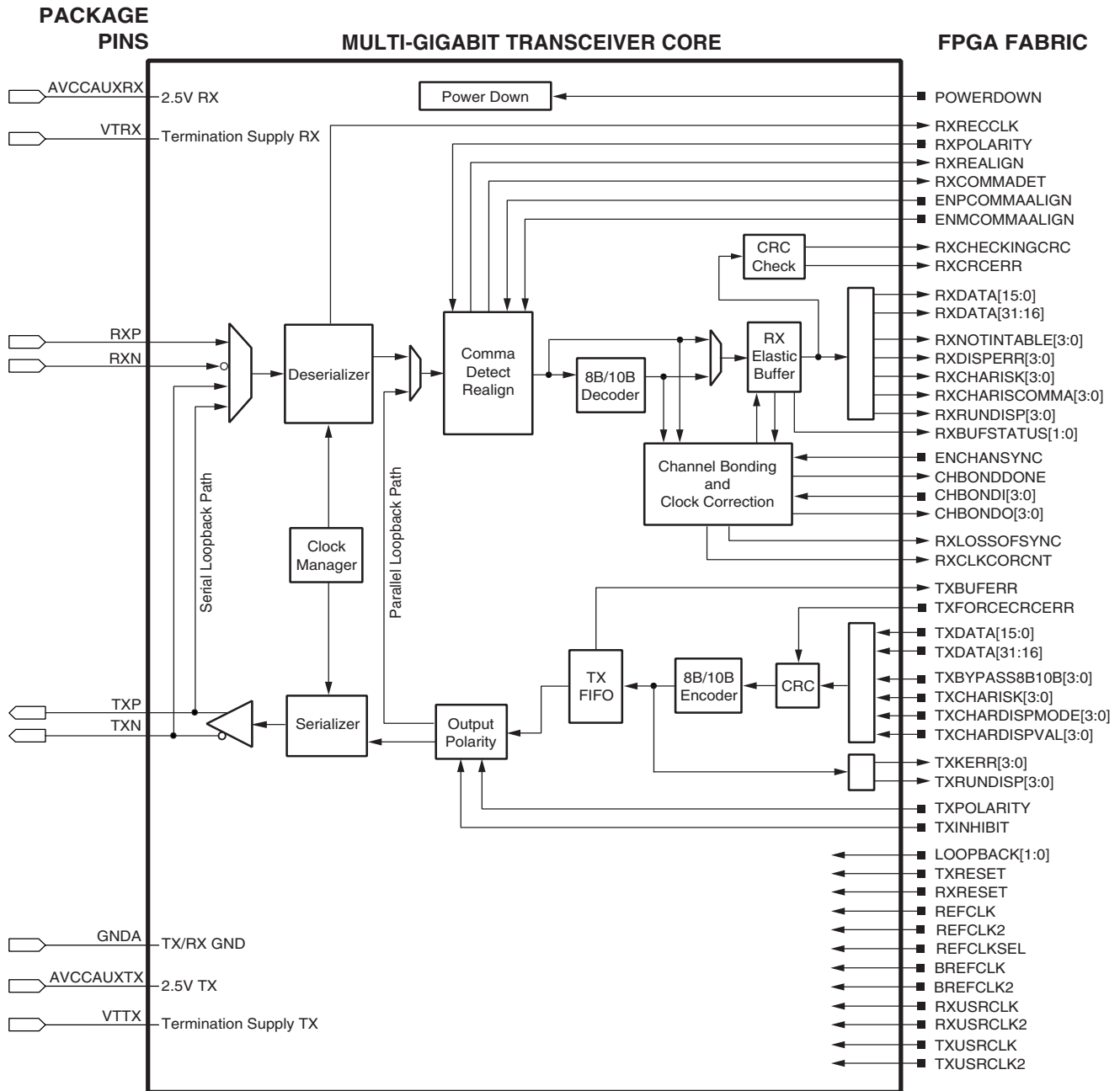
| Mode                     | Channels (Lanes) <sup>(1)</sup> | I/O Bit Rate (Gb/s) |
|--------------------------|---------------------------------|---------------------|
| Fibre Channel            | 1                               | 1.06                |
|                          |                                 | 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, ...                 | 0.622 – 3.125       |
| Custom Mode              | 1, 2, 3, 4, ...                 | 0.622 – 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       | 622 Mb/s – 1.0 Gb/s   |



DS083-2\_04\_090402

Figure 2-1: RocketIO 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 RocketIO 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 RocketIO 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 RocketIO 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 RocketIO 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 89, 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]
```

Refer to **Figure 3-10**, page 59, for a graphical representation of the transmitted 10-bit character.

### 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.

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 50 $\Omega$  (default) and 75 $\Omega$  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 800 mV and 1600 mV (differential peak-to-peak) in four increments of 200 mV.

# Receiver

## Deserializer

The RocketIO 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 70.) When the 8B/10B decoder is bypassed, the 10-bit character order is:

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

Refer to **Figure 3-11**, page 59, for a graphical representation of the received 10-bit character.

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 received 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.

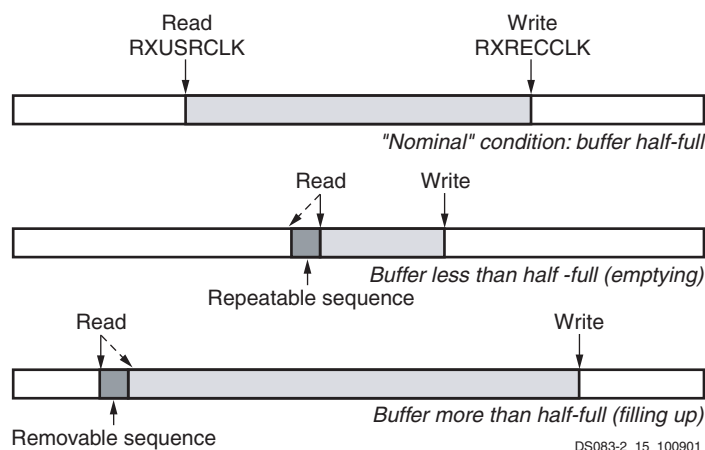


Figure 2-2: Clock Correction in Receiver

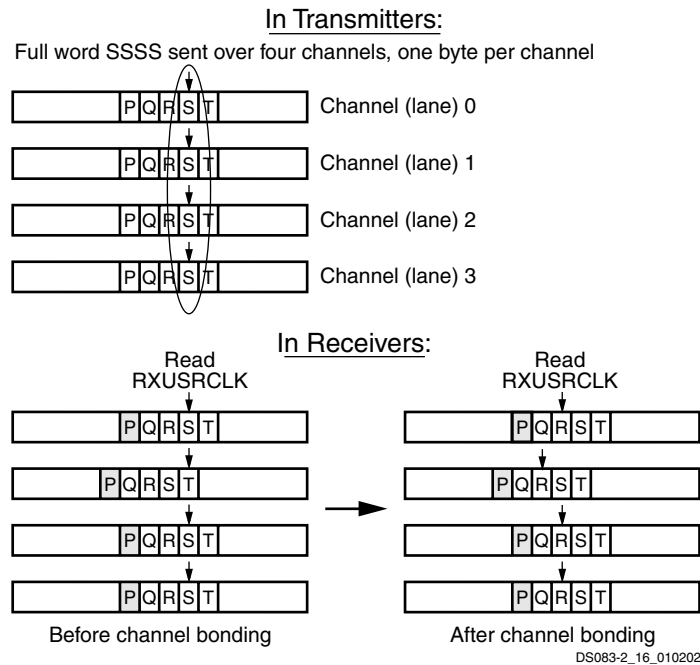
If RXUSRCLK is faster than RXRECCLK, the buffer becomes more empty over time. The clock correction logic corrects for this by decrementing the read pointer to reread a repeatable byte sequence. This is shown in the middle buffer, [Figure 2-2](#), where the solid read pointer decrements to the value represented 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 RocketIO 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 RocketIO 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 RocketIO transceiver does not compute the 16-bit variant CRC used for Infiniband. Therefore, RocketIO 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).<br>In POWERDOWN mode, transmit output pins TXP/TXN are undriven, but biased by the state of transmit termination supply VTTX. If VTTX is unpowered, TXP/TXN float to a high-impedance state. Receive input pins RXP/RXN respond similarly to the state of receive termination supply VTRX. |

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

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



## Digital Design Considerations

### List of Available Ports

The RocketIO 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<sup>(1)</sup>, GT\_AURORA, GT\_FIBRE\_CHAN<sup>(2)</sup>, GT\_ETHERNET<sup>(2)</sup>, GT\_INFINIBAND, and GT\_XAUI Primitive Ports

| Port                      | I/O | Port Size | Definition  |
|---------------------------|-----|-----------|---|
| BREFCLK(5)                | I   | 1         | This high quality reference clock uses dedicated routing to improve jitter for serial speeds 2.5 Gb/s or greater.                           |
| BREFCLK2(5)               | I   | 1         | Alternative to BREFCLK.   |
| CHBONDDONE <sup>(2)</sup> | O   | 1         | Indicates a receiver has successfully completed channel bonding when asserted High.   |
| CHBONDI <sup>(2)</sup>    | I   | 4         | The channel bonding control that is used only by "slaves" which is driven by a transceiver's CHBONDO port.                                  |
| CHBONDO <sup>(2)</sup>    | 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 <sup>(2)</sup> | 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.                              |

Table 3-1: GT\_CUSTOM<sup>(1)</sup>, GT\_AURORA, GT\_FIBRE\_CHAN<sup>(2)</sup>, GT\_ETHERNET<sup>(2)</sup>, GT\_INFINIBAND, and GT\_XAUI Primitive Ports (Continued)

| Port                         | I/O | Port Size | Definition  |
|------------------------------|-----|-----------|---|
| 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. This input is asynchronous.  |
| 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 $\pm 100$ 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 <sup>(3)</sup> | O   | 1, 2, 4   | Similar to RXCHARISK except that the data is a comma.   |
| RXCHARISK <sup>(3)</sup>     | 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 is bypassed, it remains as the first bit received (Bit "a") of the 10-bit encoded data (see <a href="#">Figure 3-11</a> ).   |
| 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 <a href="#">Clock Correction Count, page 56</a> .  |
| 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 <sup>(3)</sup>        | 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.  |
| RXLOSSOFFSYNC                | O   | 2         | Status related to byte-stream synchronization (RX_LOSS_OF_SYNC_FSM)<br>If RX_LOSS_OF_SYNC_FSM = TRUE, this outputs the state of the FSM.<br>Bit 1 signals a loss of sync.<br>Bit 0 indicates a resync state.<br>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 <sup>(4)</sup>           | I   | 1         | Serial differential port (FPGA external)  |
| RXNOTINTABLE <sup>(3)</sup>  | 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 <sup>(4)</sup>           | I   | 1         | Serial differential port (FPGA external)  |

Table 3-1: GT\_CUSTOM<sup>(1)</sup>, GT\_AURORA, GT\_FIBRE\_CHAN<sup>(2)</sup>, GT\_ETHERNET<sup>(2)</sup>, GT\_INFINIBAND, and GT\_XAUI Primitive Ports (Continued)

| Port                          | I/O | Port Size | Definition   |
|-------------------------------|-----|-----------|--|
| RXPOLARITY                    | I   | 1         | Similar to TXPOLARITY, but for RXN and RXP. When deasserted, assumes regular polarity. When asserted, reverses polarity.   |
| 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 <sup>(3)</sup>      | O   | 1, 2, 4   | Signals the running disparity (0 = negative, 1 = positive) in the received serial data. If 8B/10B encoding is bypassed, it remains as the second bit received (Bit "b") of the 10-bit encoded data (see <a href="#">Figure 3-11</a> ).   |
| 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 <sup>(3)</sup>  | 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 <sup>(3)</sup> | 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 first bit transmitted (Bit "a") of the 10-bit encoded TXDATA bus section (see <a href="#">Figure 3-10</a> ) for each byte specified by the byte-mapping.          |
| TXCHARDISPVAL <sup>(3)</sup>  | 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 second bit transmitted (Bit "b") of the 10-bit encoded TXDATA bus section (see <a href="#">Figure 3-10</a> ) for each byte specified by the byte-mapping section. |
| TXCHARISK <sup>(3)</sup>      | 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 <sup>(3)</sup>         | 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.   |

Table 3-1: GT\_CUSTOM<sup>(1)</sup>, GT\_AURORA, GT\_FIBRE\_CHAN<sup>(2)</sup>, GT\_ETHERNET<sup>(2)</sup>, GT\_INFINIBAND, and GT\_XAUI Primitive Ports (Continued)

| Port                     | I/O | Port Size | Definition   |
|--------------------------|-----|-----------|--|
| 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   |
| TXKERR <sup>(3)</sup>    | 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 <sup>(4)</sup>       | O   | 1         | Transmit differential port (FPGA external)   |
| TXP <sup>(4)</sup>       | 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 <sup>(3)</sup> | 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 and 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 size changes with relation to the primitive selected, and also correlates to the byte mapping.
4. External ports only accessible from package pins.
5. Will be available in ISE 5.1.

## 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: RocketIO Transceiver Attributes

| Attribute       | Description   |
|-----------------|---|
| ALIGN_COMMA_MSB | <p>TRUE/FALSE controls the alignment of detected commas within the transceiver's 2-byte-wide data path.</p> <p><b>FALSE:</b> 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.</p> <p><b>TRUE:</b> Aligns comma with 20-bit alignment range.</p> <p>As a result aligns on the RXDATA[15:8] byte.</p> <p><b>NOTE:</b> 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.</p> <p><b>NOTE:</b> For 32-bit data path primitives, see <a href="#">32-bit Alignment Design, page 76</a>.</p> |
| CHAN_BOND_LIMIT | <p>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.</p>  |
| CHAN_BOND_MODE  | <p>STRING<br/>OFF, MASTER, SLAVE_1_HOP, SLAVE_2_HOPS</p> <p><b>OFF:</b> No channel bonding involving this transceiver.</p> <p><b>MASTER:</b> This transceiver is master for channel bonding. Its CHBONDO port directly drives CHBONDI ports on one or more SLAVE_1_HOP transceivers.</p> <p><b>SLAVE_1_HOP:</b> 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.</p> <p><b>SLAVE_2_HOPS:</b> This transceiver is a slave for channel bonding. SLAVE_2_HOPS CHBONDI is directly driven by a SLAVE_1_HOP CHBONDO port.</p>                               |

Table 3-2: RocketIO Transceiver Attributes (Continued)

| Attribute                | Description  |
|--------------------------|--|
| CHAN_BOND_OFFSET         | <p>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.</p> <p>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.</p> <p>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.</p> <p>For optimal performance of the elastic buffer, CHAN_BOND_WAIT and CHAN_BOND_OFFSET should be set to the same value (typically 8).</p> |
| CHAN_BOND_ONE_SHOT       | <p>TRUE/FALSE that controls repeated execution of channel bonding.</p> <p><b>FALSE:</b> 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.</p> <p><b>TRUE:</b> 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.</p> <p>Slave transceivers should always have CHAN_BOND_ONE_SHOT set to FALSE.</p>  |
| CHAN_BOND_SEQ_*_*        | <p>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 <b>Receiving Vitesse Channel Bonding Sequence, page 60</b>, for format.</p>  |
| CHAN_BOND_SEQ_2_USE      | <p>Controls use of second channel bonding sequence.</p> <p><b>FALSE:</b> Channel bonding uses only one channel bonding sequence defined by CHAN_BOND_SEQ_1_1.4.</p> <p><b>TRUE:</b> Channel bonding uses two channel bonding sequences defined by:</p> <p style="padding-left: 40px;">CHAN_BOND_SEQ_1_1.4 and<br/>CHAN_BOND_SEQ_2_1.4</p> <p>as further constrained by CHAN_BOND_SEQ_LEN.</p>  |
| CHAN_BOND_SEQ_LEN        | <p>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.</p>   |
| CHAN_BOND_WAIT           | <p>Integer 1-15 that defines the length of wait (in bytes) after seeing channel bonding sequence before executing channel bonding.</p>   |
| CLK_COR_INSERT_IDLE_FLAG | <p>TRUE/FALSE controls whether RXRUNDISP input status denotes running disparity or inserted-idle flag.</p> <p><b>FALSE:</b> RXRUNDISP denotes running disparity when RXDATA is decoded data.</p> <p><b>TRUE:</b> RXRUNDISP is raised for the first byte of each inserted (repeated) clock correction ("Idle") sequence (when RXDATA is decoded data).</p>  |

Table 3-2: RocketIO Transceiver Attributes (Continued)

| Attribute           | Description  |
|---------------------|--|
| CLK_COR_KEEP_IDLE   | TRUE/FALSE controls whether or not the final byte stream must retain at least one clock correction sequence.<br><br><b>FALSE:</b> Transceiver can remove all clock correction sequences to further recenter the elastic buffer during clock correction.<br><br><b>TRUE:</b> 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.  |
| 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.<br><br><b>FALSE:</b> Clock correction uses only one clock correction sequence defined by CLK_COR_SEQ_1_1..4.<br><br><b>TRUE:</b> 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.<br><br><b>FALSE:</b> Permanently disable execution of clock correction (rate matching). Clock RXUSRCLK must be frequency-locked with RXRECCLK in this case.<br><br><b>TRUE:</b> 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      | NOTE: This attribute is only valid when CRC_FORMAT = USER_MODE. 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    | NOTE: This attribute is only valid when CRC_FORMAT = USER_MODE. 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.   |

Table 3-2: RocketIO Transceiver Attributes (Continued)

| Attribute                       | Description  |
|---------------------------------|--|
| 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.  |
| DEC_VALID_COMMA_ONLY            | TRUE/FALSE controls the raising of RXCHARISCOMMA on an invalid comma.<br><b>FALSE:</b> Raise RXCHARISCOMMA on:<br>0011111xxx<br>(if DEC_PCOMMA_DETECT is TRUE)<br>and/or on:<br>1100000xxx<br>(if DEC_MCOMMA_DETECT is TRUE)<br>regardless of the settings of the xxx bits.<br><b>TRUE:</b> 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.   |
| REF_CLK_V_SEL <sup>(1)</sup>    | 1/0:<br>1: Selects the BREFCLKs for 2.5 Gb/s or greater serial speeds.<br>0: Selects the REFCLK for serial speeds under 2.5 Gb/s.  |
| RX_BUFFER_USE                   | Always set to TRUE.  |
| RX_CRC_USE,<br>TX_CRC_USE       | TRUE/FALSE determines if CRC is used or not.   |
| RX_DATA_WIDTH,<br>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.  |



Table 3-2: RocketIO Transceiver Attributes (Continued)

| Attribute           | Description  |
|---------------------|--|
| RX_LOSS_OF_SYNC_FSM | TRUE/FALSE denotes the nature of RXLOSSOFSYNC output.<br><b>TRUE:</b> RXLOSSOFSYNC outputs the state of the FSM bit.<br>See <a href="#">RXLOSSOFSYNC</a> , 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 622 Mb/s to 1.0 Gb/s. See <a href="#">Half-Rate Clocking Scheme</a> , page 52. |
| 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 <a href="#">Table 4-2</a> , page 84.  |

**Notes:**

1. Will be available in ISE 5.1.

## 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 <sup>(1)</sup> | Default GT_ETHERNET |
|--------------------|----------------------|----------------------------------|---------------------|
| ALIGN_COMMA_MSB    | FALSE                | FALSE                            | FALSE               |
| CHAN_BOND_LIMIT    | 16                   | 16                               | 1                   |
| CHAN_BOND_MODE     | OFF <sup>(2)</sup>   | OFF                              | OFF                 |
| CHAN_BOND_OFFSET   | 8                    | 8                                | 0                   |
| CHAN_BOND_ONE_SHOT | FALSE <sup>(2)</sup> | FALSE                            | TRUE                |
| CHAN_BOND_SEQ_1_1  | 00101111100          | 00000000000                      | 00000000000         |
| CHAN_BOND_SEQ_1_2  | 00000000000          | 00000000000                      | 00000000000         |

Table 3-3: Default Attribute Values: GT\_AURORA, GT\_CUSTOM, GT\_ETHERNET (Continued)

| Attribute                | Default<br>GT_AURORA       | Default<br>GT_CUSTOM <sup>(1)</sup> | Default<br>GT_ETHERNET |
|--------------------------|----------------------------|-------------------------------------|------------------------|
| CHAN_BOND_SEQ_1_3        | 0000000000                 | 0000000000                          | 0000000000             |
| CHAN_BOND_SEQ_1_4        | 0000000000                 | 0000000000                          | 0000000000             |
| CHAN_BOND_SEQ_2_1        | 0000000000                 | 0000000000                          | 0000000000             |
| CHAN_BOND_SEQ_2_2        | 0000000000                 | 0000000000                          | 0000000000             |
| CHAN_BOND_SEQ_2_3        | 0000000000                 | 0000000000                          | 0000000000             |
| CHAN_BOND_SEQ_2_4        | 0000000000                 | 0000000000                          | 0000000000             |
| 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 <sup>(2)</sup>       | FALSE                               | FALSE <sup>(2)</sup>   |
| CLK_COR_KEEP_IDLE        | FALSE <sup>(2)</sup>       | FALSE                               | FALSE <sup>(2)</sup>   |
| CLK_COR_REPEAT_WAIT      | 1 <sup>(2)</sup>           | 1                                   | 1 <sup>(2)</sup>       |
| CLK_COR_SEQ_1_1          | 00100011100                | 0000000000                          | 00110111100            |
| CLK_COR_SEQ_1_2          | 00100011100 <sup>(4)</sup> | 0000000000                          | 00001010000            |
| CLK_COR_SEQ_1_3          | 00100011100 <sup>(5)</sup> | 0000000000                          | 0000000000             |
| CLK_COR_SEQ_1_4          | 00100011100 <sup>(5)</sup> | 0000000000                          | 0000000000             |
| CLK_COR_SEQ_2_1          | 0000000000                 | 0000000000                          | 0000000000             |
| CLK_COR_SEQ_2_2          | 0000000000                 | 0000000000                          | 0000000000             |
| CLK_COR_SEQ_2_3          | 0000000000                 | 0000000000                          | 0000000000             |
| CLK_COR_SEQ_2_4          | 0000000000                 | 0000000000                          | 0000000000             |
| CLK_COR_SEQ_2_USE        | FALSE                      | FALSE                               | FALSE                  |
| CLK_COR_SEQ_LEN          | N <sup>(3)</sup>           | 1                                   | 2                      |
| CLK_CORRECT_USE          | TRUE                       | TRUE                                | TRUE                   |
| COMMA_10B_MASK           | 1111111111                 | 1111111000                          | 1111111000             |
| CRC_END_OF_PKT           | K29_7                      | K29_7                               | Note (7)               |
| CRC_FORMAT               | USER_MODE                  | USER_MODE                           | ETHERNET               |
| CRC_START_OF_PKT         | K27_7                      | K27_7                               | Note (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             |

Table 3-3: Default Attribute Values: GT\_AURORA, GT\_CUSTOM, GT\_ETHERNET (Continued)

| Attribute                    | Default<br>GT_AURORA    | Default<br>GT_CUSTOM <sup>(1)</sup> | Default<br>GT_ETHERNET  |
|------------------------------|-------------------------|-------------------------------------|-------------------------|
| PCOMMA_DETECT                | TRUE                    | TRUE                                | TRUE                    |
| REF_CLK_V_SEL <sup>(6)</sup> | 0                       | 0                                   | 0                       |
| RX_BUFFER_USE                | TRUE                    | TRUE                                | TRUE                    |
| RX_CRC_USE                   | FALSE <sup>(2)</sup>    | FALSE                               | FALSE <sup>(2)</sup>    |
| RX_DATA_WIDTH                | N <sup>(3)</sup>        | 2                                   | N <sup>(3)</sup>        |
| RX_DECODE_USE                | TRUE                    | TRUE                                | TRUE                    |
| RX_LOS_INVALID_INCR          | 1 <sup>(2)</sup>        | 1                                   | 1 <sup>(2)</sup>        |
| RX_LOS_THRESHOLD             | 4 <sup>(2)</sup>        | 4                                   | 4 <sup>(2)</sup>        |
| RX_LOSS_OF_SYNC_FSM          | TRUE <sup>(2)</sup>     | TRUE                                | TRUE <sup>(2)</sup>     |
| SERDES_10B                   | FALSE <sup>(2)</sup>    | FALSE                               | FALSE <sup>(2)</sup>    |
| TERMINATION_IMP              | 50 <sup>(2)</sup>       | 50                                  | 50 <sup>(2)</sup>       |
| TX_BUFFER_USE                | TRUE                    | TRUE                                | TRUE                    |
| TX_CRC_FORCE_VALUE           | 11010110 <sup>(2)</sup> | 11010110                            | 11010110 <sup>(2)</sup> |
| TX_CRC_USE                   | FALSE <sup>(2)</sup>    | FALSE                               | FALSE <sup>(2)</sup>    |
| TX_DATA_WIDTH                | N <sup>(3)</sup>        | 2                                   | N <sup>(3)</sup>        |
| TX_DIFF_CTRL                 | 500 <sup>(2)</sup>      | 500                                 | 500 <sup>(2)</sup>      |
| TX_PREEMPHASIS               | 0 <sup>(2)</sup>        | 0                                   | 0 <sup>(2)</sup>        |

**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.
6. Will be available in ISE 5.1.
7. CRC\_EOP and CRC\_SOP are not applicable for this primitive.

Table 3-4: Default Attribute Values: GT\_FIBRE\_CHAN, GT\_INFINIBAND, and GT\_XAUI

| Attribute          | Default<br>GT_FIBRE_CHAN | Default<br>GT_INFINIBAND | Default<br>GT_XAUI   |
|--------------------|--------------------------|--------------------------|----------------------|
| ALIGN_COMMA_MSB    | FALSE                    | FALSE                    | FALSE                |
| CHAN_BOND_LIMIT    | 1                        | 16                       | 16                   |
| CHAN_BOND_MODE     | OFF                      | OFF <sup>(1)</sup>       | OFF <sup>(1)</sup>   |
| CHAN_BOND_OFFSET   | 0                        | 8                        | 8                    |
| CHAN_BOND_ONE_SHOT | TRUE                     | FALSE <sup>(1)</sup>     | FALSE <sup>(1)</sup> |
| CHAN_BOND_SEQ_1_1  | 000000000000             | 001101111100             | 001011111100         |

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          |
|--------------------------|-----------------------|-------------------------------|--------------------------|
| CHAN_BOND_SEQ_1_2        | 0000000000            | Lane ID (Modify with Lane ID) | 0000000000               |
| CHAN_BOND_SEQ_1_3        | 0000000000            | 00001001010                   | 0000000000               |
| CHAN_BOND_SEQ_1_4        | 0000000000            | 00001001010                   | 0000000000               |
| CHAN_BOND_SEQ_2_1        | 0000000000            | 00110111100                   | 0000000000               |
| CHAN_BOND_SEQ_2_2        | 0000000000            | Lane ID (Modify with Lane ID) | 0000000000               |
| CHAN_BOND_SEQ_2_3        | 0000000000            | 00001000101                   | 0000000000               |
| CHAN_BOND_SEQ_2_4        | 0000000000            | 00001000101                   | 0000000000               |
| 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 <sup>(1)</sup>  | FALSE <sup>(1)</sup>          | FALSE <sup>(1)</sup>     |
| CLK_COR_KEEP_IDLE        | FALSE <sup>(1)</sup>  | FALSE <sup>(1)</sup>          | FALSE <sup>(1)</sup>     |
| CLK_COR_REPEAT_WAIT      | 2 <sup>(1)</sup>      | 1 <sup>(1)</sup>              | 1 <sup>(1)</sup>         |
| 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           | Note (3)              | Note (3)                      | K29_7 <sup>(1)</sup>     |
| CRC_FORMAT               | FIBRE_CHAN            | INFINIBAND                    | USER_MODE <sup>(1)</sup> |
| CRC_START_OF_PKT         | Note (3)              | Note (3)                      | K27_7 <sup>(1)</sup>     |
| DEC_MCOMMA_DETECT        | TRUE                  | TRUE                          | TRUE                     |
| DEC_PCOMMA_DETECT        | TRUE                  | TRUE                          | TRUE                     |
| DEC_VALID_COMMA_ONLY     | TRUE                  | TRUE                          | TRUE                     |

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         |
|------------------------------|-------------------------|----------------------------|-------------------------|
| Lane ID(INFINIBAND ONLY)     | NA                      | 00000000000 <sup>(1)</sup> | NA                      |
| MCOMMA_10B_VALUE             | 1100000000              | 1100000000                 | 1100000000              |
| MCOMMA_DETECT                | TRUE                    | TRUE                       | TRUE                    |
| PCOMMA_10B_VALUE             | 0011111000              | 0011111000                 | 0011111000              |
| PCOMMA_DETECT                | TRUE                    | TRUE                       | TRUE                    |
| REF_CLK_V_SEL <sup>(4)</sup> | 0                       | 0                          | 0                       |
| RX_BUFFER_USE                | TRUE                    | TRUE                       | TRUE                    |
| RX_CRC_USE                   | FALSE <sup>(1)</sup>    | FALSE <sup>(1)</sup>       | FALSE <sup>(1)</sup>    |
| RX_DATA_WIDTH                | N <sup>(2)</sup>        | N <sup>(2)</sup>           | N <sup>(2)</sup>        |
| RX_DECODE_USE                | TRUE                    | TRUE                       | TRUE                    |
| RX_LOS_INVALID_INCR          | 1 <sup>(1)</sup>        | 1 <sup>(1)</sup>           | 1 <sup>(1)</sup>        |
| RX_LOS_THRESHOLD             | 4 <sup>(1)</sup>        | 4 <sup>(1)</sup>           | 4 <sup>(1)</sup>        |
| RX_LOSS_OF_SYNC_FSM          | TRUE <sup>(1)</sup>     | TRUE <sup>(1)</sup>        | TRUE <sup>(1)</sup>     |
| SERDES_10B                   | FALSE <sup>(1)</sup>    | FALSE <sup>(1)</sup>       | FALSE <sup>(1)</sup>    |
| TERMINATION_IMP              | 50 <sup>(1)</sup>       | 50 <sup>(1)</sup>          | 50 <sup>(1)</sup>       |
| TX_BUFFER_USE                | TRUE                    | TRUE                       | TRUE                    |
| TX_CRC_FORCE_VALUE           | 11010110 <sup>(1)</sup> | 11010110 <sup>(1)</sup>    | 11010110 <sup>(1)</sup> |
| TX_CRC_USE                   | FALSE <sup>(1)</sup>    | FALSE <sup>(1)</sup>       | FALSE <sup>(1)</sup>    |
| TX_DATA_WIDTH                | N <sup>(2)</sup>        | N <sup>(2)</sup>           | N <sup>(2)</sup>        |
| TX_DIFF_CTRL                 | 500 <sup>(1)</sup>      | 500 <sup>(1)</sup>         | 500 <sup>(1)</sup>      |
| TX_PREEMPHASIS               | 0 <sup>(1)</sup>        | 0 <sup>(1)</sup>           | 0 <sup>(1)</sup>        |

**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.
4. Will be available in ISE 5.1.

## 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 RocketIO transceiver instantiation (Table 3-6). REFCLK is a clock generated from an external source. REFCLK is connected to the REFCLK of the RocketIO 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 RocketIO transceiver.

**NOTE:** The REFCLK must be at least 40 MHz (full-rate operation only; 62.2 MHz for half-rate operation) 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   |
|-------------------------|--------|---|
| BREFCLK <sup>(1)</sup>  | Input  | Reference clock used to read the TX FIFO and multiplied by 20 for parallel-to-serial conversion (20X)   |
| BREFCLK2 <sup>(1)</sup> | Input  | Alternative to BREFCLK  |
| 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. |

Table 3-6: Clock Ports (Continued)

| Clock     | I/Os  | Description   |
|-----------|-------|---|
| 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. |

**Notes:**

- Will be available in ISE 5.1.

## 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 <sup>(1)</sup>                |
| 2 byte     | 1:1                               |
| 4 byte     | 2:1 <sup>(1)</sup>                |

**Notes:**

- 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<br>RX_DATA_WIDTH | REFCLK | TXUSRCLK<br>RXUSRCLK   | TXUSRCLK2<br>RXUSRCLK2 |
|------------|--------------------------------|--------|------------------------|------------------------|
| FALSE      | 1                              | CLKIN  | CLK0                   | CLK2X180               |
| FALSE      | 2                              | CLKIN  | CLK0                   | CLK0                   |
| FALSE      | 4                              | CLKIN  | CLK180 <sup>(1)</sup>  | CLKDV (divide by 2)    |
| TRUE       | 1                              | CLKIN  | CLKDV (divide by 2)    | CLK180 <sup>(1)</sup>  |
| TRUE       | 2                              | CLKIN  | CLKDV (divide by 2)    | CLKDV (divide by 2)    |
| TRUE       | 4                              | CLKIN  | CLKFX180 (divide by 2) | CLKDV (divide by 4)    |

**Notes:**

- 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.

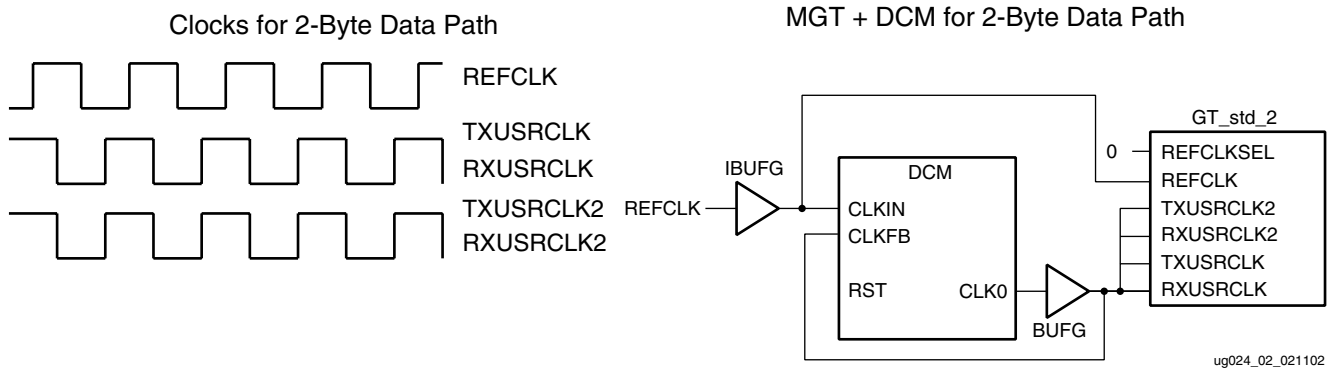


Figure 3-1: Two-Byte Clock

#### 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;
```



```

    );
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_BUF: 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 dcml (
    .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 takes 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 USRCLK\_M is 180° out of phase when using local inverters with the transceiver.

**NOTE:** These local MGT clock input inverters, shown and noted in Figure 3-2, are *not* included in the FOUR\_BYTE\_CLK templates.

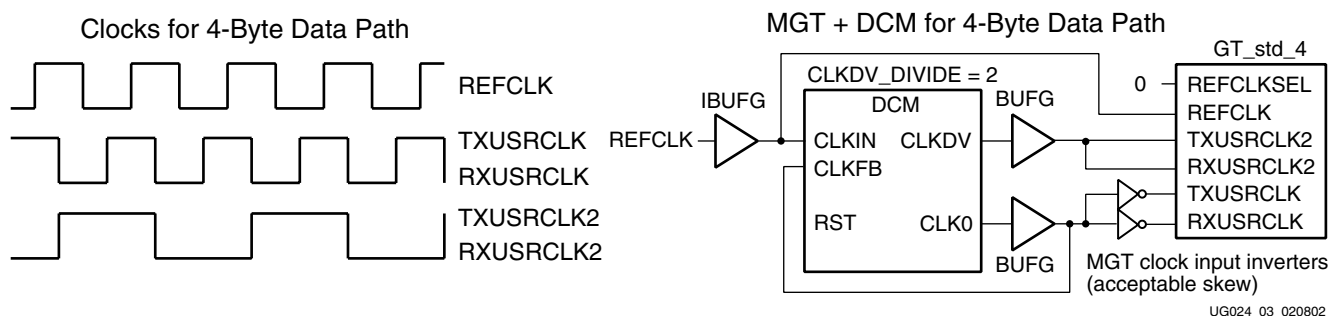


Figure 3-2: Four-Byte Clock

## 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;
        DSSSEN     : 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,
        DSSSEN   => 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_BUF: BUF
port map (
    I => CLK0_W,
    O => USRCLK_M
    );

U3_BUF: BUF
port map (
    I => CLKDV_W,
    O => USRCLK2_M
    );

end FOUR_BYTE_CLK_arch;

```

### Verilog Template

```

// 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 dcml (
    .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 )
    );

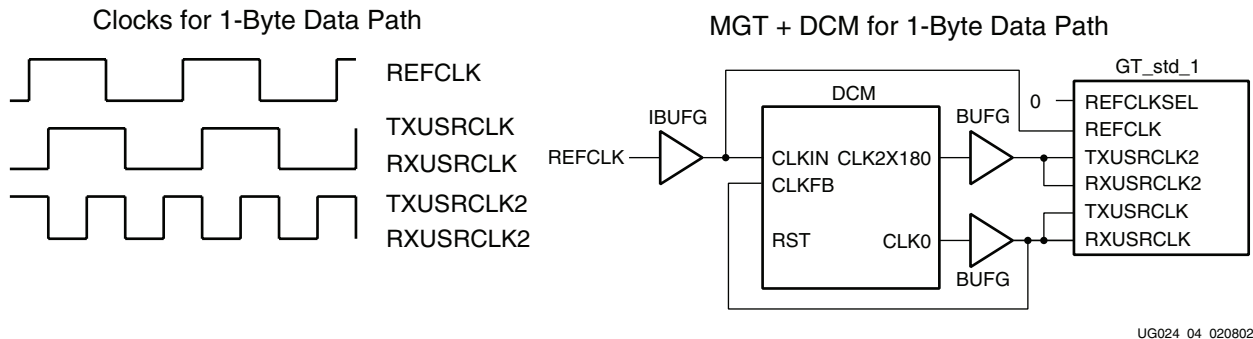
    IBUFG buf3 (
        .I ( REFCLKIN ),
        .O ( REFCLKINBUF ) );

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.



UG024\_04\_020802

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,
    PSCCLK     => 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

```

## BREFCLK

At speeds of 2.5 Gb/s or greater, the REFCLK configuration introduces more than the maximum allowable jitter to the RocketIO transceiver. For these higher speeds, the BREFCLK configuration is required. The BREFCLK configuration uses dedicated routing resources that reduce jitter.

BREFCLK must enter the FPGA through dedicated clock I/O. This BREFCLK can connect to the BREFCLK inputs of the transceiver and the CLKIN input of the DCM for creation of USRCLKs. If all the transceivers on a Virtex-II Pro FPGA are to be used, two BREFCLKs must be created, one for the top of the chip and one for the bottom.

These dedicated clocks use the same clock inputs for all packages: .

|        |          |   |        |
|--------|----------|---|--------|
| Top    | BREFCLK  | P | GCLK4S |
|        |          | N | GCLK5P |
|        | BREFCLK2 | P | GCLK2S |
|        |          | N | GCLK3P |
| Bottom | BREFCLK  | P | GCLK6P |
|        |          | N | GCLK7S |
|        | BREFCLK2 | P | GCLK0P |
|        |          | N | GCLK1S |

Figure 3-4 shows how REFCLK and BREFCLK are selected through use of REFCLKSEL and REF\_CLK\_V\_SEL.

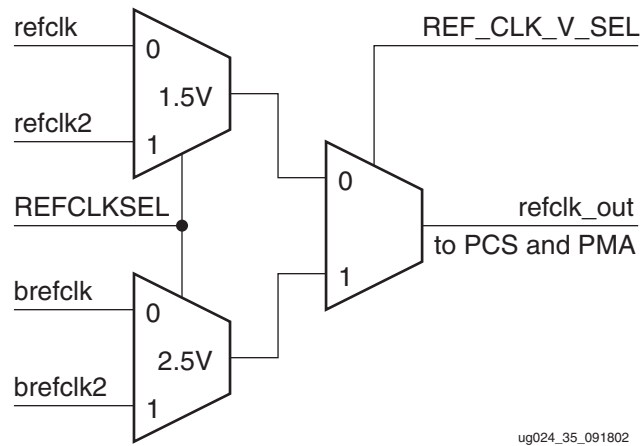


Figure 3-4: REFCLK/BREFCLK Selection Logic

Table 3-9 shows the BREFCLK pin numbers for all packages. Note that these pads must be used for BREFCLK operations

Table 3-9: BREFCLK Pin Numbers

| Package | Top                |                     | Bottom             |                     |
|---------|--------------------|---------------------|--------------------|---------------------|
|         | BREFCLK Pin Number | BREFCLK2 Pin Number | BREFCLK Pin Number | BREFCLK2 Pin Number |
| FG256   | A8/B8              | B9/A9               | R8/T8              | T9/R9               |
| FG456   | C11/D11            | D12/C12             | W11/Y11            | Y12/W12             |
| FF672   | B14/C14            | C13/B13             | AD14/AE14          | AE13/AD13           |
| FF896   | F16/G16            | G15/F15             | AH16/AJ16          | AJ15/AH15           |
| FF1152  | H18/J18            | J17/H17             | AK18/AL18          | AL17/AK17           |
| FF1148  | N/A                | N/A                 | N/A                | N/A                 |
| FF1517  | E20/D20            | J20/K20             | AR20/AT20          | AL20/AK20           |
| FF1704  | G22/F22            | F21/G21             | AU22/AT22          | AT21/AU21           |
| FF1696  | N/A                | N/A                 | N/A                | N/A                 |

## Half-Rate Clocking Scheme

Some applications require serial speeds between 622 Mb/s and 1 Gb/s. The transceiver attribute SERDES\_10B, which sets the REFCLK multiplier to 10 instead of 20, enables the half-rate speed range when set to TRUE. With this configuration, the clocking scheme also changes. The figures below illustrate the three clocking scheme waveforms when SERDES\_10B = TRUE.

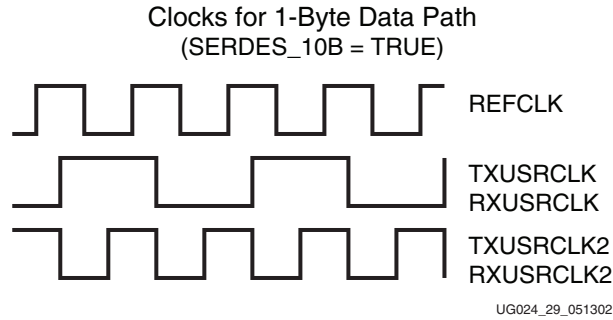


Figure 3-5: One-Byte Data Path Clocks, SERDES\_10B = TRUE

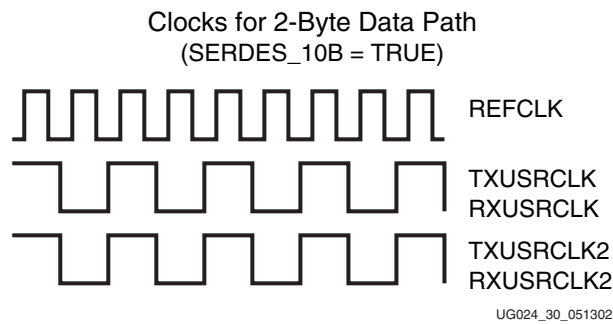


Figure 3-6: Two-Byte Data Path Clocks, SERDES\_10B = TRUE

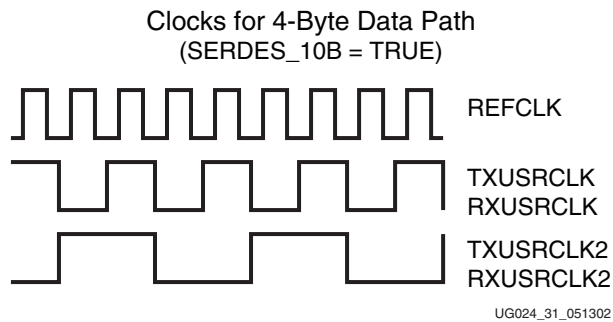


Figure 3-7: Four-Byte Data Path Clocks, SERDES\_10B = TRUE

## 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 3-8** shows how the design can use two different reference clocks connected to two different DCMs. The clocks are then multiplexed before input into the RocketIO 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.

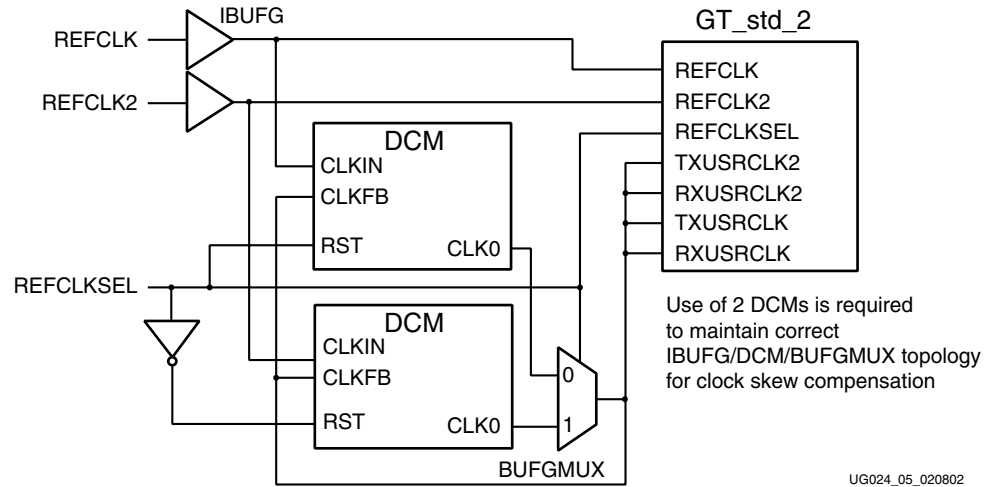


Figure 3-8: Multiplexed REFCLK

## Data Path Latency

With the many configurations of the MGT, both the transmit and receive latency of the data path varies. Below are several tables that provide approximate latencies for common configurations.

### Transmitter Latency

- Latencies for
  - TX\_BUFFER\_USE = TRUE
  - TX\_CRC\_USE = FALSE
  - SERDES\_10B = FALSE

| TX_DATA_WIDTH | Latency<br>(Approx. TXUSRCLK Cycles) |
|---------------|--------------------------------------|
| 1             | 8.75                                 |
| 2             | 8.5                                  |
| 4             | 10                                   |

- If TX FIFO bypassed (TX\_BUFFER\_USE = FALSE), subtract 4
- If TX CRC is used (TX\_CRC\_USE = TRUE), add 6
- For half-rate option (SERDES\_10B = TRUE). subtract 1 (approx.)

## Receiver Latency

- Latencies for RX\_BUFFER\_USE = TRUE

| RX_DATA_WIDTH | Latency<br>(Approx. TXUSRCLK Cycles) |
|---------------|--------------------------------------|
| 1             | 25.25                                |
| 2             | 25                                   |
| 4             | 26.5                                 |

- If RX elastic buffer is bypassed (RX\_BUFFER\_USE = FALSE), subtract 18.

## 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 further discussed and illustrated in [Appendix A, RocketIO Transceiver Timing Model](#).

## 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

```
-- 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

```

// Module:          gt_reset
// Description:     Verilog Submodule
//                 reset for 4-byte GT
//
// Device:          Virtex-II Pro Family

module gt_reset (
    USRCLK2_M,
    DCM_LOCKED,
    RST
);

input    USRCLK2_M;
input    DCM_LOCKED;
output   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

```

## RocketIO 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

For code examples, see the *Virtex-II Pro Platform FPGA Handbook* web page at: <http://www.xilinx.com/publications/products/v2pro/handbook/index.htm>

## 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 3-10 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 3-10: Clock Correction Sequence / Data Correlation for 16-Bit Data Port

| Attribute Setting              | Character | CHARISK | TXDATA (hex) |
|--------------------------------|-----------|---------|--------------|
| CLK_COR_SEQ_1_1 = 001101111100 | 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 3-11).

Table 3-11: 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 |



Table 3-11: RXCLKCORCNT Definition (Continued)

| RXCLKCORCNT[2:0] | Significance  |
|------------------|---|
| 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 RocketIO 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 3-9 shows the encoding/decoding blocks of the transceiver and how the data passes through these blocks. Table 3-12, page 58, shows the significance of 8B/10B ports that change purpose depending on whether 8B/10B is bypassed or enabled.

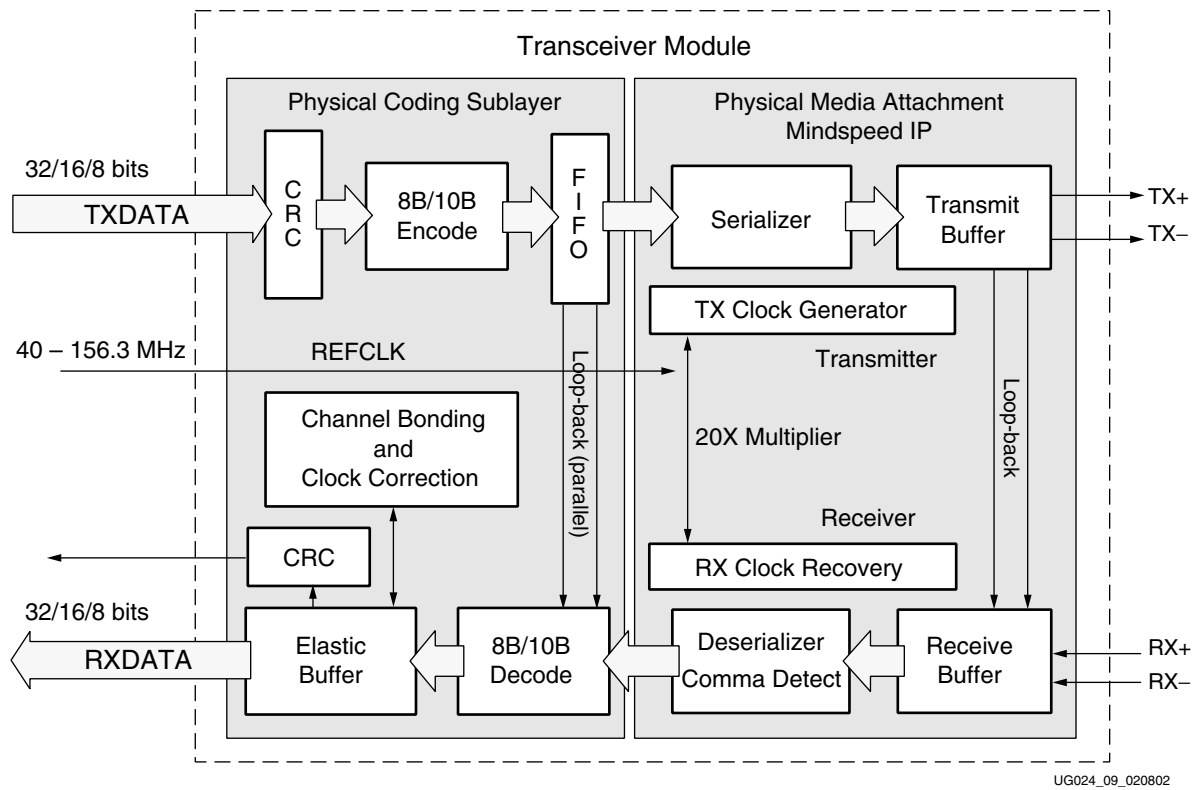


Figure 3-9: 8B/10B Data Flow

Table 3-12: 8B/10B Bypassed Signal Significance

|                               |                                | Function   |   |
|-------------------------------|--------------------------------|--|---|
| TXBYPASS8B10B                 | 0                              | 8B/10B encoding is enabled (not bypassed)                                  |   |
|                               | 1                              | 8B/10B encoding bypassed (disabled)  |   |
|                               |                                | Function, 8B/10B Enabled   | Function, 8B/10B Bypassed   |
| TXCHARDISPMODE, TXCHARDISPVAL | 00                             | Maintain running disparity normally  | Part of 10-bit encoded byte (see Figure 3-10):  |
|                               | 01                             | Invert the normally generated running disparity before encoding this byte. | TXCHARDISPMODE[0], ( or: [1] / [2] / [3] )  |
|                               | 10                             | Set negative running disparity before encoding this byte.                  | TXCHARDISPVAL[0], ( or: [1] / [2] / [3] )   |
|                               | 11                             | Set positive running disparity before encoding this byte.                  | TXDATA[7:0] ( or: [15:8] / [23:16] / [31:24] )  |
| RXCHARISK                     | Received byte is a K-character |  | Part of 10-bit encoded byte (see Figure 3-11):  |
| RXRUNDISP                     | 0                              | Indicates running disparity is NEGATIVE                                    | RXCHARISK[0], ( or: [1] / [2] / [3] )   |
|                               | 1                              | Indicates running disparity is POSITIVE                                    | RXRUNDISP[0], ( or: [1] / [2] / [3] )<br>RXDATA[7:0] ( or: [15:8] / [23:16] / [31:24] ) |

Table 3-12: 8B/10B Bypassed Signal Significance (Continued)

|               |  |        |
|---------------|--|--------|
| RXDISPERR     | Disparity error occurred on current byte | Unused |
| TXCHARISK     | Transmitted byte is a K-character        | Unused |
| RXCHARISCOMMA | Received byte is a comma                 | Unused |

During transmit, while 8B/10B encoding is enabled, the disparity of the serial transmission can be controlled with the TXCHARDISPVAL and TXCHARDISPMODE ports. When 8B/10B encoding is bypassed, these bits become Bits "b" and "a", respectively, of the 10-bit encoded data that the transceiver must transmit to the receiving terminal. Figure 3-10 illustrates the TX data map during 8B/10B bypass.

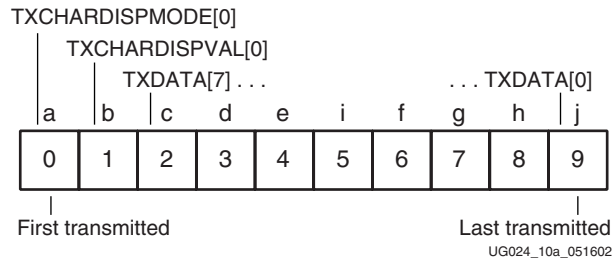


Figure 3-10: 10-Bit TX Data Map with 8B/10B Bypassed

During receive, while 8B/10B decoding is enabled, the running disparity of the serial transmission can be read by the transceiver from the RXRUNDISP port, while the RXCHARISK port indicates presence of a K-character. When 8B/10B decoding is bypassed, these bits remain as Bits "b" and "a", respectively, of the 10-bit encoded data that the transceiver passes on to the user logic. Figure 3-11 illustrates the RX data map during 8B/10B bypass.

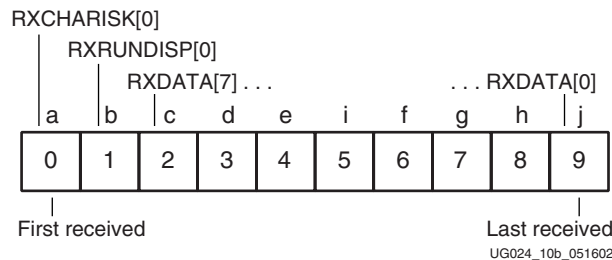


Figure 3-11: 10-Bit RX Data Map with 8B/10B Bypassed

### 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 RocketIO 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 3-13](#) and [Table 3-14](#) show the Data and K tables of valid characters.

*Table 3-13: Valid Data Characters*

| Data Byte Name | Bits<br>HGF EDCBA | Current RD –<br>abcdei fghj | Current RD +<br>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                 | 100101 0100                 |
| D10.0          | 000 01010         | 010101 1011                 | 010101 0100                 |
| D11.0          | 000 01011         | 110100 1011                 | 110100 0100                 |
| 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                 |

Table 3-13: Valid Data Characters (Continued)

| <b>Data Byte Name</b> | <b>Bits<br/>HGF EDCBA</b> | <b>Current RD –<br/>abcdei fghj</b> | <b>Current RD +<br/>abcdei fghj</b> |
|-----------------------|---------------------------|-------------------------------------|-------------------------------------|
| 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                         |
| 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                         |

Table 3-13: Valid Data Characters (Continued)

| Data Byte Name | Bits<br>HGF EDCBA | Current RD –<br>abcdei fghj | Current RD +<br>abcdei fghj |
|----------------|-------------------|-----------------------------|-----------------------------|
| 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                 |
| 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                 |

Table 3-13: Valid Data Characters (Continued)

| <b>Data Byte Name</b> | <b>Bits<br/>HGF EDCBA</b> | <b>Current RD –<br/>abcdei fghj</b> | <b>Current RD +<br/>abcdei fghj</b> |
|-----------------------|---------------------------|-------------------------------------|-------------------------------------|
| 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                         |
| 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                         |



Table 3-13: Valid Data Characters (Continued)

| <b>Data Byte Name</b> | <b>Bits<br/>HGF EDCBA</b> | <b>Current RD –<br/>abcdei fghj</b> | <b>Current RD +<br/>abcdei fghj</b> |
|-----------------------|---------------------------|-------------------------------------|-------------------------------------|
| 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                         |
| 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                         |

Table 3-13: Valid Data Characters (Continued)

| <b>Data Byte Name</b> | <b>Bits<br/>HGF EDCBA</b> | <b>Current RD –<br/>abcdei fghj</b> | <b>Current RD +<br/>abcdei fghj</b> |
|-----------------------|---------------------------|-------------------------------------|-------------------------------------|
| 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                         |
| 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                         |

Table 3-13: Valid Data Characters (Continued)

| <b>Data Byte Name</b> | <b>Bits<br/>HGF EDCBA</b> | <b>Current RD –<br/>abcdei fghj</b> | <b>Current RD +<br/>abcdei fghj</b> |
|-----------------------|---------------------------|-------------------------------------|-------------------------------------|
| 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                         |
| 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                         |

Table 3-13: Valid Data Characters (Continued)

| <b>Data Byte Name</b> | <b>Bits<br/>HGF EDCBA</b> | <b>Current RD –<br/>abcdei fghj</b> | <b>Current RD +<br/>abcdei fghj</b> |
|-----------------------|---------------------------|-------------------------------------|-------------------------------------|
| 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                         |
| 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                         |

Table 3-13: Valid Data Characters (Continued)

| Data Byte Name | Bits<br>HGF EDCBA | Current RD –<br>abcdei fghj | Current RD +<br>abcdei fghj |
|----------------|-------------------|-----------------------------|-----------------------------|
| 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 3-14: Valid Control “K” Characters

| Special Code Name    | Bits<br>HGF EDCBA | Current RD –<br>abcdei fghj | Current RD +<br>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 <sup>(1)</sup> | 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                 |

**Notes:**

- Used for testing and characterization only.

## 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 3-12. 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 3-13.

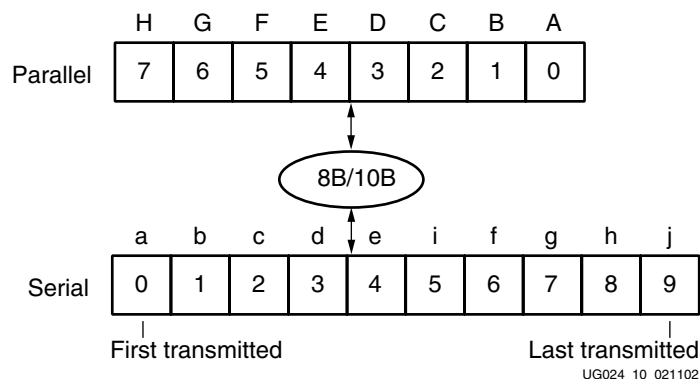


Figure 3-12: 8B/10B Parallel to Serial Conversion

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 3-13** 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].

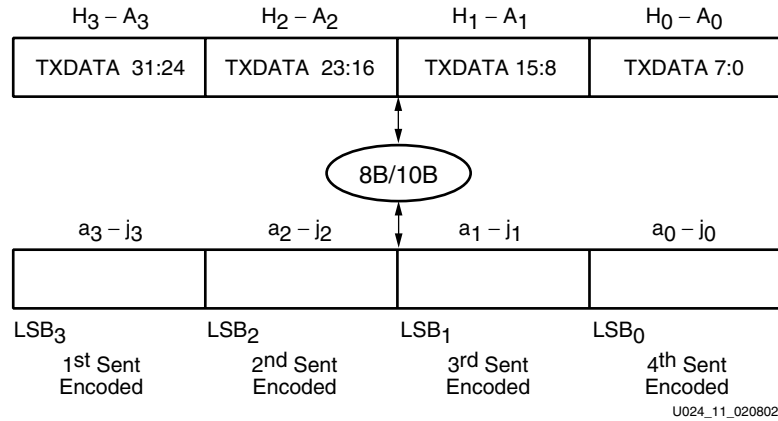


Figure 3-13: 4-Byte Serial Structure

### 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 3-14**). 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

RocketIO 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 3-14** 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.



UG024\_07\_021102

Figure 3-14: 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 TXUSRCLK/RXUSRCLK cycles, are shown in **Table 3-15**. For timing diagrams expressing these relationships, please see Module 3 of the **Virtex-II Pro Data Sheet**.

Table 3-15: Effects of CRC on Transceiver Latency

|              | TXDATA to TXP and TXN,<br>in TXUSRCLK Cycles |         | RXP and RXN to RXDATA,<br>in RXUSRCLK 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 RocketIO CRC. First, CRC is not supported in byte-stripped data. If byte-stripped (channel bonding) is required, CRC must be computed in CLBs prior to the byte-stripping. 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 RocketIO 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 3-15**, 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.

NOTE: Data must be a minimum of 16 bytes for user mode CRC generation.

## FIBRECHANNEL

The Fibre Channel CRC is similar to the USER\_MODE CRC (Figure 3-15) 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.

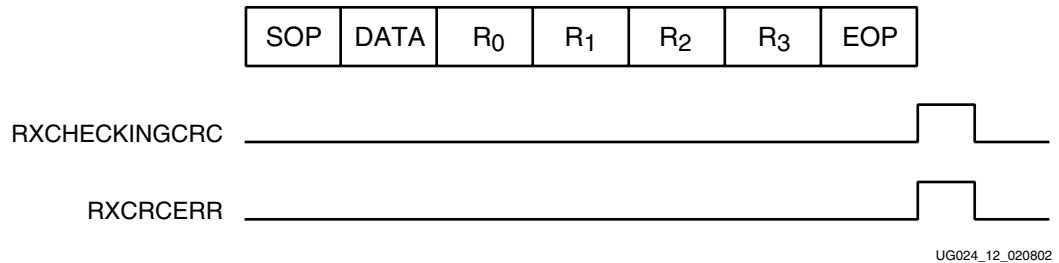


Figure 3-15: USER\_MODE / FIBRE\_CHANNEL Mode

Designs should generate only the EOF frame delimiter for a beginning running disparity (RD) that is negative; these are the frame delimiters that begin with /K28.5/D21.4/ or /K28.5/D10.4/. Never generate the EOF frame delimiter for a beginning RD that is positive; these are the frame delimiters that begin with /K28.5/D21.5/ or /K28.5/D10.5/. When the RocketIO CRC determines that the running disparity must be inverted to satisfy Fibre Channel requirements, it will convert the second byte of the EOF frame delimiter (D21.4 or D10.4) to the value required to invert the running disparity (D21.5 or D10.5).

## ETHERNET

The Ethernet CRC is more complex (Figure 3-16). 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.

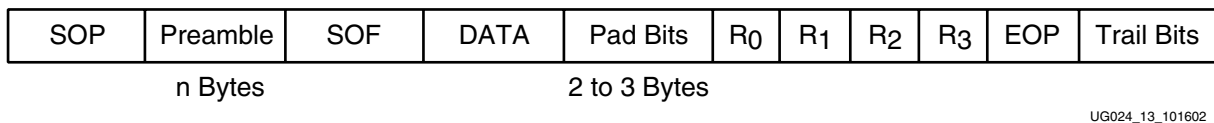


Figure 3-16: Ethernet Mode

Designs should generate only the /K28.5/D16.2/ idle sequence for transmission, never /K28.5/D5.6/. When the RocketIO CRC determines that the running disparity must be inverted to satisfy Gigabit Ethernet requirements, it will convert the first /K28.5/D16.2/ idle following a packet to /K28.5/D5.6/, performing the necessary conversion.

## 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 3-17](#).

Local IBA

|     |     |     |                |                |                |                |                |             |     |
|-----|-----|-----|----------------|----------------|----------------|----------------|----------------|-------------|-----|
| SOP | LRH | BTH | Packet Payload | R <sub>0</sub> | R <sub>1</sub> | R <sub>2</sub> | R <sub>3</sub> | Variant CRC | EOP |
|-----|-----|-----|----------------|----------------|----------------|----------------|----------------|-------------|-----|

Global IBA

|     |     |     |     |                |                |                |                |                |             |     |
|-----|-----|-----|-----|----------------|----------------|----------------|----------------|----------------|-------------|-----|
| SOP | LRH | GRH | BTH | Packet Payload | R <sub>0</sub> | R <sub>1</sub> | R <sub>2</sub> | R <sub>3</sub> | Variant CRC | EOP |
|-----|-----|-----|-----|----------------|----------------|----------------|----------------|----------------|-------------|-----|

UG024\_14\_020802

Figure 3-17: 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 3-16](#).

Table 3-16: 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 3-18](#), along with the bits the CRC uses to evaluate the next packet.

|                |                                   |                |                |                |                |                |                |
|----------------|-----------------------------------|----------------|----------------|----------------|----------------|----------------|----------------|
| B <sub>0</sub> | B <sub>17</sub> – B <sub>10</sub> | B <sub>2</sub> | B <sub>3</sub> | B <sub>4</sub> | B <sub>5</sub> | B <sub>6</sub> | B <sub>7</sub> |
|----------------|-----------------------------------|----------------|----------------|----------------|----------------|----------------|----------------|

B<sub>17</sub>, B<sub>16</sub> 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 3-18: 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 97](#).

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 3-17](#).

**Table 3-17: 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 3-18](#) shows different settings for CHAN\_BONDONE\_SHOT and ENCHANSYNC in Master and Slave applications.

**Table 3-18: 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

Code examples can be downloaded from the *Virtex-II Pro Platform FPGA Handbook* page on the Xilinx website. Go to:

<http://www.xilinx.com/publications/products/v2pro/handbook/index.htm>

## Other Important Design Notes

### Receive Data Path 32-bit Alignment

The RocketIO transceiver uses the attribute `ALIGN_COMMA_MSB` to align protocol delimiters with the use of comma characters (special K characters K28.5, K28.1, and K28.7 for most protocols). Setting the `ALIGN_COMMA_MSB` to `TRUE`/`FALSE` determines where the comma characters appear on the `RXDATA` bus. A setting of `ALIGN_COMMA_MSB = FALSE` allows the comma to appear in any byte lane of the `RXDATA` in the 2- and 4-byte primitives. When `ALIGN_COMMA_MSB` is set to `TRUE`, the comma appears in `RXDATA[15:8]` for the 2-byte primitives, and in either `RXDATA[15:8]` or `RXDATA[31:24]` for the 4-byte primitives.

In the case of a 4-byte primitive, the transceiver sets comma alignment with respect to its 2-byte internal data path, but it does not constrain the comma to appear only in `RXDATA[31:24]`. Logic must be designed in the FPGA fabric to handle comma alignment for the 32-bit primitives when implementing certain protocols.

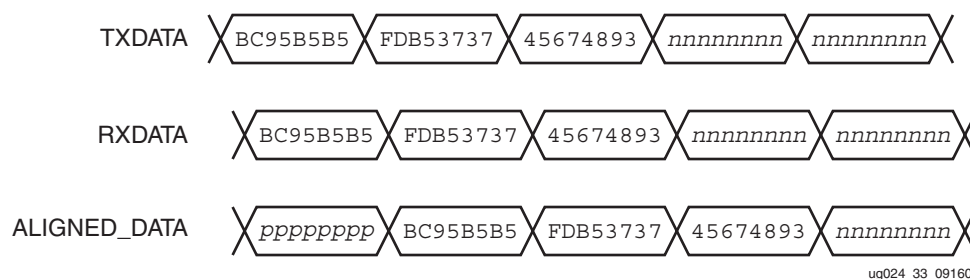
One such protocol is Fibre Channel. Delimiters such as IDLES, SOF, and EOF are four bytes long, and are assumed by the protocol logic to be aligned on a 32-bit boundary. The Fibre Channel IDLE delimiter is four bytes long and is composed of characters K28.5, D21.4, D21.5, and D21.5. The comma, K28.5, is transmitted in `TXDATA[31:24]`, which the protocol logic expects to be received in `RXDATA[31:24]`.

Using [Table 3-13, page 61](#), and [Table 3-14, page 69](#), the IDLE delimiter can be translated into a hexadecimal value `0xBC95B5B5` that represents the 32-bit `RXDATA` word. On the 32-bit `RXDATA` interface, the received word is either 32-bit aligned or misaligned, as shown in [Table 3-19](#). In the table, "*pp*" indicates a byte from a previous word of data.

**Table 3-19: 32-bit RXDATA, Aligned versus Misaligned**

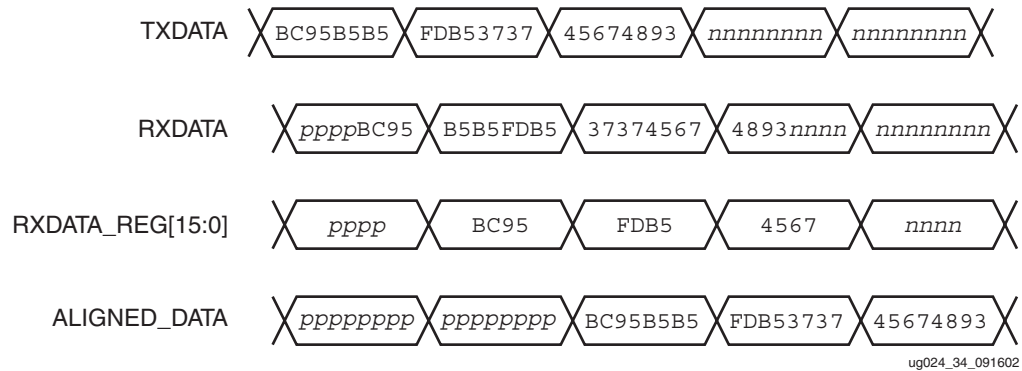
|                   | <b>RXDATA<br/>[31:24]</b> | <b>RXDATA<br/>[23:16]</b> | <b>RXDATA<br/>[15:8]</b> | <b>RXDATA<br/>[7:0]</b> |
|-------------------|---------------------------|---------------------------|--------------------------|-------------------------|
| 32-bit aligned    | BC                        | 95                        | B5                       | B5                      |
| CHARISCOMMA       | 1                         | 0                         | 0                        | 0                       |
| 32-bit misaligned | <i>pp</i>                 | <i>pp</i>                 | BC                       | 95                      |
| CHARISCOMMA       | 0                         | 0                         | 1                        | 0                       |

When `RXDATA` is 32-bit aligned, the logic should pass `RXDATA` though to the protocol logic without modification. A properly aligned data flow is shown in [Figure 3-19](#).



**Figure 3-19: RXDATA Aligned Correctly**

When RXDATA is 32-bit misaligned, the word requiring alignment is split between consecutive RXDATA words in the data stream, as shown in **Figure 3-20**.



**Figure 3-20: Realignment of RXDATA**

This conditional shift/delay operation on RXDATA also must be performed on the status outputs RXNOTINTABLE, RXDISPERR, RXCHARISK, RXCHARISCOMMA, and RXRUNDISP in order to keep them properly synchronized with RXDATA.

It is not possible to adjust RXCLKCORCNT appropriately for shifted/delayed RXDATA, because RXCLKCORCNT is summary data, and the summary for the shifted case cannot be recalculated.

### 32-bit Alignment Design

The following example code illustrates one way to create the logic to properly align 32-bit wide data with a comma in bits [31:24]. For brevity, most status bits are not included in this example design; however, these should be shifted in the same manner as RXDATA and RXCHARISK.

Note that when using a 40-bit data path (8B/10B bypassed), a similar realignment scheme may be used, but it cannot rely on RXCHARISCOMMA for comma detection.

#### Verilog

```

/*****
 *
 *   XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"
 *   AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND
 *   SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,
 *   OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,
 *   APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION
 *   THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,
 *   AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE
 *   FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY
 *   WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
 *   IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR
 *   REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF
 *   INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 *   FOR A PARTICULAR PURPOSE.
 *
 *   (c) Copyright 2002 Xilinx Inc.
 *   All rights reserved.
 *
 *****/

// Virtex-II Pro RocketIO comma alignment module
//

```

```

// This module reads RXDATA[31:0] from a RocketIO transceiver
// and copies it to
// its output, realigning it if necessary so that commas
// are aligned to the MSB position
// [31:24]. The module assumes ALIGN_COMMA_MSB is TRUE,
// so that the comma
// is already aligned to [31:24] or [15:8].
//
// Outputs
//
// aligned_data[31:0] -- Properly aligned 32-bit ALIGNED_DATA
// sync -- Indicator that aligned_data is properly aligned
// aligned_rxisk[3:0] - poperly aligned 4 bit RXCHARISK
// Inputs - These are all RocketIO inputs or outputs
// as indicated:
//
// usrclk2 -- RXUSRCLK2
// rxreset -- RXRESET
// rxisk[3:0] RXCHARISK[3:0]
// rxdata[31:0] RXDATA[31:0] -- (commas aligned to
//                                     [31:24] or [15:8])
// rxrealign -- RXREALIGN
// rxcommadet -- RXCOMMADET
// rxchariscomma3 -- RXCHARISCOMMA[3]
// rxchariscommal -- RXCHARISCOMMA[1]
//
module align_comma_32 ( aligned_data, aligned_rxisk, sync,
                        usrclk2, rxreset,
                        rxdata, rxisk,
                        rxrealign, rxcommadet,
                        rxchariscomma3, rxchariscommal );

    output [31:0] aligned_data;
    output [3:0] aligned_rxisk;
    output sync;

    reg [31:0] aligned_data;
    reg sync;

    input usrclk2;
    input rxreset;
    input [31:0] rxdata;
    input [3:0] rxisk;
    input rxrealign;
    input rxcommadet;
    input rxchariscomma3;
    input rxchariscommal;

    reg [15:0] rxdata_reg;
    reg [1:0] rxisk_reg;
    reg [3:0] aligned_rxisk;
    reg byte_sync;

    reg [3:0] wait_to_sync;
    reg count;

// This process maintains wait_to_sync and count,
// which are used only to
// maintain output sync; this provides some idea
// of when the output is properly
// aligned, with the comma in aligned_data[31:24]. The

```

```

// counter is set to a high value
// whenever the elastic buffer is reinitialized;
// that is, upon asserted RXRESET or
// RXREALIGN. Count-down is enabled whenever a
// comma is known to have
// come through the comma detection circuit,
// that is, upon an asserted RXREALIGN
// or RXCOMMADET.

always @ ( posedge usrclk2 )
begin
    if ( rxreset )
    begin
        wait_to_sync <= 4'b1111;
        count        <= 1'b0;
    end
    else if ( rxrealign )
    begin
        wait_to_sync <= 4'b1111;
        count        <= 1'b1;
    end
    else
    begin
        if ( count && ( wait_to_sync != 4'b0000 ) )
            wait_to_sync <= wait_to_sync - 4'b0001;
        if ( rxcommadet )
            count        <= 1'b1;
    end
end

// This process maintains output sync, which indicates
// when outgoing aligned_data
// should be properly aligned, with the comma in aligned_data[31:24].
// Output aligned_data is
// considered to be in sync when a comma is seen on
// rxdata (as indicated
// by rxchariscomma3 or 1) after the counter wait_to_sync
// has reached 0, indicating
// that commas seen by the comma detection circuit
// have had time to propagate to
// aligned_data after initialization of the elastic buffer.

always @ ( posedge usrclk2 )
begin
    if ( rxreset | rxrealign )
        sync <= 1'b0;
    else if ( ( wait_to_sync == 4'b0000 ) &
              ( rxchariscomma3 | rxchariscomma1 ) )
        sync <= 1'b1;
end

// This process generates aligned_data with commas aligned in [31:24],
// assuming that incoming commas are aligned to [31:24] or [15:8].
// Here, you could add code to use ENPCOMMAALIGN and
// ENMCOMMAALIGN to enable a move back into the byte_sync=0 state.

always @ ( posedge usrclk2 or posedge rxreset )
begin
    if ( rxreset )
    begin
        rxdata_reg <= 16'h0000;
        aligned_data <= 32'h0000_0000;
    end
end

```

```

        rxisk_reg  <= 2'b00;
        aligned_rxisk    <= 4'b0000;
        byte_sync  <= 1'b0;
    end
    else
    begin
        rxdata_reg[15:0] <= rxdata[15:0];
        rxisk_reg[1:0]   <= rxisk[1:0];
        if ( rxchariscomma3 )
        begin
            aligned_data[31:0] <= rxdata[31:0];
            aligned_rxisk[3:0]  <= rxisk[3:0];
            byte_sync    <= 1'b0;
        end
        else
        if ( rxchariscomma1 | byte_sync )
        begin
            aligned_data[31:0] <= { rxdata_reg[15:0], rxdata[31:16] };
            aligned_rxisk[3:0]  <= { rxisk_reg[1:0], rxisk[3:2] };
            byte_sync    <= 1'b1;
        end
        else
        begin
            aligned_data[31:0] <= rxdata[31:0];
            aligned_rxisk <= rxisk;
        end
    end
end
end

endmodule // align_comma_32

```

## VHDL

```

-- *
-- *****
-- *****
-- *
-- * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"
-- * AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND
-- * SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,
-- * OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,
-- * APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION
-- * THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,
-- * AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE
-- * FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY
-- * WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
-- * IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR
-- * REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF
-- * INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
-- * FOR A PARTICULAR PURPOSE.
-- *
-- * (c) Copyright 2002 Xilinx Inc.
-- * All rights reserved.
-- *
-- *****
-- Virtex-II Pro RocketIO comma alignment module
--
-- This module reads RXDATA[31:0] from a RocketIO transceiver
-- and copies it to
-- its output, realigning it if necessary so that commas

```

```

-- are aligned to the MSB position
-- [31:24]. The module assumes ALIGN_COMMA_MSB is TRUE,
-- so that the comma
-- is already aligned to [31:24] or [15:8].
--
-- Outputs
--
-- aligned_data[31:0] -- Properly aligned 32-std_logic ALIGNED_DATA
-- sync -- Indicator that aligned_data is properly aligned
-- aligned_rxisk[3:0] -properly aligned 4-std_logic RXCHARISK
-- Inputs - These are all RocketIO inputs or outputs
-- as indicated:
--
-- usrclk2 -- RXUSRCLK2
-- rxreset -- RXRESET
-- rxdata[31:0] RXDATA[31:0] -- (commas aligned to
--                               [31:24] or [15:8])
-- rxisk[3:0] - RXCHARISK[3:0]
-- rxrealign -- RXREALIGN
-- rxcommadet -- RXCOMMADET
-- rxchariscomma3 -- RXCHARISCOMMA[3]
-- rxchariscommal -- RXCHARISCOMMA[1]
--
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;

ENTITY align_comma_32 IS
  PORT (
    aligned_data      : OUT std_logic_vector(31 DOWNTO 0);
    aligned_rxisk     : OUT std_logic_vector(3 DOWNTO 0);
    sync              : OUT std_logic;
    usrclk2           : IN std_logic;
    rxreset           : IN std_logic;
    rxdata            : IN std_logic_vector(31 DOWNTO 0);
    rxisk             : IN std_logic_vector(3 DOWNTO 0);
    rxrealign         : IN std_logic;
    rxcommadet        : IN std_logic;
    rxchariscomma3    : IN std_logic;
    rxchariscommal    : IN std_logic);
END ENTITY align_comma_32;

ARCHITECTURE translated OF align_comma_32 IS

  SIGNAL rxdata_reg      : std_logic_vector(15 DOWNTO 0);
  SIGNAL rxisk_reg       : std_logic_vector(1 DOWNTO 0);
  SIGNAL byte_sync       : std_logic;
  SIGNAL wait_to_sync    : std_logic_vector(3 DOWNTO 0);
  SIGNAL count           : std_logic;
  SIGNAL rxdata_hold     : std_logic_vector(31 DOWNTO 0);
  SIGNAL rxisk_hold      : std_logic_vector(3 DOWNTO 0);
  SIGNAL sync_hold       : std_logic;

BEGIN
  aligned_data <= rxdata_hold;
  aligned_rxisk <= rxisk_hold;
  sync <= sync_hold;

  -- This process maintains wait_to_sync and count,

```



```
-- which are used only to
-- maintain output sync; this provides some idea
-- of when the output is properly
-- aligned, with the comma in aligned_data[31:24].
-- The counter is set to a high value
-- whenever the elastic buffer is reinitialized;
-- that is, upon asserted RXRESET or
-- RXREALIGN. Count-down is enabled whenever a
-- comma is known to have
-- come through the comma detection circuit, that
-- is, upon an asserted RXREALIGN
-- or RXCOMMADET.

PROCESS (usrclk2)
BEGIN
  IF (usrclk2'EVENT AND usrclk2 = '1') THEN
    IF (rxreset = '1') THEN
      wait_to_sync <= "1111";
      count <= '0';
    ELSE
      IF (rxrealign = '1') THEN
        wait_to_sync <= "1111";
        count <= '1';
      ELSE
        IF (count = '1') THEN
          IF (wait_to_sync /= "0000") THEN
            wait_to_sync <= wait_to_sync - "0001";
          END IF;
        END IF;
        IF (rxcommadet = '1') THEN
          count <= '1';
        END IF;
      END IF;
    END IF;
  END IF;
END PROCESS;

-- This process maintains output sync, which
-- indicates when outgoing aligned_data
-- should be properly aligned, with the comma
-- in aligned_data[31:24]. Output aligned_data is
-- considered to be in sync when a comma is seen
-- on rxdata (as indicated
-- by rxchariscomma3 or 1) after the counter
-- wait_to_sync has reached 0, indicating
-- that commas seen by the comma detection circuit
-- have had time to propagate to
-- aligned_data after initialization of the elastic buffer.
```

```
PROCESS (usrclk2)
BEGIN
  IF (usrclk2'EVENT AND usrclk2 = '1') THEN
    IF ((rxreset OR rxrealign) = '1') THEN
      sync_hold <= '0';
    ELSE
      IF (wait_to_sync = "0000") THEN
        IF ((rxchariscomma3 OR rxchariscomma1) = '1') THEN
          sync_hold <= '1';
        END IF;
      END IF;
    END IF;
  END IF;
END PROCESS;
```

```

END PROCESS;

-- This process generates aligned_data with commas
-- aligned in [31:24],
-- assuming that incoming commas are aligned
-- to [31:24] or [15:8].
-- Here, you could add code to use ENPCOMMAALIGN and
-- ENMCOMMAALIGN to enable a move back into the
-- byte_sync=0 state.

PROCESS (usrclk2, rxreset)
BEGIN
  IF (rxreset = '1') THEN
    rxdata_reg <= "0000000000000000";
    rxdata_hold <= "00000000000000000000000000000000";
    rxisk_reg <= "00";
    rxisk_hold <= "0000";
    byte_sync <= '0';
  ELSIF (usrclk2'EVENT AND usrclk2 = '1') THEN
    rxdata_reg(15 DOWNT0 0) <= rxdata(15 DOWNT0 0);
    rxisk_reg(1 DOWNT0 0) <= rxisk(1 DOWNT0 0);

    IF (rxchariscomma3 = '1') THEN
      rxdata_hold(31 DOWNT0 0) <= rxdata(31 DOWNT0 0);
      rxisk_hold(3 DOWNT0 0) <= rxisk(3 DOWNT0 0);
      byte_sync <= '0';
    ELSE
      IF ((rxchariscomma1 OR byte_sync) = '1') THEN
        rxdata_hold(31 DOWNT0 0) <= rxdata_reg(15 DOWNT0 0) &
          rxdata(31 DOWNT0 16);
        rxisk_hold(3 DOWNT0 0) <= rxisk_reg(1 DOWNT0 0) &
          rxisk(3 DOWNT0 2);

        byte_sync <= '1';
      ELSE
        rxdata_hold(31 DOWNT0 0) <= rxdata(31 DOWNT0 0);
        rxisk_hold(3 DOWNT0 0) <= rxisk(3 DOWNT0 0);
      END IF;
    END IF;
  END IF;
END PROCESS;

END ARCHITECTURE translated;

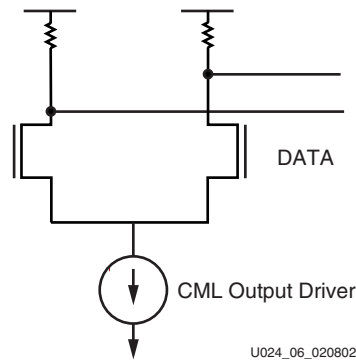
```

## Analog Design Considerations

### Serial I/O Description

The RocketIO 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](#).



**Figure 4-1: Differential Amplifier**

The RocketIO 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 84](#).

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).

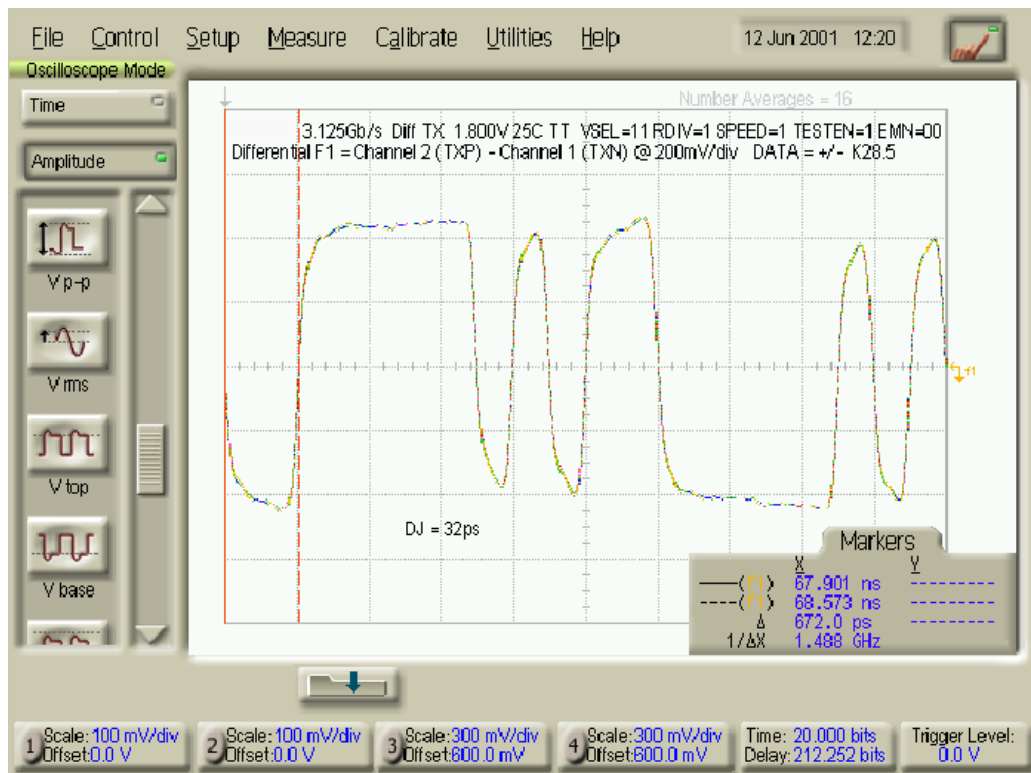
A second characteristic of RocketIO transceiver pre-emphasis is that the STRONG level is reduced after some time to the LOGIC level, thereby minimizing the voltage swing necessary to switch the differential pair into the opposite state.

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](#).

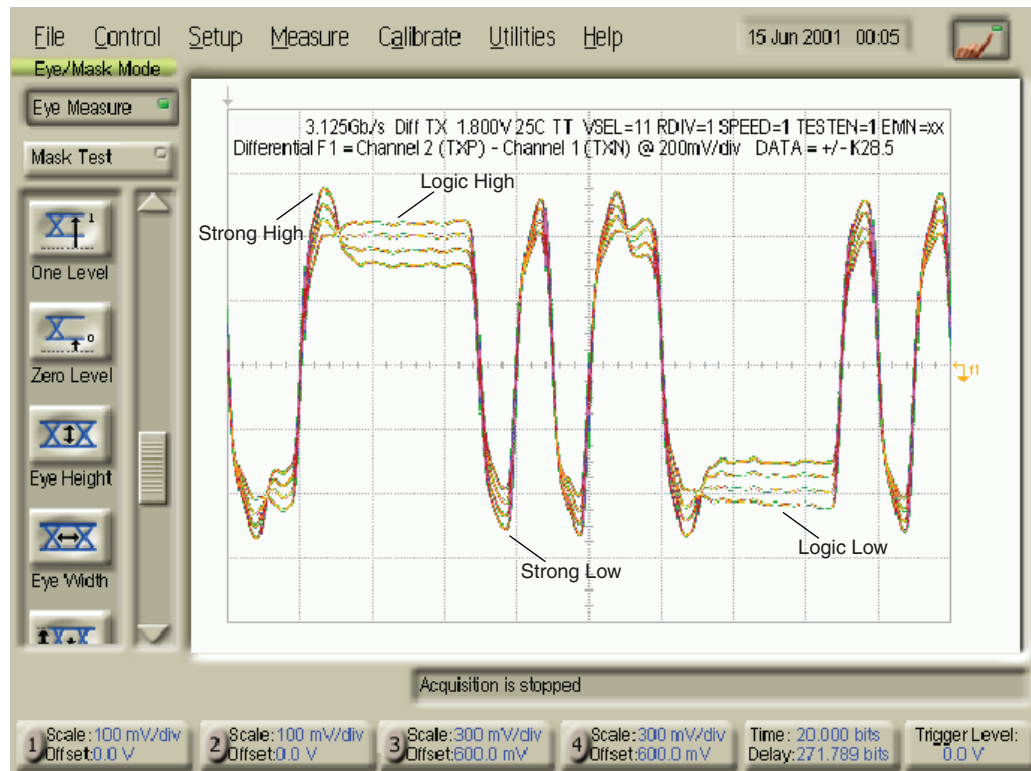
Table 4-2: Pre-emphasis Values

| Attribute Values | Emphasis (%) |
|------------------|--------------|
| 0                | 10           |
| 1                | 20           |
| 2                | 25           |
| 3                | 33           |



UG024\_17\_020802

Figure 4-2: Alternating K28.5+ with No Pre-Emphasis



UG024\_18\_020802

Figure 4-3: K28.5+ with Pre-Emphasis

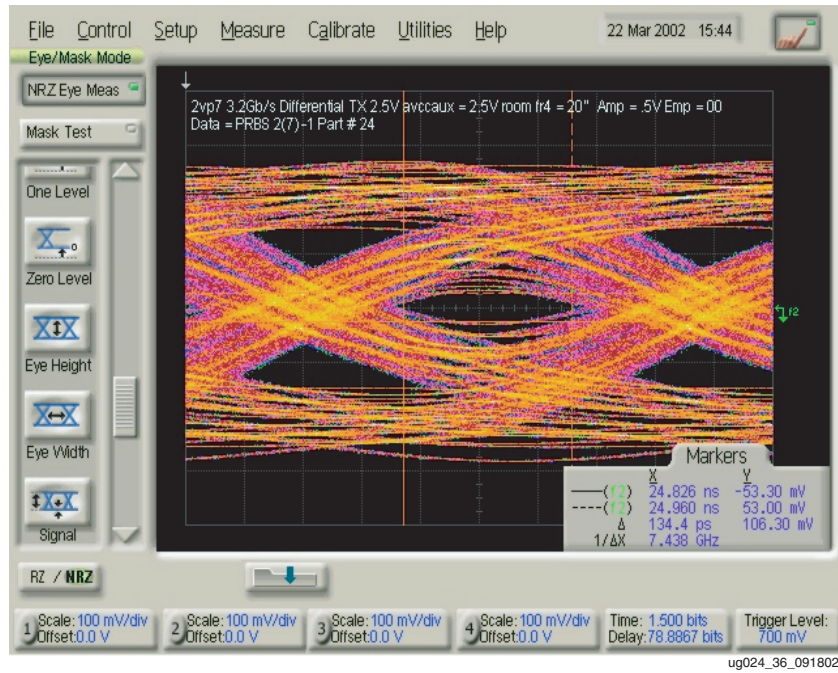


Figure 4-4: Eye Diagram, 10% Pre-Emphasis

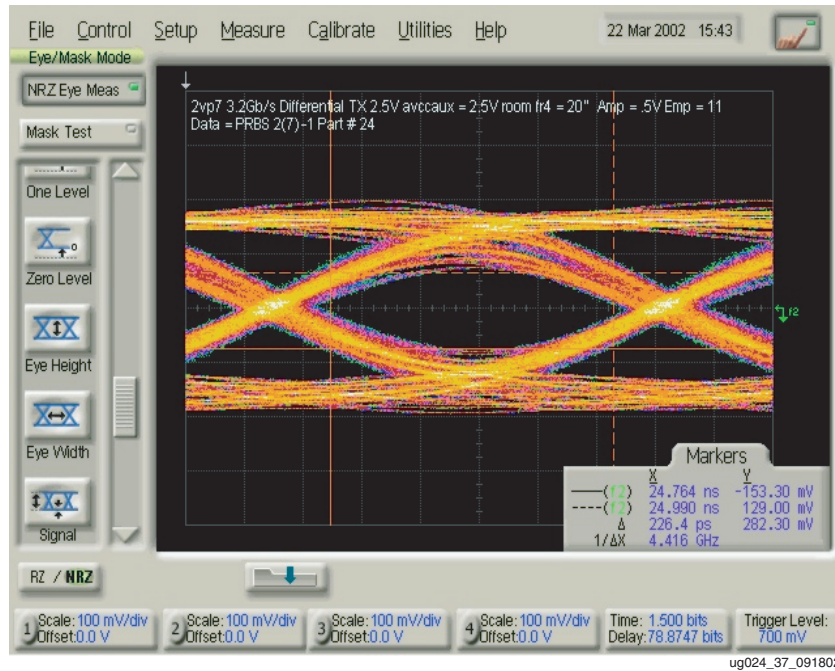


Figure 4-5: Eye Diagram, 33% Pre-Emphasis

## 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 50 $\Omega$  or 75 $\Omega$  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 <sup>(1)</sup> |            |
| $T_{DJTOL}$ | 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 RocketIO 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)                                | 311 |     | 1,562.5 | MHz             |                                    |
| Frequency Offset                     |  |     |     |         | ppm             |                                    |
| T <sub>DREF</sub>                    | REFCLK duty cycle  | 45  | 50  | 55      | %               |                                    |
| T <sub>RCLK</sub> /T <sub>FCLK</sub> | REFCLK rise and fall time (see Virtex-II Pro Data Sheet, Module 3) |     |     | 75      | ps              | Between 20% and 80% voltage levels |
| T <sub>GJT</sub>                     | REFCLK total jitter  |     |     | 40      | ps              | Peak-to-peak                       |
| T <sub>LOCK</sub>                    | Clock recovery frequency acquisition time                          |     | 10  |         | μs              |                                    |
| T <sub>UNLOCK</sub>                  |  |     |     |         | cycles          |                                    |
|                                      | PLL length   |     |     | 75      | non-transitions | Requirement when bypassing 8B/10B  |

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.

An additional feature of CDR is its ability to accept an external precision clock, REFCLK, which either acts to clock incoming data or to assist in synchronizing the derived RXRECCLK. 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 RocketIO 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 Conditioning

Each RocketIO transceiver has five power supply pins, all of which are sensitive to noise. [Table 4-5](#), summarizes the power supply pins, their names, associated voltages, and power requirements.

To operate properly, the RocketIO transceiver requires a certain level of noise isolation from surrounding noise sources. For this reason, it is required that both dedicated voltage regulators and passive high-frequency filtering be used to power the RocketIO circuitry.

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

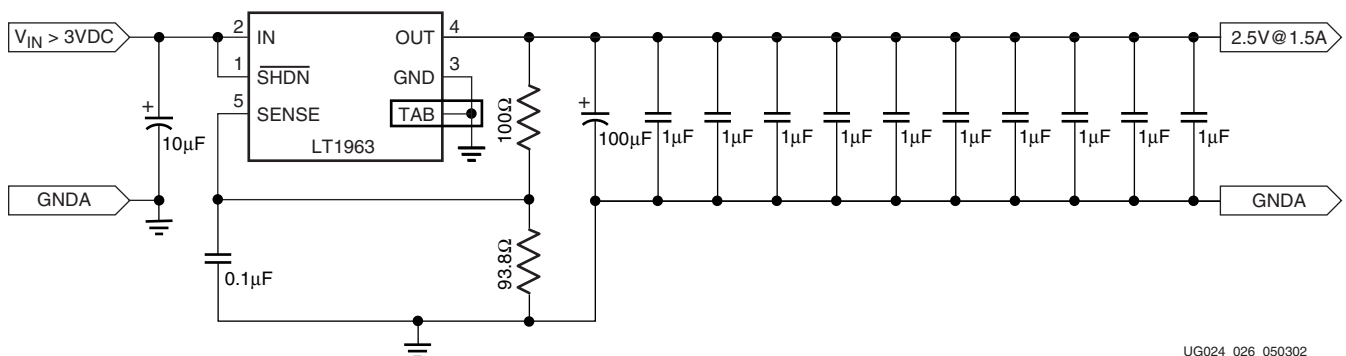
| Supply    | 2.5V | 1.8V - 2.625V | Power <sup>(1)</sup> (mW) | Description  |
|-----------|------|---------------|---------------------------|--|
| AVCCAUXRX | X    |               | 90                        | Analog RX supply                                       |
| AVCCAUXTX | X    |               | 130                       | Analog TX supply                                       |
| VTRX      |      | X             | 27 <sup>(2)</sup>         | RX termination supply                                  |
| VTTX      |      | X             | 27 <sup>(2)</sup>         | TX termination supply                                  |
| GNDA      |      |               | N/A                       | Analog ground for transmit and receive analog supplies |

**Notes:**

1. Power at max data rate. Power figures shown do not include power requirements of  $V_{CCINT}$  (28 mW) and  $V_{CCAUX}$  (48 mW), which power the PCS and PMA respectively.
2. Dependent on voltage level and whether AC or DC coupling is used. These numbers are based on AC coupling with 2.5V  $V_{TTX}$  and  $V_{TRX}$ .

### Voltage Regulation

The transceiver voltage regulator 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; however, each supply pin must still have its own separate passive filtering network.



**Figure 4-6: Power Supply Circuit Using LT1963 Regulator**

The required voltage regulator is the Linear Technology LT1963 device. This regulator must be used in the circuit specified by the manufacturer. [Figure 4-6](#) shows the schematic

for the adjustable version of the LT1963 device with values for a 2.5 V supply, as would be used for AVCCAUXRX and AVCCAUTX. Alternatively, fixed output voltage devices in the same series may be used, such as the LT1963-2.5. If the fixed version is used, SENSE should be connected to OUT.

Termination voltages  $V_{TTX}$  and  $V_{TRX}$  may be of any value in the range of 1.8 V to 2.625 V. In cases where the RocketIO transceiver is interfacing with a transceiver from another vendor, termination voltage may be dictated by the specifications of the other transceiver. In cases where the RocketIO transceiver is interfacing with another RocketIO transceiver, any termination voltage may be used. The logical choice is 2.5 V, as this voltage is already available on the board for the AVCCAUTX and AVCCAUXRX supplies.

The LT1963 circuit's output capacitors (100  $\mu$ F and 1  $\mu$ F) may be placed anywhere on the board, preferably close to the output of the LT1963 device.

Refer to the manufacturer's Web page at <http://www.linear-tech.com> for further information about this device.

### Passive Filtering

To achieve the necessary isolation from high-frequency power supply noise, passive filter networks are required on the power supply pins. The topology of these capacitor and ferrite bead circuits is given in Figure 4-7.

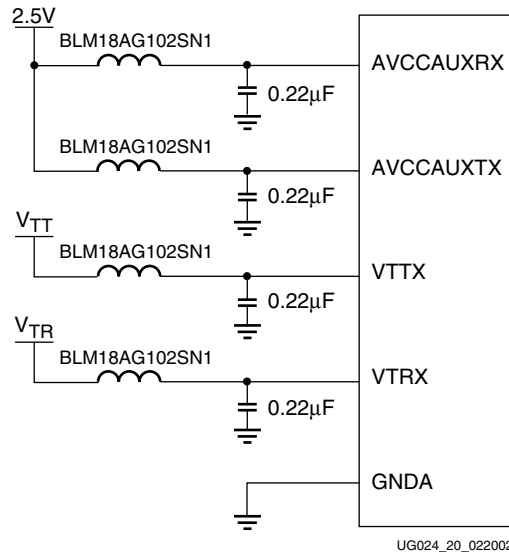
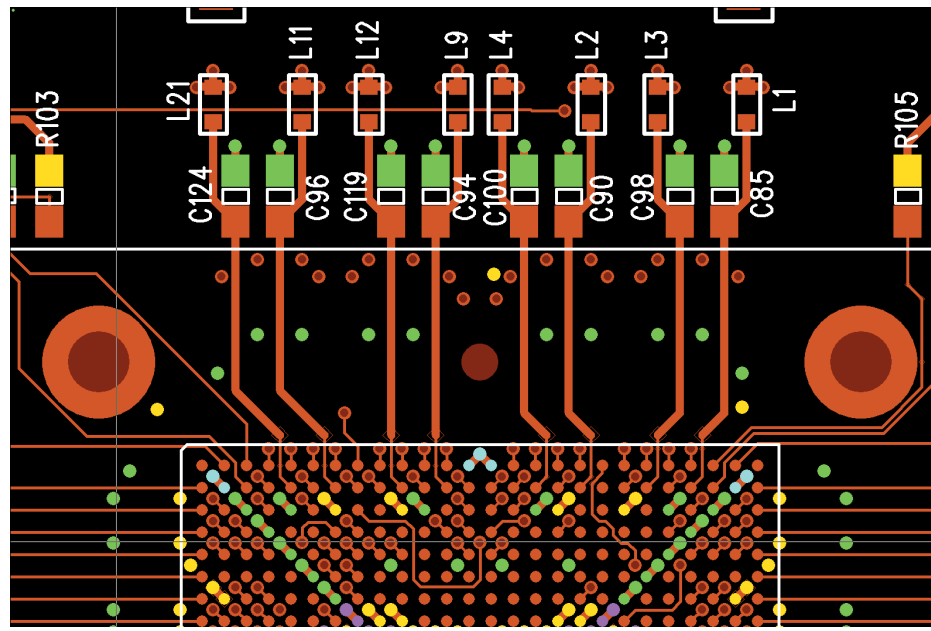


Figure 4-7: Power Filtering Network for One Transceiver

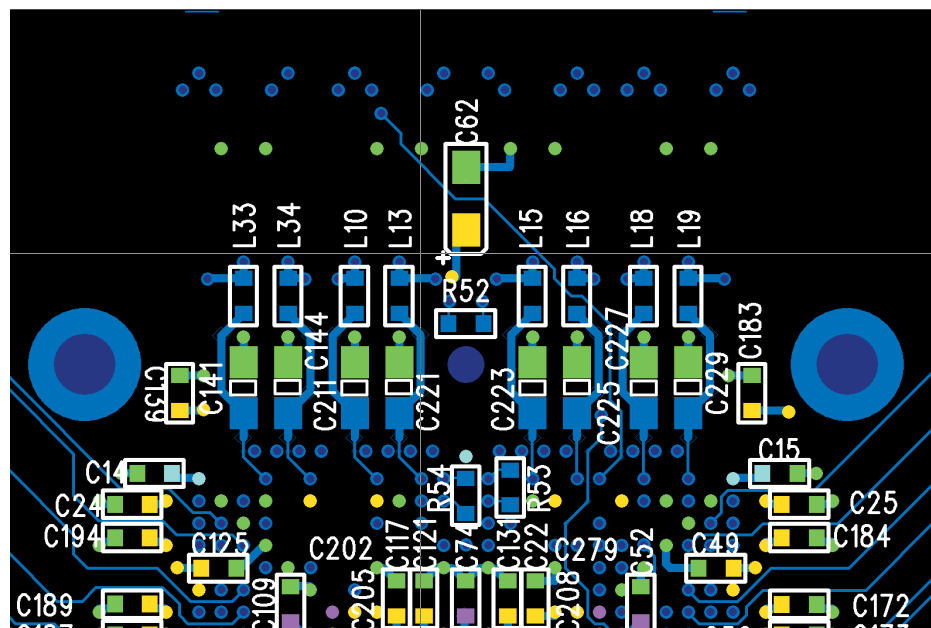
Each transceiver power pin requires one capacitor and one ferrite bead. The capacitors must be of value 0.22  $\mu$ F in an 0603 SMT package of X7R dielectric material at 10% tolerance, rated to at least 5 V. These capacitors must be placed within 1 cm of the pins they are connected to. The ferrite bead is the Murata BLM18AG102SN1. These components may not be shared under any circumstances.

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 AVCCAUTX 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.



UG024\_27\_022202

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

All AVCCAUXTX and AVCCAUXRX pins in a Virtex-II Pro device must be connected to 2.5 V, regardless of whether or not they are used. See [Powering the RocketIO Transceivers](#) and [The POWERDOWN Port](#), page 95, for details.

## High-Speed Serial Trace Design

### Routing Serial Traces

All RocketIO 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 RocketIO transceivers: 50Ω and 75Ω. Controlled

impedance traces with a corresponding impedance should be used to connect the RocketIO 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.

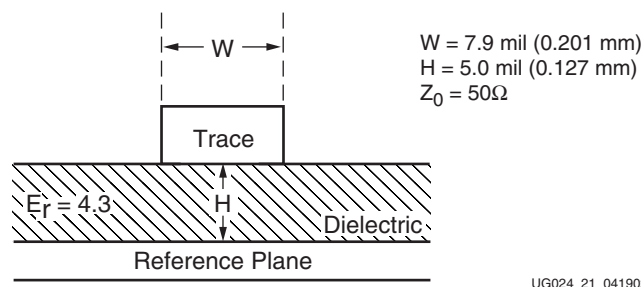
To control crosstalk, serial differential traces should be spaced at least five trace separation widths from all other PCB routes, including other serial pairs. A larger spacing is required if the other PCB routes carry especially noisy signals, such as TTL and other similarly noisy standards.

The RocketIO 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 RocketIO 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.

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



UG024\_21\_041902

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.

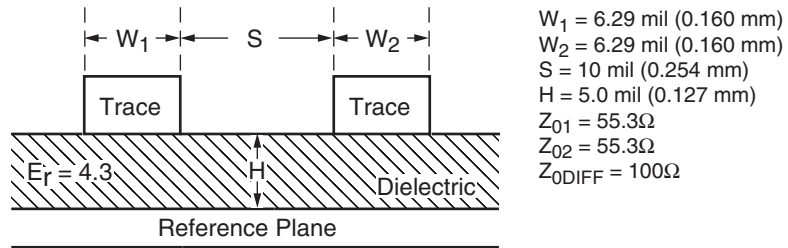


Figure 4-11: Microstrip Edge-Coupled Differential Pair

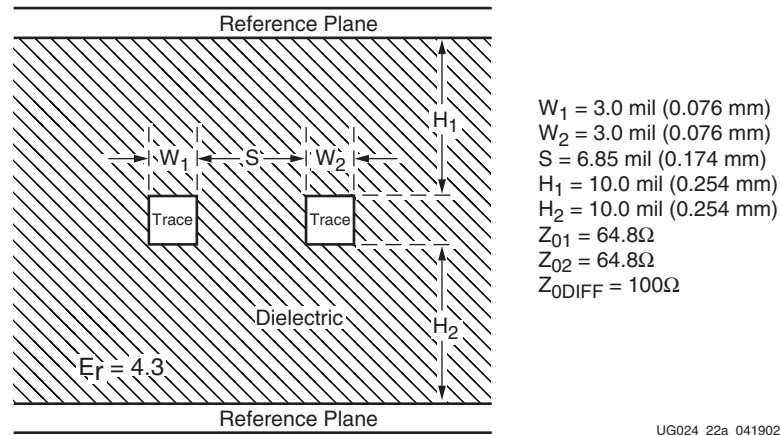


Figure 4-12: Stripline Edge-Coupled Differential Pair

### 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.

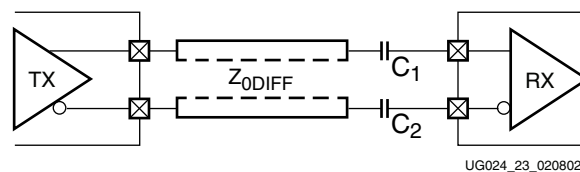


Figure 4-13: AC-Coupled Serial Link

Capacitors of value 0.01  $\mu$ F in a 0402 package are suitable for AC coupling at 3.125 Gb/s when 8B/10B encoding is used. Different data rates and different encoding schemes may require a different value.

DC coupling (direct connection) is preferable in cases where RocketIO transceivers are interfaced with other RocketIO 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](#).

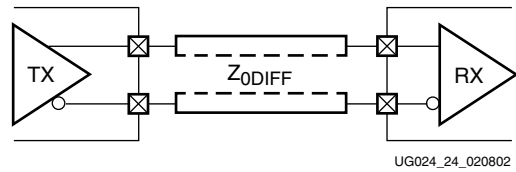


Figure 4-14: DC-Coupled Serial Link

## Reference Clock

A high degree of accuracy is required from the reference clock. For this reason, it is required that one of the oscillators listed in this section be used:

### Epson EG-2121CA 2.5V (LVPECL outputs)

See the [Epson Electronics America website](#) for detailed information. The power supply circuit specified by the manufacturer must be used.

The circuit shown in [Figure 4-15](#) must be used to interface the oscillator's LVPECL outputs to the LVDS inputs of the transceiver reference clock.

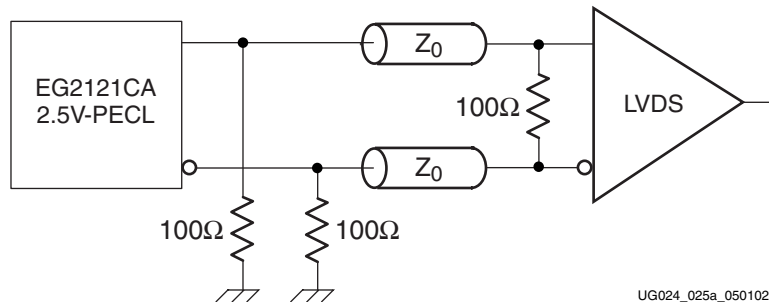


Figure 4-15: LVPECL Reference Clock Oscillator Interface

### Pletronics LV1145B (LVDS outputs)

See the [Pletronics website](#) for detailed information.

The circuit shown in [Figure 4-16](#) must be used to interface the oscillator's LVDS outputs to the LVDS inputs of the transceiver reference clock.

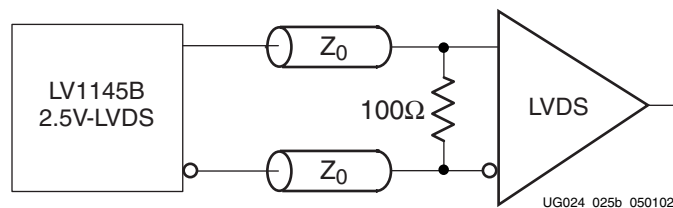


Figure 4-16: LVDS Reference Clock Oscillator Interface

## Other Important Design Notes

### Powering the RocketIO Transceivers

**IMPORTANT!** All RocketIO transceivers in the FPGA, whether instantiated in the design or not, must be connected to power and ground. Unused transceivers may be powered by any 2.5 V source, and passive filtering is not required. Refer to **Power Conditioning**, page 89, for detailed information on powering instantiated RocketIO transceivers.

The maximum power consumption per port is 350 mW at 3.125 Gb/s operation.

### The POWERDOWN Port

POWERDOWN is a single-bit primitive port (see **Table 3-1**, page 25) that 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 asserted, 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.

Any given transceiver that is *not* instantiated in the design will automatically be set to the POWERDOWN state by the Xilinx ISE development software, and will consume no power. An instantiated transceiver, however, will consume some power, even if it is not engaged in transmitting or receiving. Therefore, when a transceiver is not to be used for an extended period of time, the POWERDOWN port should be asserted High to reduce overall power consumption by the Virtex-II Pro FPGA.

Deasserting the POWERDOWN port restores the transceiver to normal functional status.





# Simulation and Implementation

---

## Simulation Models

### SmartModels

*SmartModels* are encrypted versions of the actual HDL code. These models allow the user to simulate the actual functionality of the design without having access to the code itself. A simulator with SmartModel capability is required to use SmartModels.

The models must be extracted before they can be used. For information on how to extract the SmartModels under ISE 5.1i, see [Solution Record 15501](#). For revisions of ISE *below* 5.1i, see [Solution Record 14019](#).

### HSPICE

*HSPICE* is an analog design model that allows simulation of the RX and TX high-speed transceiver. To obtain these HSPICE models, go to the SPICE Suite Access web page at <http://support.xilinx.com/support/software/spice/spice-request.htm>.

## 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 (optionally) 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.

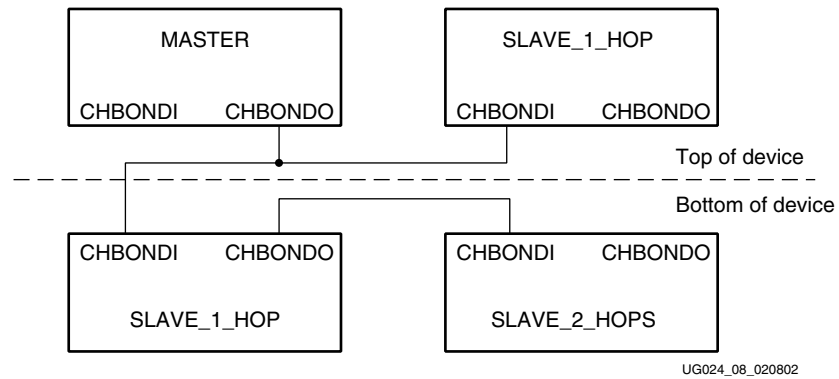


Figure 5-1: 2VP2 Implementation

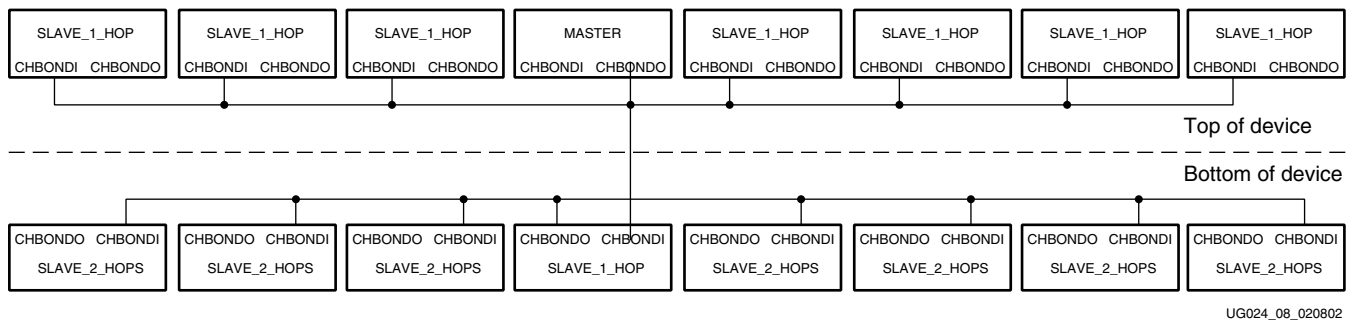


Figure 5-2: 2VP50 Implementation

## 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:

$$6.4 \text{ ns} - 0.2 \text{ ns} - 2.0 \text{ ns} = 4.2 \text{ ns}$$

This design used four RocketIO 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 RocketIO 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.

## MGT Package Pins

The MGTs are a hard core placed in the FPGA fabric; all package pins for the MGTs are dedicated on the Virtex-II Pro device. This is shown in the package pin diagrams in the Virtex-II Pro Handbook. When creating a design, LOC constraints must be used to implement a specific MGT on the die. This LOC constraint also determines which package pins are used. [Table 5-1](#) shows the correlation between the LOC grid and the package pins themselves. The pin numbers are the TXNPAD, TXPPAD, RXPPAD, and RXNPAD, respectively. The power pins are adjacent to these pins in the package pin diagrams of the handbook.

*Table 5-1: LOC Grid & Package Pins Correlation for FG256, FG456, and FF672*

| LOC Constraints | FG256              | FG456                  |                        | FF672                  |                        |
|-----------------|--------------------|------------------------|------------------------|------------------------|------------------------|
|                 | 2VP2/2VP4          | 2VP2/2VP4              | 2VP7                   | 2VP2/2VP4              | 2VP7                   |
| GT_X0_Y0        | T4, T5, T6, T7     | AB7, AB8, AB9, AB10    | AB3, AB4, AB5, AB6     | AF18, AF17, AF16, AF15 | AF23, AF22, AF21, AF20 |
| GT_X0_Y1        | T10, T11, T12, T13 | AB13, AB14, AB15, AB16 | A3, A4, A5, A6         | A18, A17, A16, A15     | A23, A22, A21, A20     |
| GT_X1_Y0        | A4, A5, A6, A7     | A7, A8, A9, A10        | AB7, AB8, AB9, AB10    | AF12, AF11, AF10, AF9  | AF18, AF17, AF16, AF15 |
| GT_X1_Y1        | A10, A11, A12, A13 | A13, A14, A15, A16     | A7, A8, A9, A10        | A12, A11, A10, A9      | A18, A17, A16, A15     |
| GT_X2_Y0        |                    |                        | AB13, AB14, AB15, AB16 |                        | AF12, AF11, AF10, AF9  |
| GT_X2_Y1        |                    |                        | A13, A14, A15, A16     |                        | A12, A11, A10, A9      |
| GT_X3_Y0        |                    |                        | AB17, AB18, AB19, AB20 |                        | AF7, AF6, AF5, AF4     |
| GT_X3_Y1        |                    |                        | A17, A18, A19, A20     |                        | A7, A6, A5, A4         |

*Table 5-2: LOC Grid & Package Pins Correlation for FF896 and FF1152*

| LOC Constraints | FF896                  |                        | FF1152                 |                        |                        |
|-----------------|------------------------|------------------------|------------------------|------------------------|------------------------|
|                 | 2VP7 /2VP20            | 2VP30                  | 2VP20 /2VP30           | 2VP40                  | 2VP50                  |
| GT_X0_Y0        | AK27, AK26, AK25, AK24 | AK27, AK26, AK25, AK24 | AP29, AP28, AP27, AP26 | AP33, AP32, AP31, AP30 | AP33, AP32, AP31, AP30 |
| GT_X0_Y1        | A27, A26, A25, A24     | A27, A26, A25, A24     | A29, A28, A27, A26     | A33, A32, A31, A30     | A33, A32, A31, A30     |
| GT_X1_Y0        | AK20, AK19, AK18, AK17 | AK20, AK19, AK18, AK17 | AP21, AP20, AP19, AP18 | AP29, AP28, AP27, AP26 | AP29, AP28, AP27, AP26 |
| GT_X1_Y1        | A20, A19, A18, A17     | A20, A19, A18, A17     | A21, A20, A19, A18     | A29, A28, A27, A26     | A29, A28, A27, A26     |
| GT_X2_Y0        | AK14, AK13, AK12, AK11 | AK14, AK13, AK12, AK11 | AP17, AP16, AP15, AP14 | AP21, AP20, AP19, AP18 | AP25, AP24, AP23, AP22 |

Table 5-2: LOC Grid &amp; Package Pins Correlation for FF896 and FF1152 (Continued)

| LOC Constraints | FF896              |                    | FF1152                 |                        |                        |
|-----------------|--------------------|--------------------|------------------------|------------------------|------------------------|
|                 | 2VP7 /2VP20        | 2VP30              | 2VP20 /2VP30           | 2VP40                  | 2VP50                  |
| GT_X2_Y1        | A14, A13, A12, A11 | A14, A13, A12, A11 | A17, A16, A15, A14     | A21, A20, A19, A18     | A25, A24, A23, A22     |
| GT_X3_Y0        | AK7, AK6, AK5, AK4 | AK7, AK6, AK5, AK4 | AP13, AP12, AP11, AP10 | AP17, AP16, AP15, AP14 | AP21, AP20, AP19, AP18 |
| GT_X3_Y1        | A7, A6, A5, A4     | A7, A6, A5, A4     | A13, A12, A11, A10     | A17, A16, A15, A14     | A21, A20, A19, A18     |
| GT_X4_Y0        |                    |                    |                        | AP9, AP8, AP7, AP6     | AP17, AP16, AP15, AP14 |
| GT_X4_Y1        |                    |                    |                        | A9, A8, A7, A6         | A17, A16, A15, A14     |
| GT_X5_Y0        |                    |                    |                        | AP5, AP4, AP3, AP2     | AP13, AP12, AP11, AP10 |
| GT_X5_Y1        |                    |                    |                        | A5, A4, A3, A2         | A13, A12, A11, A10     |
| GT_X6_Y0        |                    |                    |                        |                        | AP9, AP8, AP7, AP6     |
| GT_X6_Y1        |                    |                    |                        |                        | A9, A8, A7, A6         |
| GT_X7_Y0        |                    |                    |                        |                        | AP5, AP4, AP3, AP2     |
| GT_X7_Y1        |                    |                    |                        |                        | A5, A4, A3, A2         |
| GT_X8_Y0        |                    |                    |                        |                        |                        |
| GT_X8_Y1        |                    |                    |                        |                        |                        |
| GT_X9_Y0        |                    |                    |                        |                        |                        |
| GT_X9_Y1        |                    |                    |                        |                        |                        |

Table 5-3: LOC Grid &amp; Package Pins Correlation for FF1517 and FF1704

| LOC Constraints | FF1517                 |                        |                        | FF1704                 |                        |
|-----------------|------------------------|------------------------|------------------------|------------------------|------------------------|
|                 | 2VP40                  | 2VP50                  | 2VP70                  | 2VP70 /2VP100          | 2VP125                 |
| GT_X0_Y0        | AW36, AW35, AW34, AW33 | AW36, AW35, AW34, AW33 | AW36, AW35, AW34, AW33 | BB41, BB40, BB39, BB38 |                        |
| GT_X0_Y1        | A36, A35, A34, A33     | A36, A35, A34, A33     | A36, A35, A34, A33     | A41, A40, A39, A38     |                        |
| GT_X1_Y0        | AW32, AW31, AW30, AW29 | AW32, AW31, AW30, AW29 | AW32, AW31, AW30, AW29 | BB37, BB36, BB35, BB34 | BB41, BB40, BB39, BB38 |
| GT_X1_Y1        | A32, A31, A30, A29     | A32, A31, A30, A29     | A32, A31, A30, A29     | A37, A36, A35, A34     | A41, A40, A39, A38     |

Table 5-3: LOC Grid &amp; Package Pins Correlation for FF1517 and FF1704 (Continued)

| LOC Constraints | FF1517                 |                        |                        | FF1704                 |                        |
|-----------------|------------------------|------------------------|------------------------|------------------------|------------------------|
|                 | 2VP40                  | 2VP50                  | 2VP70                  | 2VP70 /2VP100          | 2VP125                 |
| GT_X2_Y0        | AW24, AW23, AW22, AW21 | AW28, AW27, AW26, AW25 | AW32, AW31, AW30, AW29 | BB33, BB32, BB31, BB30 | BB37, BB36, BB35, BB34 |
| GT_X2_Y1        | A24, A23, A22, A21     | A28, A27, A26, A25     | A32, A31, A30, A29     | A33, A32, A31, A30     | A37, A36, A35, A34     |
| GT_X3_Y0        | AW19, AW18, AW17, AW16 | AW24, AW23, AW22, AW21 | AW28, AW27, AW26, AW25 | BB29, BB28, BB27, BB26 | BB33, BB32, BB31, BB30 |
| GT_X3_Y1        | A19, A18, A17, A16     | A24, A23, A22, A21     | A28, A27, A26, A25     | A29, A28, A27, A26     | A33, A32, A31, A30     |
| GT_X4_Y0        | AW11, AW10, AW9, AW8   | AW19, AW18, AW17, AW16 | AW24, AW23, AW22, AW21 | BB25, BB24, BB23, BB22 | BB29, BB28, BB27, BB26 |
| GT_X4_Y1        | A11, A10, A9, A8       | A19, A18, A17, A16     | A24, A23, A22, A21     | A25, A24, A23, A22     | A29, A28, A27, A26     |
| GT_X5_Y0        | AW7, AW6, AW5, AW4     | AW15, AW14, AW13, AW12 | AW19, AW18, AW17, AW16 | BB21, BB20, BB19, BB18 | BB25, BB24, BB23, BB22 |
| GT_X5_Y1        | A7, A6, A5, A4         | A15, A14, A13, A12     | A19, A18, A17, A16     | A21, A20, A19, A18     | A25, A24, A23, A22     |
| GT_X6_Y0        |                        | AW11, AW10, AW9, AW8   | AW15, AW14, AW13, AW12 | BB17, BB16, BB15, BB14 | BB21, BB20, BB19, BB18 |
| GT_X6_Y1        |                        | A11, A10, A9, A8       | A15, A14, A13, A12     | A17, A16, A15, A14     | A21, A20, A19, A18     |
| GT_X7_Y0        |                        | AW7, AW6, AW5, AW4     | AW11, AW10, AW9, AW8   | BB13, BB12, BB11, BB10 | BB17, BB16, BB15, BB14 |
| GT_X7_Y1        |                        | A7, A6, A5, A4         | A11, A10, A9, A8       | A13, A12, A11, A10     | A17, A16, A15, A14     |
| GT_X8_Y0        |                        |                        |                        | BB9, BB8, BB7, BB6     | BB13, BB12, BB11, BB10 |
| GT_X8_Y1        |                        |                        |                        | A9, A8, A7, A6         | A13, A12, A11, A10     |
| GT_X9_Y0        |                        |                        | AW7, AW6, AW5, AW4     | BB5, BB4, BB3, BB2     | BB9, BB8, BB7, BB6     |
| GT_X9_Y1        |                        |                        | A7, A6, A5, A4         | A5, A4, A3, A2         | A9, A8, A7, A6         |
| GT_X10_Y0       |                        |                        |                        |                        | BB5, BB4, BB3, BB2     |
| GT_X10_Y1       |                        |                        |                        |                        | A5, A4, A3, A2         |
| GT_X11_Y0       |                        |                        |                        |                        |                        |
| GT_X11_Y1       |                        |                        |                        |                        |                        |

## Diagnostic Signals

Often a diagnostic check is needed upon power-up. RocketIO 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 5-4](#) shows the three modes for loopback.

Table 5-4: 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.                    |

## RocketIO Transceiver Timing Model

---

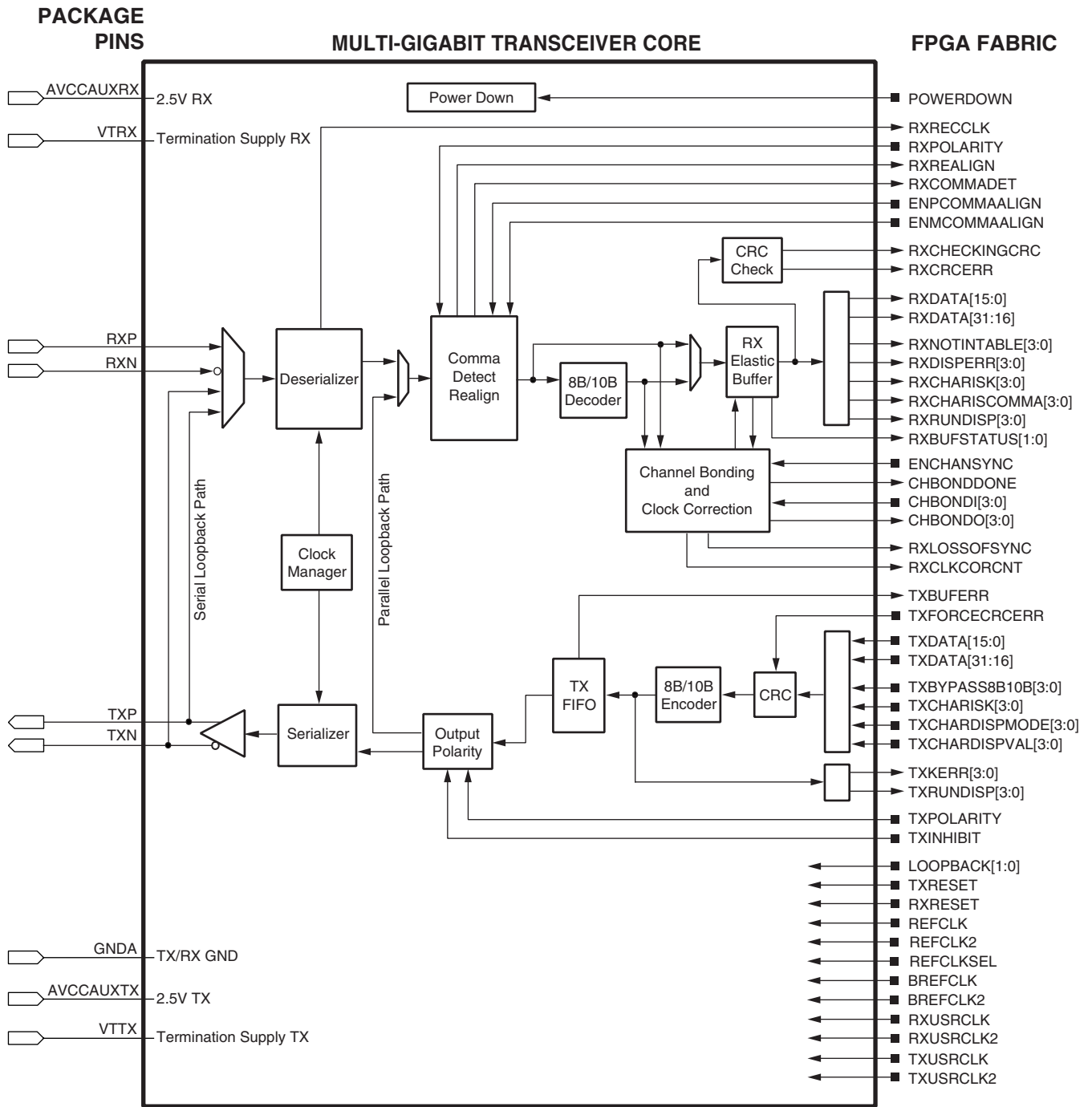
This appendix explains all of the timing parameters associated with the RocketIO™ transceiver core. It is intended to be used in conjunction with Module 3 of the [Virtex-II Pro Data Sheet](#) and the Timing Analyzer (TRCE) report from Xilinx software. For specific timing parameter values, refer to the data sheet.

There are many signals entering and exiting the RocketIO core. (Refer to [Figure A-1](#).) The model presented in this section treats the RocketIO core as a “black box.” Propagation delays internal to the RocketIO core logic are ignored. Signals are characterized with setup and hold times for inputs, and with clock to valid output times for outputs.

There are five clocks associated with the RocketIO core, but only three of these clocks—RXUSRCLK, RXUSRCLK2, and TXUSRCLK2—have I/Os that are synchronous to them. The following table gives a brief description of all of these clocks. For an in-depth discussion of clocking the RocketIO core, refer to [Chapter 3, Digital Design Considerations](#).

*Table A-1: RocketIO Clock Descriptions*

| CLOCK SIGNAL | DESCRIPTION   |
|--------------|---|
| REFCLK       | Main reference clock for RocketIO transceiver   |
| TXUSRCLK     | Clock used for writing the TX buffer. Frequency-locked to REFCLK.   |
| TXUSRCLK2    | Clocks transmission data and status and reconfiguration data between the transceiver and the FPGA core. Relationship between TXUSRCLK2 and TXUSRCLK depends on width of transmission data path. |
| RXUSRCLK     | Clock used for reading the RX elastic buffer. Clocks CHBONDI and CHBONO into and out of the transceiver. Typically the same as TXUSRCLK.  |
| RXUSRCLK2    | Clocks receiver data and status between the transceiver and the FPGA core. Typically the same as TXUSRCLK2. Relationship between RXUSRCLK2 and RXUSRCLK depends on width of receiver data path. |



DS083-2\_04\_090402

Figure A-1: RocketIO Transceiver Block Diagram

## Timing Parameters

Parameter designations are constructed to reflect the functions they perform, as well as the I/O signals to which they are synchronous. The following subsections explain the meaning of each of the basic timing parameter designations used in the tables.



## Setup/Hold Times of Inputs Relative to Clock

Basic Format:

*ParameterName\_SIGNAL*

where

*ParameterName* = T with subscript string defining the timing relationship  
*SIGNAL* = name of RocketIO signal synchronous to the clock

ParameterName Format:

$T_{GxCK}$  = Setup time before clock edge  
 $T_{GCKx}$  = Hold time after clock edge

where

$x$  = C (Control inputs)  
 D (Data inputs)

Setup/Hold Time (Examples):

$T_{GCKC\_RRST}/T_{GCKC\_PLB}$  Setup/hold times of RX Reset input relative to rising edge of RXUSRCLK2  
 $T_{GDCK\_TDAT}/T_{GCKD\_TDAT}$  Setup/hold times of TX Data inputs relative to rising edge of TXUSRCLK2

## Clock to Output Delays

Basic Format:

*ParameterName\_SIGNAL*

where

*ParameterName* = T with subscript string defining the timing relationship  
*SIGNAL* = name of RocketIO signal synchronous to the clock

ParameterName Format:

$T_{GCKx}$  = Delay time from clock edge to output

where

$x$  = CO (Control outputs)  
 DO (Data outputs)  
 ST (Status outputs)

Output Delay Time (Examples):

$T_{GCKCO\_CHBO}$  Rising edge of RXUSRCLK to Channel Bond outputs  
 $T_{GCKDO\_RDAT}$  Rising edge of RXUSRCLK2 to RX Data outputs  
 $T_{GCKST\_TBERR}$  Rising edge of TXUSRCLK2 to TX Buffer Err output

## Clock Pulse Width

ParameterName Format:

$T_{xPWH}$  = Minimum pulse width, High state  
 $T_{xPWL}$  = Minimum pulse width, Low state

where

$x$  = REF (REFCLK)  
 TX (TXUSRCLK)  
 TX2 (TXUSRCLK2)  
 RX (RXUSRCLK)  
 RX2 (RXUSRCLK2)

Pulse Width (Examples):

$T_{TX2PWL}$  Minimum pulse width, TX2 clock, Low state  
 $T_{REFPWH}$  Minimum pulse width, Reference clock, High state

## Timing Parameter Tables and Diagram

The following four tables list the timing parameters as reported by the implementation tools relative to the clocks given in [Table A-1](#), along with the RocketIO signals that are synchronous to each clock. (No signals are synchronous to REFCLK or TXUSRCLK.)

A timing diagram ([Figure A-2](#)) illustrates the timing relationships.

- [Table A-2, Parameters Relative to the RX User Clock \(RXUSRCLK\)](#), page 106
- [Table A-3, Parameters Relative to the RX User Clock2 \(RXUSRCLK2\)](#), page 106
- [Table A-4, Parameters Relative to the TX User Clock2 \(TXUSRCLK2\)](#), page 107
- [Table A-5, Miscellaneous Clock Parameters](#), page 107

**Table A-2: Parameters Relative to the RX User Clock (RXUSRCLK)**

| Parameter                                       | Function                      | Signals      |
|---|-------------------------------|--------------|
| <b>Setup/Hold:</b>                              |                               |              |
| T <sub>GCKC</sub> _CHBI/T <sub>GCKC</sub> _CHBI | Control inputs                | CHBONDI[3:0] |
| <b>Clock to Out:</b>                            |                               |              |
| T <sub>GCKCO</sub> _CHBO                        | Control outputs               | CHBONDO[3:0] |
| <b>Clock:</b>                                   |                               |              |
| T <sub>RXPWH</sub>                              | Clock pulse width, High state | RXUSRCLK     |
| T <sub>RXPWL</sub>                              | Clock pulse width, Low state  | RXUSRCLK     |

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

| Parameter                                       | Function                      | Signals            |
|---|-------------------------------|--------------------|
| <b>Setup/Hold:</b>                              |                               |                    |
| T <sub>GCKC</sub> _RRST/T <sub>GCKC</sub> _RRST | Control input                 | RXRESET            |
| T <sub>GCKC</sub> _RPOL/T <sub>GCKC</sub> _RPOL | Control input                 | RXPOLARITY         |
| T <sub>GCKC</sub> _ECSY/T <sub>GCKC</sub> _ECSY | Control input                 | ENCHANSYNC         |
| <b>Clock to Out:</b>                            |                               |                    |
| T <sub>GCKST</sub> _RNIT                        | Status outputs                | RXNOTINTABLE[3:0]  |
| T <sub>GCKST</sub> _RDERR                       | Status outputs                | RXDISPERR[3:0]     |
| T <sub>GCKST</sub> _RCMCH                       | Status outputs                | RXCHARISCOMMA[3:0] |
| T <sub>GCKST</sub> _ALIGN                       | Status output                 | RXREALIGN          |
| T <sub>GCKST</sub> _CMDT                        | Status output                 | RXCOMMADET         |
| T <sub>GCKST</sub> _RLOS                        | Status outputs                | RXLOSSOFSYNC[1:0]  |
| T <sub>GCKST</sub> _RCCCNT                      | Status outputs                | RXCLKCORCNT[2:0]   |
| T <sub>GCKST</sub> _RBSTA                       | Status outputs                | RXBUFSTATUS[1:0]   |
| T <sub>GCKST</sub> _RCCRC                       | Status output                 | RXCHECKINGCRC      |
| T <sub>GCKST</sub> _RCRCE                       | Status output                 | RXCRCERR           |
| T <sub>GCKST</sub> _CHBD                        | Status output                 | CHBONDDONE         |
| T <sub>GCKST</sub> _RKCH                        | Status outputs                | RXCHARISK[3:0]     |
| T <sub>GCKST</sub> _RRDIS                       | Status outputs                | RXRUNDISP[3:0]     |
| T <sub>GCKDO</sub> _RDAT                        | Data outputs                  | RXDATA[31:0]       |
| <b>Clock:</b>                                   |                               |                    |
| T <sub>RX2PWH</sub>                             | Clock pulse width, High state | RXUSRCLK2          |
| T <sub>RX2PWL</sub>                             | Clock pulse width, Low state  | RXUSRCLK2          |

Table A-4: Parameters Relative to the TX User Clock2 (TXUSRCLK2)

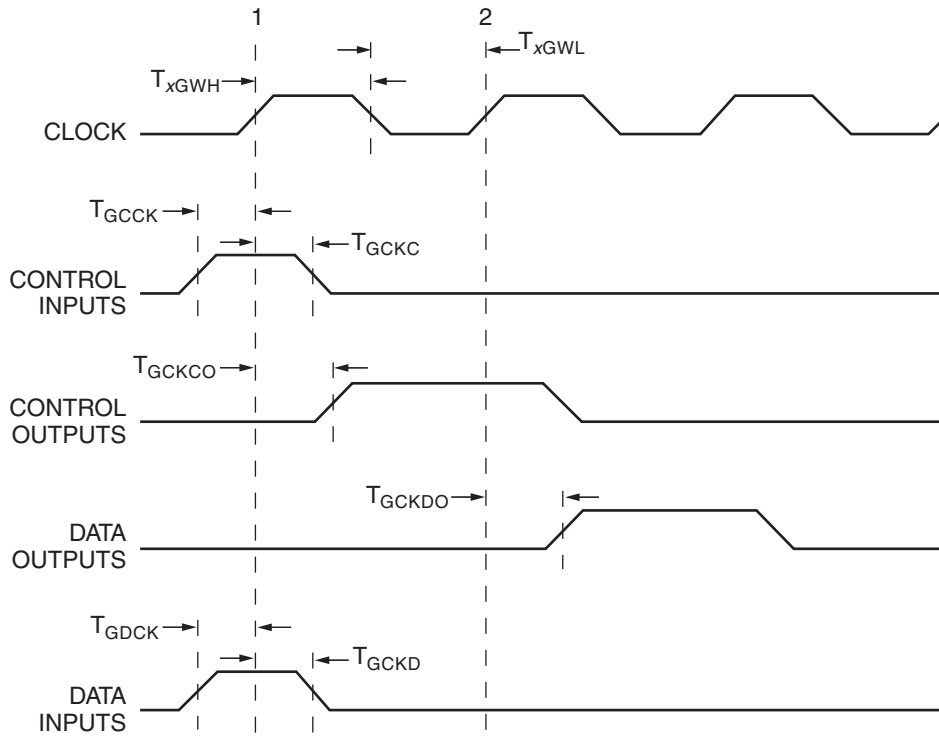
| Parameter                         | Function                      | Signals             |
|-----------------------------------|-------------------------------|---------------------|
| <b>Setup/Hold:</b>                |                               |                     |
| $T_{GCKC\_CFGEN}/T_{GCKC\_CFGEN}$ | Control inputs                | CONFIGENABLE        |
| $T_{GCKC\_TBYP}/T_{GCKC\_TBYP}$   | Control inputs                | TXBYPASS8B10B[3:0]  |
| $T_{GCKC\_TCRCE}/T_{GCKC\_TCRCE}$ | Control inputs                | TXFORCECERCERR      |
| $T_{GCKC\_TPOL}/T_{GCKC\_TPOL}$   | Control inputs                | TXPOLARITY          |
| $T_{GCKC\_TINH}/T_{GCKC\_TINH}$   | Control inputs                | TXINHIBIT           |
| $T_{GCKC\_LBK}/T_{GCKC\_LBK}$     | Control inputs                | LOOPBACK[1:0]       |
| $T_{GCKC\_TRST}/T_{GCKC\_TRST}$   | Control inputs                | TXRESET             |
| $T_{GCKC\_TKCH}/T_{GCKC\_TKCH}$   | Control inputs                | TXCHARISK[3:0]      |
| $T_{GCKC\_TCDM}/T_{GCKC\_TCDM}$   | Control inputs                | TXCHARDISPMODE[3:0] |
| $T_{GCKC\_TCDV}/T_{GCKC\_TCDV}$   | Control inputs                | TXCHARDISPVAL[3:0]  |
| $T_{GDCK\_CFGIN}/T_{GDCK\_CFGIN}$ | Data inputs                   | CONFIGIN            |
| $T_{GDCK\_TDAT}/T_{GDCK\_TDAT}$   | Data inputs                   | TXDATA[31:0]        |
| <b>Clock to Out:</b>              |                               |                     |
| $T_{GCKST\_TBERR}$                | Status outputs                | TXBUFERR            |
| $T_{GCKST\_TKERR}$                | Status outputs                | TXKERR[3:0]         |
| $T_{GCKDO\_TRDIS}$                | Data outputs                  | TXRUNDISP[3:0]      |
| $T_{GCKDO\_CFGOUT}$               | Data outputs                  | CONFIGOUT           |
| <b>Clock:</b>                     |                               |                     |
| $T_{TX2PWH}$                      | Clock pulse width, High state | TXUSRCLK2           |
| $T_{TX2PWH}$                      | Clock pulse width, Low state  | TXUSRCLK2           |

Table A-5: Miscellaneous Clock Parameters

| Parameter     | Function                      | Signals                 |
|---------------|-------------------------------|-------------------------|
| <b>Clock:</b> |                               |                         |
| $T_{REFPWH}$  | Clock pulse width, High state | REFCLK <sup>(1)</sup>   |
| $T_{REFPWL}$  | Clock pulse width, Low state  | REFCLK <sup>(1)</sup>   |
| $T_{TXPWH}$   | Clock pulse width, High state | TXUSRCLK <sup>(2)</sup> |
| $T_{TXPWL}$   | Clock pulse width, Low state  | TXUSRCLK <sup>(2)</sup> |

**Notes:**

- REFCLK is not synchronous to any RocketIO signals.
- TXUSRCLK is not synchronous to any RocketIO signals.



UG012\_106\_02\_100101

Figure A-2: RocketIO Transceiver Timing Relative to Clock Edge

# *RocketIO Transceiver Cell Models*

---

## Summary

This appendix documents the RocketIO™ Multi-Gigabit Transceiver cell models. The following information lists the Verilog module declarations of the model and pins associated with each of the RocketIO 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,  
    BREFCLK,
```

```

BREFCLK2 ,
RXN ,
RXP ,
RXPOLARITY ,
RXRESET ,
RXUSRCLK ,
RXUSRCLK2 ,
TXBYPASS8B10B ,
TXCHARDISPMODE ,
TXCHARDISPVAL ,
TXCHARISK ,
TXDATA ,
TXFORCECERR ,
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 ,
    BREFCLK ,
    BREFCLK2 ,
    RXN ,
    RXP ,
    RXPOLARITY ,
    RXRESET ,

```

```

RXUSRCLK,
RXUSRCLK2,
TXBPASS8B10B,
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,
    BREFCLK,
    BREFCLK2,
    RXN,
    RXP,
    RXPOLARITY,
    RXRESET,
    RXUSRCLK,
    RXUSRCLK2,
    TXBPASS8B10B,
    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,
  BREFCLK,
  BREFCLK2,
  RXN,
  RXP,
  RXPOLARITY,
  RXRESET,
  RXUSRCLK,
  RXUSRCLK2,
  TXBYPASS8B10B,
  TXCHARDISPMODE,
  TXCHARDISPVAL,
  TXCHARISK,
  TXDATA,
  TXFORCECRCERR,
  TXINHIBIT,
  TXPOLARITY,

```



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

## GT\_ETHERNET\_1

```
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,  
    BREFCLK,  
    BREFCLK2,  
    RXN,  
    RXP,  
    RXPOLARITY,  
    RXRESET,  
    RXUSRCLK,  
    RXUSRCLK2,  
    TXBYPASS8B10B,  
    TXCHARDISPMODE,  
    TXCHARDISPVAL,  
    TXCHARISK,  
    TXDATA,  
    TXFORCECRCERR,  
    TXINHIBIT,  
    TXPOLARITY,  
    TXRESET,  
    TXUSRCLK,  
    TXUSRCLK2  
);
```

## GT\_ETHERNET\_2

```

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,
    BREFCLK,
    BREFCLK2,
    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,
BREFCLK,
BREFCLK2,
RXN,
RXP,
RXPOLARITY,
RXRESET,
RXUSRCLK,
RXUSRCLK2,
TXBPASS8B10B,
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,
BREFCLK,
BREFCLK2,
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,
BREFCLK,
BREFCLK2,
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,
    BREFCLK,
    BREFCLK2,
    RXN,
    RXP,
    RXPOLARITY,

```

```

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

```

## GT\_INFINIBAND\_1

```

module GT_INFINIBAND_1 (
    CHBONDDONE,
    CHBONDO,
    CONFIGOUT,
    RXBUFSTATUS,
    RXCHARISCOMMA,
    RXCHARISK,
    RXCHECKINGCRC,
    RXCLKCORCNT,
    RXCOMMADET,
    RXCRCERR,
    RXDATA,
    RXDISPERR,
    RXLOSSOF SYNC,
    RXNOTINTABLE,
    RXREALIGN,
    RXRECCLK,
    RXRUNDISP,
    TXBUFERR,
    TXKERR,
    TXN,
    TXP,
    TXRUNDISP,
    CHBONDI,
    CONFIGENABLE,
    CONFIGIN,
    ENCHANSYNC,
    LOOPBACK,
    POWERDOWN,
    REFCLK,
    REFCLK2,
    REFCLKSEL,
    BREFCLK,
    BREFCLK2,
    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,
    BREFCLK,
    BREFCLK2,
    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,
    BREFCLK,
    BREFCLK2,
    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,
    BREFCLK,
    BREFCLK2,
    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,
    BREFCLK,
    BREFCLK2,
    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,  
BREFCLK,  
BREFCLK2,  
RXN,  
RXP,  
RXPOLARITY,  
RXRESET,  
RXUSRCLK,  
RXUSRCLK2,  
TXBYPASS8B10B,  
TXCHARDISPMODE,  
TXCHARDISPVAL,  
TXCHARISK,  
TXDATA,  
TXFORCECRCERR,  
TXINHIBIT,  
TXPOLARITY,  
TXRESET,  
TXUSRCLK,  
TXUSRCLK2  
);
```

