

# LogiCORE™ IP Aurora 8B/10B v5.1

## *User Guide*

UG353 (v5.1) December 2, 2009



Xilinx is providing this product documentation, hereinafter “Information,” to you “AS IS” with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice.

XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.

© 2008-2009 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

---

## Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/24/08	2.9	Initial Xilinx release.
03/24/08	2.9.1	Post-release updates and corrections.
06/27/08	3.0	Virtex-5 FPGA Aurora v3.0 release. Added <a href="#">“Using the Build Script”</a> section. Added <a href="#">“Using the Build Script”</a> section. Corrected UFC_TX_MS[0:2] data in <a href="#">Figure 4-5</a> . Added PMA_INIT and INIT_CLK to <a href="#">Table 5-1</a> .
04/24/09	4.1	<p>LogiCORE IP Aurora 8B/10B v4.1 release. Updated tools to v11.1. Renamed document title to include the 8B/10B protocol. Renamed Clock Source to <a href="#">“Clock Source1 &amp; Clock Source2.”</a> Replaced the perl script topic with the shell script topic in <a href="#">“Using the Build Script.”</a> Deleted GREFCLK. Renamed RX/TX simplex Both to RX/TX Simplex, TX simplex only to TX-only Simplex, RX simplex only to RX-only Simplex, duplex to Duplex, and ERROR_COUNT to ERR_COUNT. Updated many figures and tables throughout.</p> <p>Added the following:</p> <ul style="list-style-type: none"><li>• Framing and Streaming figures in <a href="#">Chapter 2, “Customizing the Aurora 8B/10B Core”</a></li><li>• <a href="#">“FRAME_GEN”</a> section</li><li>• <a href="#">“FRAME_CHECK”</a> section</li><li>• <a href="#">“Designing with the Core”</a> section</li><li>• <a href="#">“Top Level Architecture”</a> section</li><li>• <a href="#">Appendix A, “Two Aurora 8B/10B Cores in the Virtex-5 Family Sharing a High-Speed Serial GTX Transceiver Tile”</a></li><li>• <a href="#">Appendix B, “Handling Timing Errors”</a></li><li>• <a href="#">Appendix C, “Performance and Core Latency”</a></li><li>• <a href="#">Appendix D, “Generating a Virtex-5 FPGA GTP/GTX Wrapper File from the RocketIO GTP/GTX Transceiver Wizard”</a></li></ul>
06/24/09	4.2	LogiCORE IP Aurora 8B/10B v4.2 release. Updated tools to v11.2. Removed callouts <a href="#">Figure 2-1, page 22</a> . Added <a href="#">“Back Channel,” page 25</a> and <a href="#">“Column Used,” page 26</a> , Corrected “can” to “cannot” in <a href="#">Table 5-1, page 71</a> . Replaced all occurrences of DCM with PLL. Replaced all occurrences of DCM_NOT_LOCKED with PLL_NOT_LOCKED. Major updates to <a href="#">Table 6-1, page 87</a> .
12/02/09	5.1	<p>LogiCORE IP Aurora 8B/10B v5.1 release. Updated tools to v11.4. Added support for Spartan-6 FPGA GTP transceivers. Updated <a href="#">“Using the IP Customizer.”</a> Added <a href="#">“Framing NFC RX Interface”</a> and <a href="#">“Reference Clocks for Spartan-6 FPGA GTP Transceiver Designs.”</a></p> <p>Replaced bus range element <math>16wn</math> to <math>wn</math>; REF_CLK to GT_REFCLK; and BUFDS to IBUFDS. Updated <a href="#">“Steps to Modify Transceiver Specific Ports and Attributes”</a> in <a href="#">Appendix A</a> and <a href="#">“Solutions”</a> in <a href="#">Appendix B</a>.</p>



# Table of Contents

---

Revision History .....	3
<b>Schedule of Figures</b> .....	9
<b>Schedule of Tables</b> .....	13
<b>Preface: About This Guide</b>	
Contents .....	15
Additional Resources .....	15
Conventions .....	16
Typographical .....	16
<b>Chapter 1: Introduction</b>	
About the Core .....	19
Recommended Design Experience .....	19
Related Xilinx Documents .....	19
Additional Core Resources .....	20
Technical Support .....	20
Feedback .....	20
Aurora 8B/10B Core .....	20
Document .....	20
<b>Chapter 2: Customizing the Aurora 8B/10B Core</b>	
Introduction .....	21
Using the IP Customizer .....	21
IP Customizer .....	21
Using the Build Script .....	27
Example Design Overview .....	27
FRAME_GEN .....	29
Framing TX Data Interface .....	29
Streaming TX Data Interface .....	31
Framing UFC TX Interface .....	32
Framing NFC TX Interface .....	33
FRAME_CHECK .....	34
Framing RX Data Interface .....	34
Streaming RX Data Interface .....	35
Framing UFC RX Interface .....	36
Framing NFC RX Interface .....	38
Designing with the Core .....	39
General Design Guidelines .....	39
Use the Example Design as a Starting Point .....	39
Know the Degree of Difficulty .....	39

Keep It Registered .....	39
Recognize Timing Critical Signals .....	39
Use Supported Design Flows .....	39
Make Only Allowed Modifications .....	40

## Chapter 3: User Interface

<b>Introduction</b> .....	41
<b>Top Level Architecture</b> .....	41
<b>Framing Interface</b> .....	43
LocalLink TX Ports .....	44
LocalLink RX Ports .....	45
LocalLink Bit Ordering .....	45
Transmitting Data .....	46
Receiving Data .....	50
Framing Efficiency .....	52
<b>Streaming Interface</b> .....	55
Streaming TX Ports .....	55
Streaming RX Ports .....	56
Transmitting and Receiving Data .....	56

## Chapter 4: Flow Control

<b>Introduction</b> .....	59
<b>Native Flow Control</b> .....	60
Example A: Transmitting an NFC Message .....	61
Example B: Receiving a Message with NFC Idles Inserted .....	61
<b>User Flow Control</b> .....	62
Transmitting UFC Messages .....	63
Receiving User Flow Control Messages .....	67

## Chapter 5: Status, Control, and the GTP/GTX Block Interface

<b>Introduction</b> .....	69
<b>Full-Duplex Cores</b> .....	70
Full-Duplex Status and Control Ports .....	70
Error Signals in Full-Duplex Cores .....	72
Full-Duplex Initialization .....	74
<b>Simplex Cores</b> .....	74
Simplex TX Status and Control Ports .....	74
Simplex RX Status and Control Ports .....	75
Simplex Both (RX and TX) Status and Control Ports .....	77
Error Signals in Simplex Cores .....	80
Simplex Initialization .....	81
<b>Reset and Power Down</b> .....	82
Reset .....	82
Power Down .....	82
Timing .....	83

## Chapter 6: Clock Interface and Clocking

<b>Introduction</b> .....	85
---------------------------	----

Clock Interface Ports for GTP/GTX Transceiver Cores .....	86
Reference Clocks for Virtex-5 FPGA GTP/GTX Transceiver Designs .....	87
Clocking from an External Source .....	88
Clocking from a Neighboring GTP/GTX_DUAL Tile .....	89
Clock from a Neighboring GTX_QUAD Tile for Virtex-6 FPGA Designs ..	90
Reference Clocks for Spartan-6 FPGA GTP Transceiver Designs .....	92
Clocking from and External Source .....	92
Clocking from a Neighboring GTPA1_DUAL Tile .....	93
Clock Rates for GTP/GTX Transceiver Designs .....	94

## Chapter 7: Clock Compensation

Introduction .....	95
Clock Compensation Interface .....	96

## Appendix A: Two Aurora 8B/10B Cores in the Virtex-5 Family Sharing a High-Speed Serial GTX Transceiver Tile

Shared Ports and Attributes .....	101
Steps to Modify Transceiver Specific Ports and Attributes .....	102
Steps to Bring Out Unused Transceiver Ports to the Aurora 8B/10B GTX Transceiver Wrapper File .....	102
Steps to Instantiate Two Aurora 8B/10B Cores in the Top Level File .....	103

## Appendix B: Handling Timing Errors

Solutions .....	105
-----------------	-----

## Appendix C: Performance and Core Latency

Introduction .....	107
Latency of the Frame path .....	107

## Appendix D: Generating a Virtex-5 FPGA GTP/GTX Wrapper File from the RocketIO GTP/GTX Transceiver Wizard



# Schedule of Figures

---

## Preface: About This Guide

## Chapter 1: Introduction

## Chapter 2: Customizing the Aurora 8B/10B Core

<i>Figure 2-1: Aurora 8B/10B IP Customizer</i> .....	22
<i>Figure 2-2: Second GUI Page for Virtex-5 FPGA RocketIO GTP Transceivers</i> .....	22
<i>Figure 2-3: Second GUI Page for Virtex-5 FPGA RocketIO GTX Transceivers</i> .....	23
<i>Figure 2-4: Second GUI Page for Virtex-6 FPGA GTX Transceivers</i> .....	23
<i>Figure 2-5: Second GUI Page for Spartan-6 FPGA GTP Transceivers</i> .....	24
<i>Figure 2-6: Example Design</i> .....	27
<i>Figure 2-7: Aurora 8B/10B Core Framing TX Data Interface (FRAME_GEN)</i> .....	29
<i>Figure 2-8: Aurora 8B/10B Core Streaming TX Data Interface (FRAME_GEN)</i> .....	31
<i>Figure 2-9: Aurora 8B/10B Core Framing UFC TX Interface (FRAME_GEN)</i> .....	32
<i>Figure 2-10: Aurora 8B/10B Core Framing NFC TX Interface (FRAME_GEN)</i> .....	33
<i>Figure 2-11: Aurora 8B/10B Core Framing RX Data Interface (FRAME_CHECK)</i> .....	34
<i>Figure 2-12: Aurora 8B/10B Core Streaming RX Data Interface (FRAME_CHECK)</i> .....	35
<i>Figure 2-13: Aurora 8B/10B Core Framing UFC RX Interface (FRAME_CHECK)</i> .....	36
<i>Figure 2-14: Aurora 8B10B Core Framing NFC RX Interface (FRAME_CHECK)</i> .....	38

## Chapter 3: User Interface

<i>Figure 3-1: Top-Level Architecture</i> .....	42
<i>Figure 3-2: Top-Level User Interface</i> .....	43
<i>Figure 3-3: Aurora 8B/10B Core Framing Interface (LocalLink)</i> .....	43
<i>Figure 3-4: LocalLink Interface Bit Ordering</i> .....	45
<i>Figure 3-5: Simple Data Transfer</i> .....	47
<i>Figure 3-6: Data Transfer with Pad</i> .....	48
<i>Figure 3-7: Data Transfer with Pause</i> .....	48
<i>Figure 3-8: Data Transfer Paused by Clock Compensation</i> .....	49
<i>Figure 3-9: Transmitting Data</i> .....	50
<i>Figure 3-10: Data Reception with Pause</i> .....	51
<i>Figure 3-11: Receiving Data</i> .....	52
<i>Figure 3-12: Formula for Calculating Overhead</i> .....	53
<i>Figure 3-13: Aurora 8B/10B Core Streaming User Interface</i> .....	55
<i>Figure 3-14: Typical Streaming Data Transfer</i> .....	57
<i>Figure 3-15: Typical Data Reception</i> .....	57

## Chapter 4: Flow Control

<i>Figure 4-1: Top-Level Flow Control</i> .....	59
<i>Figure 4-2: Transmitting an NFC Message</i> .....	61
<i>Figure 4-3: Transmitting a Message with NFC Idles Inserted</i> .....	62
<i>Figure 4-4: Data Switching Circuit</i> .....	63
<i>Figure 4-5: Transmitting a Single-Cycle UFC Message</i> .....	65
<i>Figure 4-6: Transmitting a Multi-Cycle UFC Message</i> .....	66
<i>Figure 4-7: Receiving a Single-Cycle UFC Message</i> .....	67
<i>Figure 4-8: Receiving a Multi-Cycle UFC Message</i> .....	67

## Chapter 5: Status, Control, and the GTP/GTX Block Interface

<i>Figure 5-1: Top-Level GTP/GTX Block Interface</i> .....	70
<i>Figure 5-2: Status and Control Interface for Full-Duplex Cores</i> .....	70
<i>Figure 5-3: Status and Control Interface for Simplex TX Core</i> .....	74
<i>Figure 5-4: Status and Control Interface for Simplex RX Core</i> .....	75
<i>Figure 5-5: Status and Control Interface for Simplex Both Cores</i> .....	77
<i>Figure 5-6: Reset and Power Down Timing</i> .....	83

## Chapter 6: Clock Interface and Clocking

<i>Figure 6-1: Top-Level Clocking</i> .....	86
<i>Figure 6-2: Single GTP/GTX_DUAL Tile Clocked Externally</i> .....	88
<i>Figure 6-3: Multiple Tiles with a Shared Reference Clock</i> .....	89
<i>Figure 6-4: North-South Routing Adjustments in Virtex-6 FPGAs</i> .....	91
<i>Figure 6-5: Single GTPA1_DUAL Tile Clocked Externally</i> .....	92
<i>Figure 6-6: GTPA1_DUAL Tiles with Shared Reference Clock</i> .....	93

## Chapter 7: Clock Compensation

<i>Figure 7-1: Top-Level Clock Compensation</i> .....	96
<i>Figure 7-2: Streaming Data with Clock Compensation Inserted</i> .....	97
<i>Figure 7-3: Data Reception Interrupted by Clock Compensation</i> .....	97

## Appendix A: Two Aurora 8B/10B Cores in the Virtex-5 Family Sharing a High-Speed Serial GTX Transceiver Tile

<i>Figure A-1: Example Showing Two Single Lane Aurora 8B/10B Cores Sharing a Transceiver Tile in the Virtex-5 Family</i> .....	103
--	-----

## Appendix B: Handling Timing Errors

## Appendix C: Performance and Core Latency

<i>Figure C-1: Latency of the Frame Path</i> .....	107
--	-----

## Appendix D: Generating a Virtex-5 FPGA GTP/GTX Wrapper File from the RocketIO GTP/GTX Transceiver Wizard



# Schedule of Tables

---

## Preface: About This Guide

## Chapter 1: Introduction

## Chapter 2: Customizing the Aurora 8B/10B Core

<i>Table 2-1: Example Design I/O Ports</i> .....	28
<i>Table 2-2: FRAME_GEN Framing User I/O Ports (TX)</i> .....	29
<i>Table 2-3: FRAME_GEN Streaming User I/O Ports (TX)</i> .....	31
<i>Table 2-4: FRAME_GEN Framing UFC User I/O Ports (TX)</i> .....	32
<i>Table 2-5: FRAME_GEN Framing NFC User I/O Ports (TX)</i> .....	33
<i>Table 2-6: FRAME_CHECK Framing User I/O Ports (RX)</i> .....	34
<i>Table 2-7: FRAME_CHECK Streaming User I/O Ports (RX)</i> .....	36
<i>Table 2-8: FRAME_CHECK Framing UFC User I/O Ports (RX)</i> .....	37
<i>Table 2-9: Aurora 8B10B Core Framing NFC RX Interface (FRAME_CHECK)</i> .....	38

## Chapter 3: User Interface

<i>Table 3-1: LocalLink User I/O Ports (TX)</i> .....	44
<i>Table 3-2: LocalLink User I/O Ports (RX)</i> .....	45
<i>Table 3-3: TX Data Remainder Values</i> .....	46
<i>Table 3-4: Typical Channel Frame</i> .....	47
<i>Table 3-5: Efficiency Example</i> .....	54
<i>Table 3-6: Typical Overhead for Transmitting 256 Data Bytes</i> .....	54
<i>Table 3-7: TX_REM Value and Corresponding Bytes of Overhead</i> .....	55
<i>Table 3-8: Streaming User I/O Ports (TX)</i> .....	55
<i>Table 3-9: Streaming User I/O Ports (RX)</i> .....	56

## Chapter 4: Flow Control

<i>Table 4-1: NFC Codes</i> .....	60
<i>Table 4-2: NFC I/O Ports</i> .....	60
<i>Table 4-3: UFC I/O Ports</i> .....	62
<i>Table 4-4: SIZE Encoding</i> .....	63
<i>Table 4-5: Number of Data Beats Required to Transmit UFC Messages</i> .....	64

## Chapter 5: Status, Control, and the GTP/GTX Block Interface

<i>Table 5-1: Status and Control Ports for Full-Duplex Cores</i> .....	71
<i>Table 5-2: Error Signals in Full-Duplex Cores</i> .....	73
<i>Table 5-3: Status and Control Ports for Simplex TX Cores</i> .....	74
<i>Table 5-4: Status and Control Ports for Simplex RX Cores</i> .....	76

<i>Table 5-5: Status and Control Ports for Simplex Both Cores</i> .....	78
<i>Table 5-6: Error Signals in Simplex Cores</i> .....	80

## **Chapter 6: Clock Interface and Clocking**

<i>Table 6-1: Clock Ports for a GTP/GTX Aurora 8B/10B Core</i> .....	87
--	----

## **Chapter 7: Clock Compensation**

<i>Table 7-1: Clock Compensation I/O Ports</i> .....	96
<i>Table 7-2: Clock Compensation Cycles</i> .....	97
<i>Table 7-3: Lookahead Cycles</i> .....	98
<i>Table 7-4: Standard CC I/O Port</i> .....	99

## **Appendix A: Two Aurora 8B/10B Cores in the Virtex-5 Family Sharing a High-Speed Serial GTX Transceiver Tile**

## **Appendix B: Handling Timing Errors**

## **Appendix C: Performance and Core Latency**

## **Appendix D: Generating a Virtex-5 FPGA GTP/GTX Wrapper File from the RocketIO GTP/GTX Transceiver Wizard**

# About This Guide

---

The *LogiCORE IP Aurora 8B/10B v5.1 User Guide* provides information for generating a LogiCORE™ IP Aurora 8B/10B core using Virtex®-5 FPGA GTP/GTX transceivers and Virtex-6 FPGA GTX transceivers. The core implements the Aurora 8B/10 protocol using the high-speed serial transceivers on the Virtex-5 LXT, SXT, FXT, and TXT family, the Virtex-6 LXT/SXT, CXT, and Lower Power family, and the Spartan®-6 family. This user guide describes the function and operation of the LogiCORE™ IP Aurora 8B/10B v5.1 core and provides information about designing, customizing, and implementing the core.

## Contents

This guide contains the following chapters:

- [Preface, “About this Guide”](#) introduces the organization and purpose of the design guide, a list of additional resources, and the conventions used in this document.
- [Chapter 1, “Introduction”](#) describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- [Chapter 2, “Customizing the Aurora 8B/10B Core”](#) provides port descriptions for the user interface.
- [Chapter 4, “Flow Control”](#) describes the user flow control and native flow control options for sending and receiving data.
- [Chapter 5, “Status, Control, and the GTP/GTX Block Interface”](#) provides details on using the Aurora 8B/10B core's status and control ports to initialize and monitor the Aurora 8B/10B channel.
- [Chapter 6, “Clock Interface and Clocking”](#) describes how to connect FPGA clocking resources.
- [Chapter 7, “Clock Compensation”](#) covers Aurora 8B/10B clock compensation, and explains how to customize it for a given system.

## Additional Resources

To find additional documentation, see the Xilinx website at:

<http://www.xilinx.com/support/documentation/index.htm>.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

<http://www.xilinx.com/support/mysupport.htm>.

## Conventions

This document uses the following conventions. An example illustrates each convention.

### Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
Courier font	Messages, prompts, and program files that the system displays	<code>speed grade: - 100</code>
<b>Courier bold</b>	Literal commands that you enter in a syntactical statement	<code><b>ngdbuild</b> design_name</code>
<b>Helvetica bold</b>	Commands that you select from a menu	<b>File</b> → <b>Open</b>
	Keyboard shortcuts	<b>Ctrl+C</b>
<i>Italic font</i>	Variables in a syntax statement for which you must supply values	<code><i>ngdbuild</i> design_name</code>
	References to other manuals	See the <i>User Guide</i> for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are <i>not</i> connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as <b>bus [7:0]</b> , they are required.	<code><b>ngdbuild</b> [<i>option_name</i>] design_name</code>
Braces { }	A list of items from which you must choose one or more	<code><b>lowpwr</b> = {<b>on</b>   <b>off</b>}</code>
Vertical bar	Separates items in a list of choices	<code><b>lowpwr</b> = {<b>on</b>   <b>off</b>}</code>
Angle brackets < >	User-defined variable or in code samples	<directory name>
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Repetitive material that has been omitted	<code><b>allow block</b> block_name loc1 loc2 ... locn;</code>

Convention	Meaning or Use	Example
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	<b>usr_teof_n</b> is active low.



# Introduction

---

This chapter introduces the LogiCORE™ IP Aurora 8B/10B core and provides related information, including recommended design experience, additional resources, technical support, and how to submit feedback to Xilinx. The Aurora core is a high-speed serial solution based on the Aurora protocol, Virtex®-5 FPGA GTP/GTX transceivers, Virtex-6 FPGA GTX transceivers, and Spartan®-6 FPGA GTP transceivers. The core is delivered as open-source code and supports both Verilog and VHDL design environments. Each core comes with an example design and supporting modules.

## About the Core

The Aurora 8B/10B core is a CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see [www.xilinx.com/aurora](http://www.xilinx.com/aurora). For information about system requirements, installation, and licensing options, see the *LogiCORE IP Aurora 8B/10B v5.1 Getting Started Guide*.

## Recommended Design Experience

Although the Aurora core is a fully verified solution, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application. For best results, previous experience building high-performance, pipelined FPGA designs using Xilinx implementation software and user constraints files (.ucf) is recommended. Read [Chapter 5, “Status, Control, and the GTP/GTX Block Interface”](#) carefully, and consult the PCB design requirements information in the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide*, *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*, *Virtex-6 FPGA GTX Transceivers User Guide*, and *Spartan-6 FPGA GTP Transceivers User Guide*. Contact your local Xilinx representative for a closer review and estimation for your specific requirements.

## Related Xilinx Documents

Prior to generating an Aurora core, users should be familiar with the following:

- [SP002](#), *Aurora 8B/10B Protocol Specification*
- SP006, *LocalLink Interface Specification* is located on the [LocalLink product page](#)
- [UG196](#), *Virtex-5 FPGA RocketIO GTP Transceiver User Guide*
- [UG198](#), *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*
- [UG366](#), *Virtex-6 FPGA GTX Transceivers User Guide*
- [UG386](#), *Spartan-6 FPGA GTP Transceivers User Guide*
- ISE® software documentation: [xilinx.com/ise](http://xilinx.com/ise)

## Additional Core Resources

For detailed information and updates about the Aurora core, see the following documents, located on the Aurora product page at [www.xilinx.com/aurora](http://www.xilinx.com/aurora).

- [DS637](#): *Aurora 8B/10B v5.1 Data Sheet*
- [UG352](#): *LogiCORE IP Aurora 8B/10B v5.1 Getting Started Guide*
- UG058: *Aurora 8B/10B Bus Functional Model User Guide* (Contact: [auroramkt@xilinx.com](mailto:auroramkt@xilinx.com))
- Aurora 8B/10B Release Notes
- Aurora Solution list – AR#21263

## Technical Support

For technical support, go to [www.xilinx.com/support](http://www.xilinx.com/support). Questions are routed to a team of engineers with expertise using the Aurora core.

Xilinx will provide technical support for use of this product as described in the *LogiCORE IP Aurora 8B/10B v5.1 User Guide* and the *LogiCORE IP Aurora 8B/10B v5.1 Getting Started Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow the guidelines.

## Feedback

Xilinx welcomes comments and suggestions about the Aurora core and the accompanying documentation.

### Aurora 8B/10B Core

For comments or suggestions about the Aurora 8B/10B core, please submit a WebCase from <http://www.xilinx.com/support/clarexpress/websupport.htm>. Be sure to include the following information:

- Product name
- Core version number
- List of parameter settings
- Explanation of your comments

### Document

For comments or suggestions about the Aurora 8B/10B documents, please submit a WebCase from <http://www.xilinx.com/support/clarexpress/websupport.htm>. Be sure to include the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

# Customizing the Aurora 8B/10B Core

---

## Introduction

The Aurora 8B/10B core can be customized to suit a wide variety of requirements using the CORE Generator™ software. This chapter details the customization parameters available to the user and how these parameters are specified within the IP Customizer interface.

## Using the IP Customizer

The Aurora 8B/10B IP Customizer is presented when the user selects the Aurora 8B/10B core in the CORE Generator software. For help starting and using the CORE Generator software, see the *CORE Generator Help* in the ISE® software documentation. Each numbered item in [Figure 2-1](#) corresponds to its respective section that describes the purpose of the feature.

### IP Customizer

[Figure 2-1, page 22](#) shows the customizer. The left side displays a representative block diagram of the Aurora 8B/10B core as currently configured. The right side consists of user-configurable parameters.

The second pages of the GUI are shown in:

- [Figure 2-2, page 22](#) for Virtex®-5 FPGA RocketIO™ GTP transceivers
- [Figure 2-3, page 23](#) for Virtex-5 FPGA RocketIO GTX transceivers
- [Figure 2-4, page 23](#) for Virtex-6 FPGA GTX transceivers
- [Figure 2-5, page 24](#) for Spartan®-6 FPGA GTP transceivers

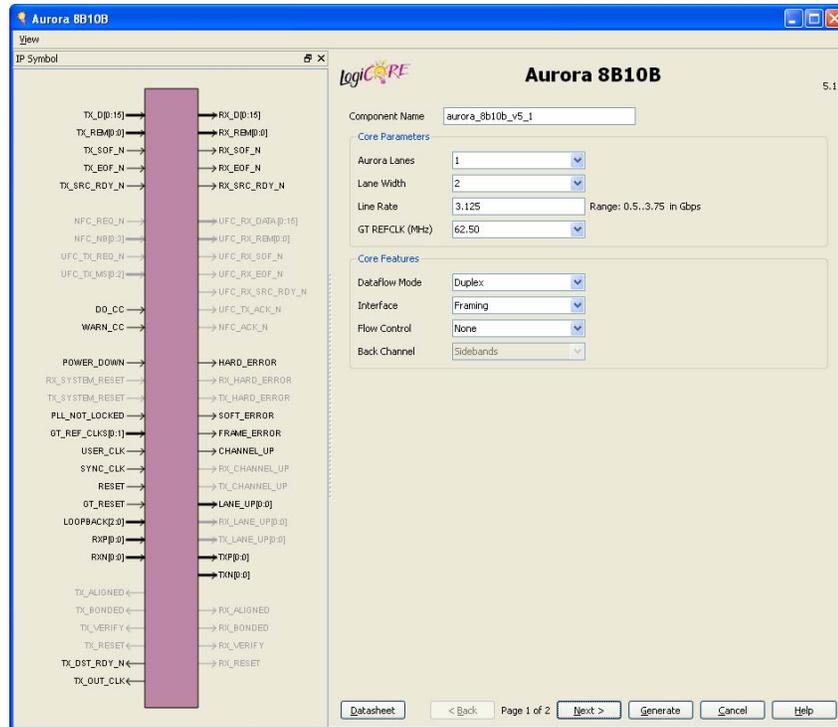


Figure 2-1: Aurora 8B/10B IP Customizer

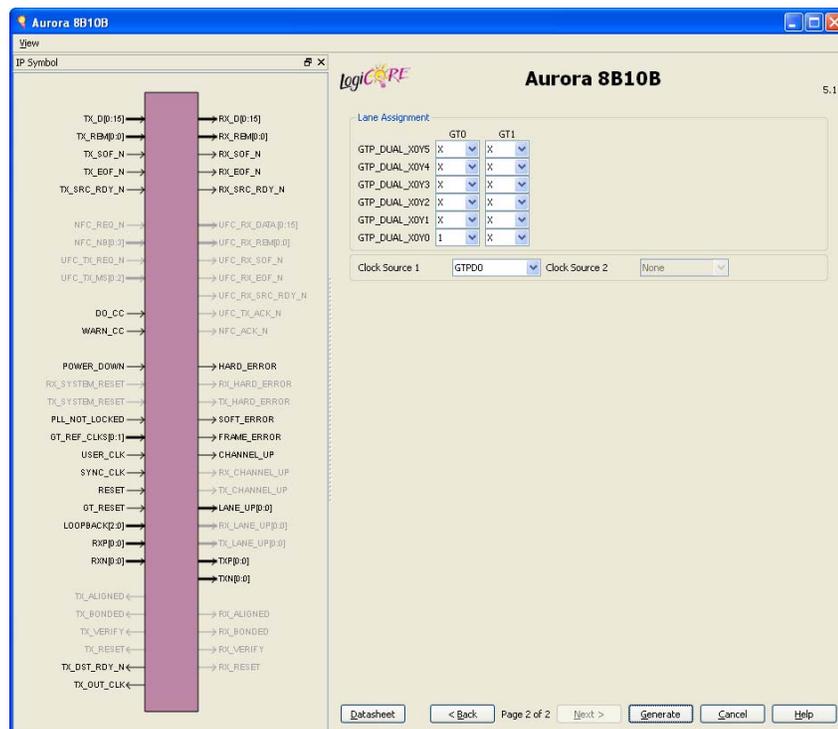


Figure 2-2: Second GUI Page for Virtex-5 FPGA RocketIO GTP Transceivers

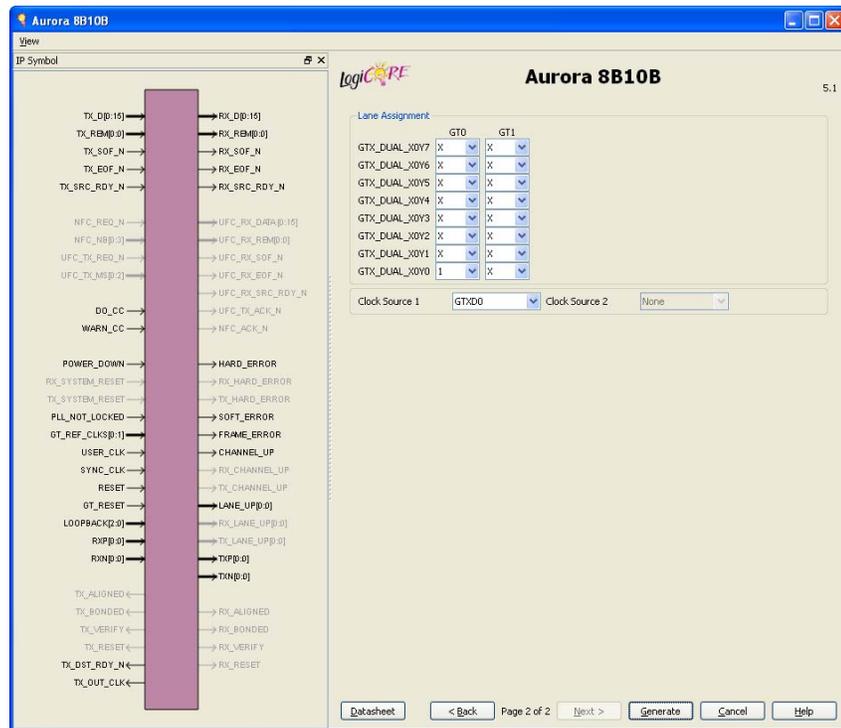


Figure 2-3: Second GUI Page for Virtex-5 FPGA RocketIO GTX Transceivers

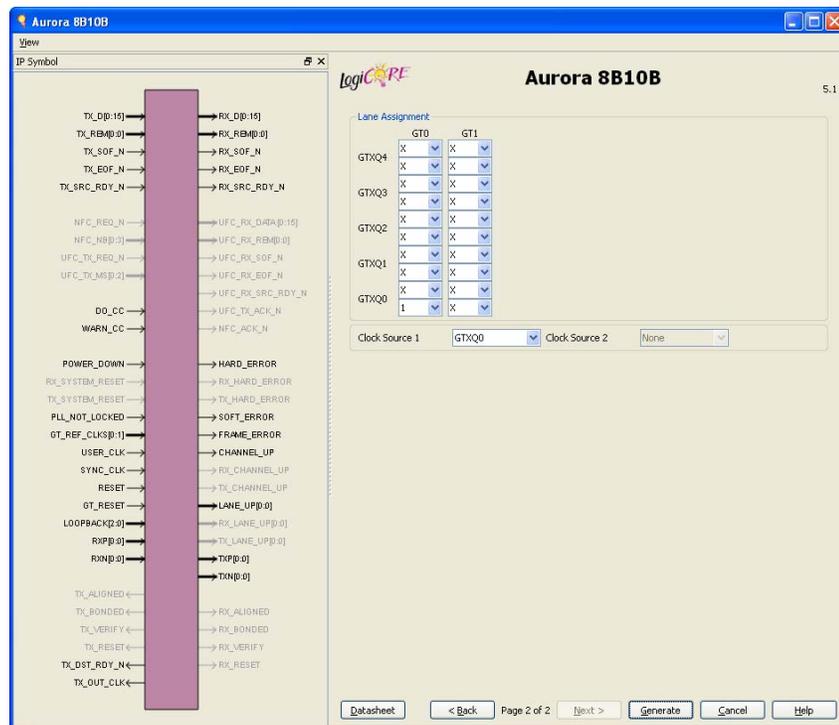


Figure 2-4: Second GUI Page for Virtex-6 FPGA GTX Transceivers

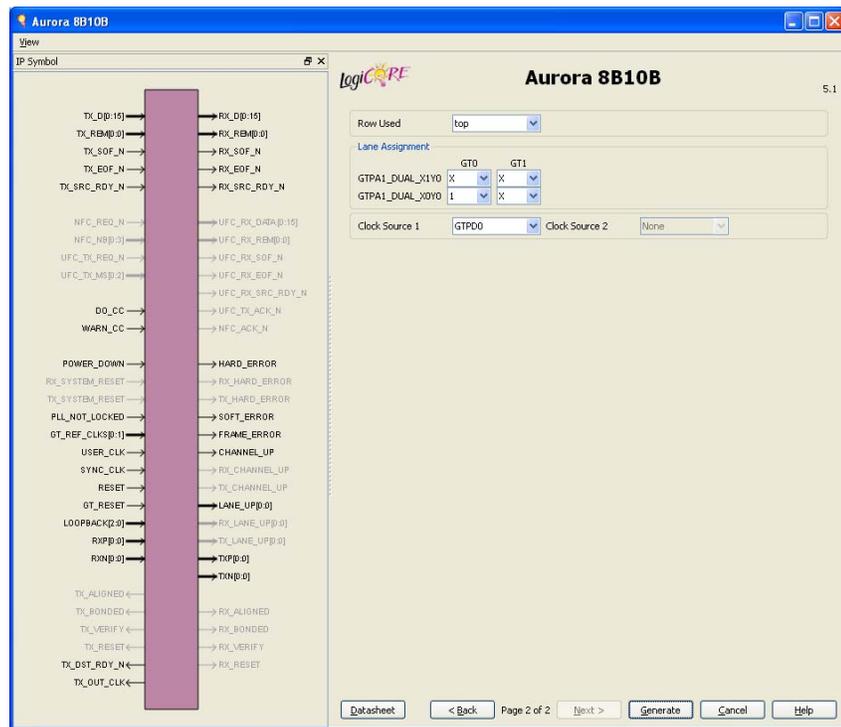


Figure 2-5: Second GUI Page for Spartan-6 FPGA GTP Transceivers

## Component Name

Enter the top-level name for the core in this text box. Illegal names are highlighted in red until they are corrected. All files for the generated core are placed in a subdirectory using this name. The top-level module for the core also uses this name.

Default: `aurora_8b10b_v5_1`

## Lane Assignment

Refer to the diagram in the information area in [Figure 2-1, page 22](#). Each numbered column represents a GTP/GTX\_DUAL or two columns represents GTX Quad and each active box represents an available GTP/GTX transceiver.

## Aurora 8B/10B Lanes

Select the number of lanes (GTP/GTX transceivers) to be used in the core. The valid range depends on the target device selected.

Default: 1

## Lane Width

Select the byte width of the GTP/GTX transceivers used in the core.

Default: 2

## Interface

Select the type of data path interface used for the core. Select *Framing* to use a LocalLink interface that allows encapsulation of data frames of any length. Select *Streaming* to use a simple word-based interface with a data valid signal to stream data through the Aurora 8B/10B channel. See [Chapter 3, “User Interface”](#) for more information.

Default: Framing

## Dataflow Mode

Select the options for direction of the channel the Aurora 8B/10B core will support. Simplex Aurora 8B/10B cores have a single, unidirectional serial port that connects to a complementary simplex Aurora 8B/10B core. Available options are *RX-only Simplex*, *TX-only Simplex*, *RX/TX Simplex*, and *Duplex*. RX/TX Simplex creates two cores, one RX and one TX, that share a single GTP/GTX transceiver. See [Chapter 5, “Status, Control, and the GTP/GTX Block Interface”](#) for more information.

Default: Duplex

## Back Channel

Select the options for Back Channel only for Simplex Aurora cores; Duplex Aurora cores does not require this option. The available options are:

- Sidebands
- Timer

Default: Sidebands

**Note:** There is no functionality difference between RX-only Simplex design with Sidebands option and RX-only Simplex design with Timer option.

## Flow Control

Select the required option to add flow control to the core. User flow control (UFC) allows applications to send a brief, high-priority message through the Aurora 8B/10B channel. Native flow control (NFC) allows full duplex receivers to regulate the rate of the data send to them. Immediate mode allows idle codes to be inserted within data frames while completion mode only inserts idle codes between complete data frames.

Available options are:

- None
- UFC
- Immediate Mode - NFC
- Completion Mode - NFC
- UFC + Immediate Mode - NFC
- UFC + Completion Mode - NFC

See [Chapter 4, “Flow Control”](#) for more information.

Default: None

## Line Rate

Enter a floating-point value in gigabits per second within the valid range. This determines the un-encoded bit rate at which data will be transferred over the serial link. The aggregate data rate of the core is  $(0.8 \times \text{line rate}) \times \text{Aurora 8B/10B lanes}$ .

Default: 3.125 Gbps

## Reference Clock Frequency

Select a reference clock frequency from the drop-down list. Reference clock frequencies are given in megahertz (MHz), and depend on the line rate selected. For best results, select the highest rate that can be practically applied to the reference clock input of the target device.

Default: 156.25 MHz

## Clock Source1 & Clock Source2

Select reference clock sources for the GTP/GTX\_DUALs or GTX Quad from the drop-down list in this section.

- Default: Clock Source1 - GTXD0; Clock Source2 - None for Virtex-5 FPGA GTX transceivers
- Default: Clock Source1 - GTPD0; Clock Source2 - None for Virtex-5 FPGA GTP and Spartan-6 FPGA GTP transceivers
- Default: Clock Source1 - GTXQ0; Clock Source2 - None for Virtex-6 FPGA GTX transceivers
- GTXD0, GTPD0 and GTXQ0 will change based on the selected device and package.

## Column Used

Select the appropriate column of transceivers used from the drop down list. The column used is enabled only for Virtex-5 TXT devices and is disabled for all other devices.

Default: left

## Row Used

Select the appropriate row of transceivers used from the drop down list. The row used is enabled only for Spartan-6 LXT devices and is disabled for all other devices.

Default: top

## Generate

Click Generate to generate the core. The modules for the Aurora 8B/10B core are written to the CORE Generator software project directory using the same name as the top level of the core. See the *LogiCORE IP Aurora 8B/10B v5.1 Getting Started Guide* for more details about the example\_design directory and files.

## Using the Build Script

A shell script called `implement.sh` is delivered with the Aurora 8B/10B core in the `implement` subdirectory. The script can be run to synthesize using XST, generate project files, or implement the core. Make sure the XILINX environment variable is set properly then run the script by entering the following command in the scripts directory:

```
./implement.sh
```

## Example Design Overview

Each core includes an example design (`aurora_example`) that uses the core in a simple data transfer system. For more details about the `example_design` directory, see the *LogiCORE IP Aurora 8B/10B v5.1 Getting Started Guide*.

In the example design, a frame generator is connected to the TX user interface, and a frame checker is connected to the RX user interface. [Figure 2-6, page 27](#) is a block diagram of the example design for a full-duplex core. [Table 2-1, page 28](#) describes the ports of the example design.

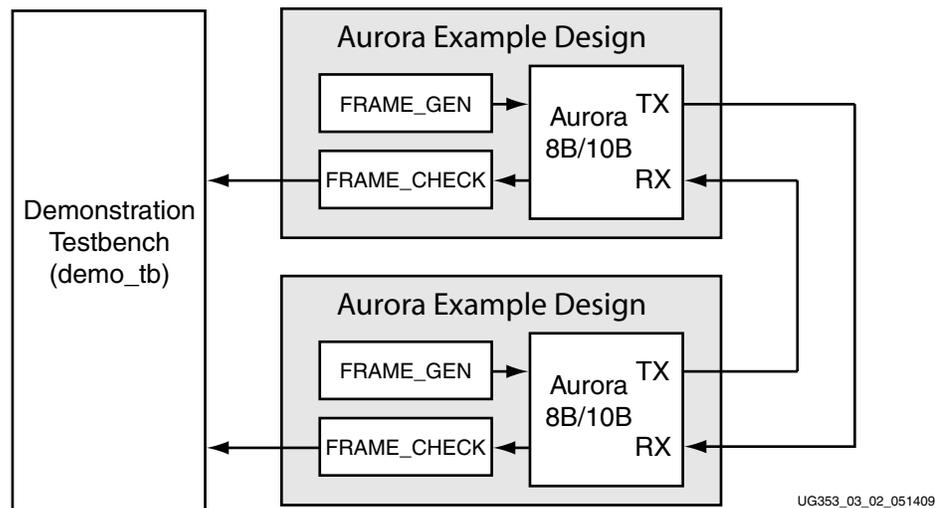


Figure 2-6: Example Design

The example designs uses all the interfaces of the core. Simplex cores without a TX or RX interface will have no `FRAME_GEN` or `FRAME_CHECK` block, respectively. The frame generator produces a constant stream of data for cores with a streaming interface.

Using the scripts provided in the `implement` subdirectory, the example design can be used to quickly get an Aurora 8B/10B design up and running on a board, or perform a quick simulation of the module. The design can also be used as a reference for the connecting the trickier interfaces on the Aurora 8B/10B core, such as the clocking interface.

When using the example design on a board, be sure to edit the `<component_name>_example_design.ucf` file in the `example_design` subdirectory to supply the correct pins and clock constraints.

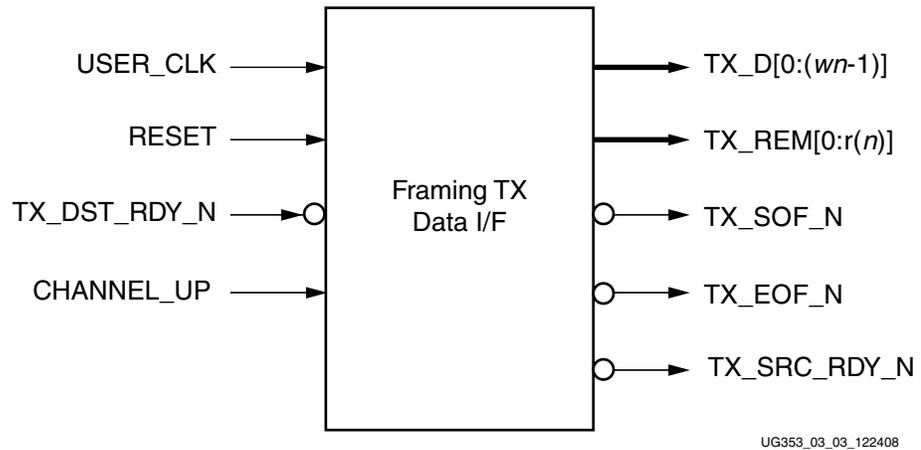
Table 2-1: Example Design I/O Ports

Port	Direction	Description
RXN[0:m-1]	Input	Negative differential serial data input pin.
RXP[0:m-1]	Input	Positive differential serial data input pin.
TXN[0:m-1]	Output	Negative differential serial data output pin.
TXP[0:m-1]	Output	Positive differential serial data output pin.
ERR_COUNT[0:7]	Output	Count of the number of data words received by the frame checker that did not match the expected value.
RESET	Input	Reset signal for the example design. The reset is debounced using a USER_CLK signal generated from the reference clock input.
<reference clock(s)>	Input	The reference clocks for the Aurora 8B/10B core are brought to the top level of the example design. See <a href="#">Chapter 6, “Clock Interface and Clocking”</a> for details about the reference clocks.
<core error signals>	Output	The error signals from the Aurora 8B/10B core's Status and Control interface are brought to the top level of the example design and registered. See <a href="#">Chapter 5, “Status, Control, and the GTP/GTX Block Interface”</a> for details.
<core channel up signals>	Output	The channel up status signals for the core are brought to the top level of the example design and registered. Full-duplex cores will have a single channel up signal; simplex cores will have one for each channel direction supported. See <a href="#">Chapter 5, “Status, Control, and the GTP/GTX Block Interface”</a> for details.
<core lane up signals>	Output	The lane up status signals for the core are brought to the top level of the example design and registered. Cores have a lane up signal for each GTP/GTX transceiver they use. Simplex cores have a separate lane up signal per GTP/GTX transceiver they use for each channel direction supported. See <a href="#">Chapter 5, “Status, Control, and the GTP/GTX Block Interface”</a> for details.
<simplex initialization signals>	Input/Output	If the core is a simplex core, its sideband initialization ports will be registered and brought to the top level of the example design. See <a href="#">Chapter 5, “Status, Control, and the GTP/GTX Block Interface”</a> for details.

# FRAME\_GEN

## Framing TX Data Interface

Figure 2-7 shows the FRAME\_GEN framing user interface of the Aurora 8B/10B core, with LocalLink compliant ports for TX data.



UG353\_03\_03\_122408

Figure 2-7: Aurora 8B/10B Core Framing TX Data Interface (FRAME\_GEN)

Table 2-2 lists the FRAME\_GEN framing TX data ports and their descriptions.

Table 2-2: FRAME\_GEN Framing User I/O Ports (TX)

Name	Direction	Description
TX_D[0:(wn-1)]	Output	Outgoing data. Width is $(wn - 1)$ where $w$ represents GTP/GTX transceiver internal data path width. <ul style="list-style-type: none"> <li><math>w</math> takes values 16 or 32 depending on whether width is 2-byte or 4-byte</li> <li><math>n</math> is the number of lanes</li> </ul>
TX_REM[0:r(n)]	Output	Specifies the number of valid bytes in the last data beat; valid only while TX_EOF_N is asserted. REM bus widths are given by $[0:r(n)]$ , where $r(n) = \text{ceiling} \{ \log_2 n \} - 1$
TX_SRC_RDY_N	Output	Asserted (Low) when LocalLink signals and/or data from the source are valid. Deasserted (High) when LocalLink control signals and/or data from the source should be ignored (active-Low).
TX_SOF_N	Output	Signals the start of the outgoing channel frame (active-Low).
TX_EOF_N	Output	Signals the end of the frame (active-Low).

Table 2-2: FRAME\_GEN Framing User I/O Ports (TX) (Cont'd)

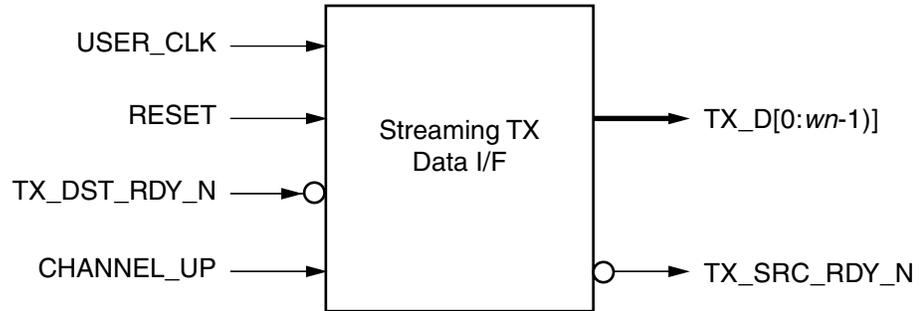
Name	Direction	Description
TX_DST_RDY_N	Input	Asserted (Low) during clock edges when signals from the source will be accepted (if TX_SRC_RDY_N is also asserted). Deasserted (High) on clock edges when signals from the source will be ignored.
CHANNEL_UP	Input	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to transmit data.
USER_CLK	Input	Parallel clock shared by the Aurora 8B/10B core and the user application.
RESET	Input	Resets the Aurora 8B/10B core (active-High).

To transmit the user data, the FRAME\_GEN user data state machine manipulates control signals to do the following:

- After the Aurora 8B/10B interface is out of RESET and reaches the CHANNEL\_UP state, pseudo-random data is generated using user data linear feedback shift register (LFSR) and connected to TX\_D bus.
- Generate the TX\_SOF\_N and TX\_EOF\_N for the current frame based on two counters. An 8-bit counter determines the size of the frame and another 8-bit counter keeps track of the number of user data bytes that has been sent. Frame size counter is initialized to zero and incremented by one for every frame.
- TX\_REM bus is connected to lower bits of user data LFSR.
- TX\_SRC\_RDY\_N is asserted according to LocalLink framing protocol specification.
- User data state machine state transitions are controlled by TX\_DST\_RDY\_N provided by Aurora 8B/10B's LocalLink interface.
- Various kinds of frame traffic are generated including single cycle frame where in TX\_SOF\_N, TX\_EOF\_N are asserted in same cycle.

## Streaming TX Data Interface

Figure 2-8 shows the FRAME\_GEN streaming user interface of the Aurora 8B/10B core ports for TX data.



UG353\_03\_04\_121208

Figure 2-8: Aurora 8B/10B Core Streaming TX Data Interface (FRAME\_GEN)

Table 2-3 lists the FRAME\_GEN streaming TX data ports and their descriptions.

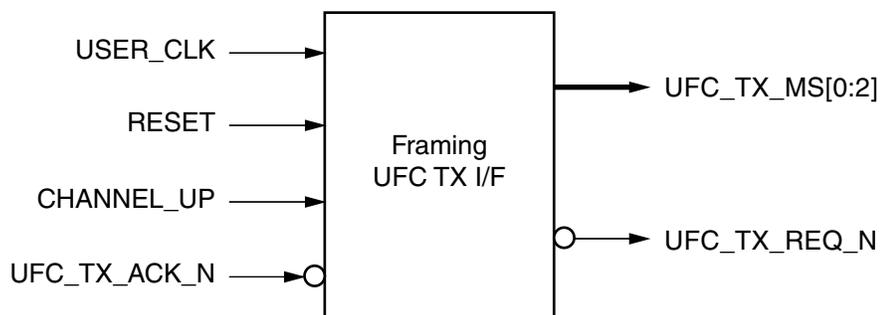
Table 2-3: FRAME\_GEN Streaming User I/O Ports (TX)

Name	Direction	Description
TX_D[0:(wn-1)]	Output	Outgoing data. Width is $wn$ where $w$ represents GTP/GTX transceiver data path width. <ul style="list-style-type: none"> <li><math>w</math> takes 16 or 32 depending on whether width is 2-byte or 4-byte.</li> <li><math>n</math> is the number of lanes</li> </ul>
TX_SRC_RDY_N	Output	Asserted (Low) when LocalLink signals from the source are valid. Deasserted (High) when LocalLink control signals and/or data from the source should be ignored (active-Low).
TX_DST_RDY_N	Input	Asserted (Low) during clock edges when signals from the source will be accepted (if TX_SRC_RDY_N is also asserted). Deasserted (High) on clock edges when signals from the source will be ignored.
CHANNEL_UP	Input	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to transmit data.
USER_CLK	Input	Parallel clock shared by the Aurora 8B/10B core and the user application.
RESET	Input	Resets the Aurora 8B/10B core (active-High).

After the Aurora 8B/10B interface is out of RESET and reaches the CHANNEL\_UP state, pseudo-random data is generated using LFSR and connected to TX\_D bus. LFSR generates new data for every assertion of TX\_DST\_RDY\_N. TX\_SRC\_RDY\_N is always asserted.

## Framing UFC TX Interface

Figure 2-9 shows the FRAME\_GEN framing UFC TX interface of the Aurora 8B/10B core for UFC TX data.



UG353\_03\_05\_121208

Figure 2-9: Aurora 8B/10B Core Framing UFC TX Interface (FRAME\_GEN)

Table 2-4 lists the FRAME\_GEN framing UFC TX data ports and their descriptions

Table 2-4: FRAME\_GEN Framing UFC User I/O Ports (TX)

Name	Direction	Description
UFC_TX_REQ_N	Output	Asserted to request a UFC message to be sent to the channel partner (active-Low). Must be held until UFC_TX_ACK_N is asserted.
UFC_TX_MS[0:2]	Output	Specifies the size of the UFC message that will be sent. The size encoding is a value between 0 and 7.
CHANNEL_UP	Input	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to transmit data.
USER_CLK	Input	Parallel clock shared by the Aurora 8B/10B core and the user application.
RESET	Input	Resets the Aurora 8B/10B core (active-High).
UFC_TX_ACK_N	Output	Asserted when UFC message transmission starts.

To transmit the UFC data, the FRAME\_GEN UFC state machine manipulates control signals to do the following:

- Asserts UFC\_TX\_REQ\_N after CHANNEL\_UP indication from the Aurora 8B/10B TX interface.
- UFC\_TX\_MS[0:2] is also transmitted along with UFC\_TX\_REQ\_N. UFC\_TX\_MS is sent as zero initially for the first UFC frame and is incremented by one for the next UFC frame until it reaches 7.
- UFC data is transmitted immediately from UFC LFSR after receiving UFC\_TX\_ACK\_N from the Aurora 8B/10B TX interface.
- UFC frame transmission frequency is controlled by UFC\_IFG parameter.

## Framing NFC TX Interface

Figure 2-10 shows the FRAME\_GEN framing NFC TX interface of the Aurora 8B/10B core, with LocalLink compliant ports for NFC TX data.

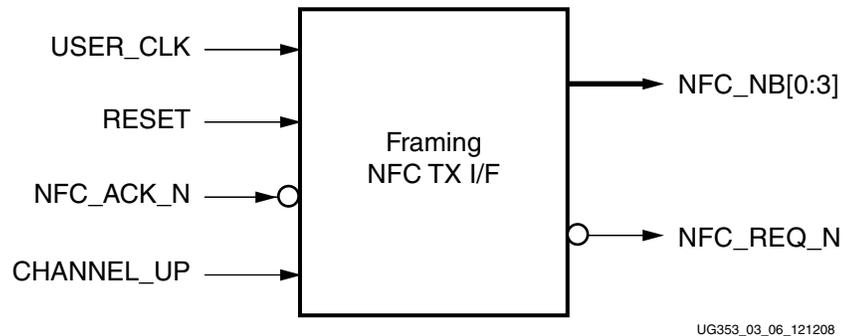


Figure 2-10: Aurora 8B/10B Core Framing NFC TX Interface (FRAME\_GEN)

Table 2-5 lists the FRAME\_GEN framing NFC TX data ports and their descriptions.

Table 2-5: FRAME\_GEN Framing NFC User I/O Ports (TX)

Name	Direction	Description
NFC_REQ_N	Output	Asserted to request an NFC message to be sent to the channel partner (active-Low). Must be held until NFC_ACK_N is asserted.
NFC_NB[0:3]	Output	Indicates the number of PAUSE idles the channel partner must send when it receives the NFC message. Must be held until NFC_ACK_N is asserted.
NFC_ACK_N	Input	Asserted when an Aurora 8B/10B core accepts an NFC request (active-Low).
CHANNEL_UP	Input	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data.
USER_CLK	Input	Parallel clock shared by the Aurora 8B/10B core and the user application.
RESET	Input	Resets the Aurora 8B/10B core (active-High).

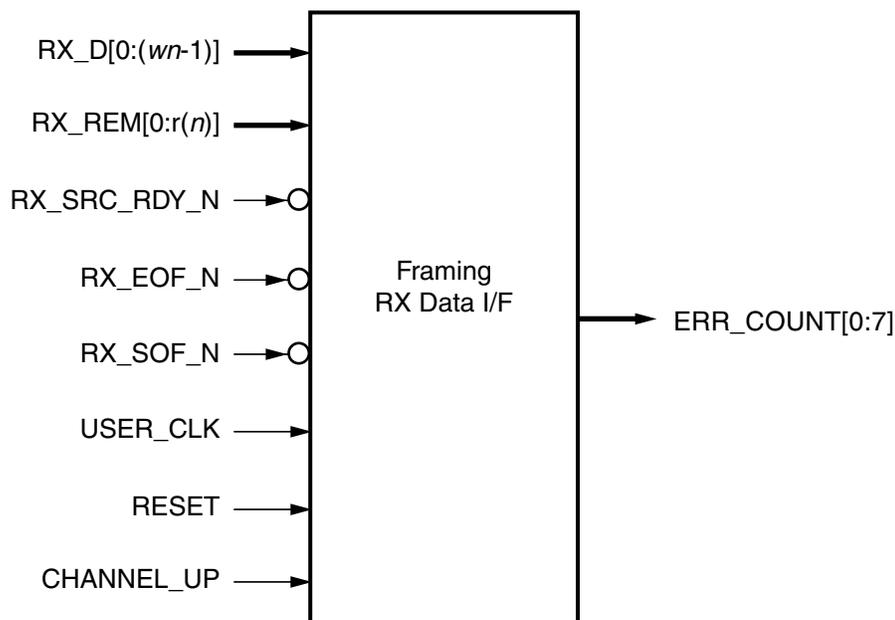
To transmit the NFC data, the FRAME\_GEN NFC state machine manipulates control signals to do the following:

- NFC state machine waits until TX user data transmission and start transmitting first NFC frame after predefined period of time.
- NFC\_NB[0:3] value is transmitted along with NFC\_REQ\_N.
- After a predefined period of time, the NFC state machine enters into NFC XOFF mode.
- NFC state machine enters into NFC XON mode after a predefined period of time.
- NFC frame transmission frequency is controlled by NFC\_IFG parameter.
- NFC state transitions are governed by NFC\_ACK\_N.

## FRAME\_CHECK

### Framing RX Data Interface

Figure 2-11 shows the FRAME\_CHECK framing user interface of the Aurora 8B/10B core, with LocalLink compliant ports for RX data.



UG353\_03\_07\_030309

Figure 2-11: Aurora 8B/10B Core Framing RX Data Interface (FRAME\_CHECK)

Table 2-6 lists the FRAME\_CHECK framing RX data ports and their descriptions.

Table 2-6: FRAME\_CHECK Framing User I/O Ports (RX)

Name	Direction	Description
RX_D[0:(wn-1)]	Input	Incoming data from channel partner (Ascending bit order).
RX_REM[0:r(n)]	Input	Specifies the number of valid bytes in the last data beat; valid only when RX_EOF_N is asserted. REM bus widths are given by [0:r(n)], where r(n) = ceiling [ $\lceil \log_2(n) \rceil - 1$ ].
RX_SRC_RDY_N	Input	Asserted (Low) when data and control signals from an Aurora 8B/10B core are valid. Deasserted (High) when data and/or control signals from an Aurora 8B/10B core should be ignored (active-Low).
RX_SOF_N	Input	Signals the start of the incoming frame (active-Low, asserted for a single USER_CLK cycle).

Table 2-6: FRAME\_CHECK Framing User I/O Ports (RX) (Cont'd)

Name	Direction	Description
RX_EOF_N	Input	Signals the end of the incoming frame (active-Low, asserted for a single USER_CLK cycle).
ERR_COUNT[0:7]	Output	Count of the number of RX data words received by the frame checker that did not match with the expected value
CHANNEL_UP	Input	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to transmit data.
USER_CLK	Input	Parallel clock shared by the Aurora 8B/10B core and the user application.
RESET	Input	Resets the Aurora 8B/10B core (active-High).

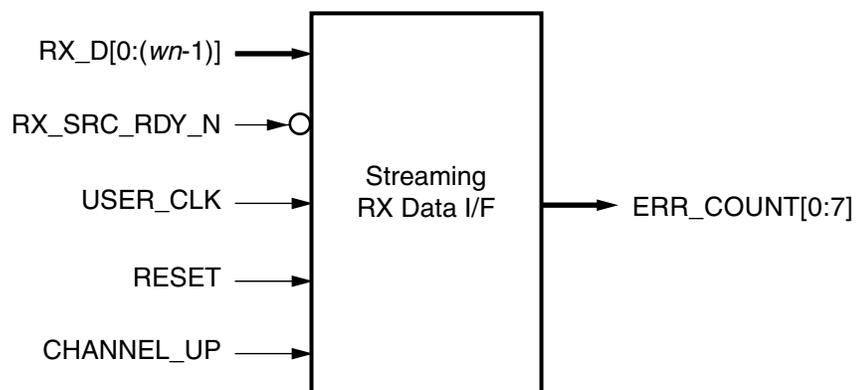
Expected frame RX data is computed by LFSR. Received user data is validated by checking against the following LocalLink protocol rules:

1. Start the frame when RX\_SOF\_N is asserted
2. RX\_REM bus is valid during RX\_EOF\_N assertion
3. RX\_SRC\_RDY\_N should be asserted for all of the valid user data

The incoming RX data through RX\_D port is registered and compared with the calculated RX data internal to FRAME\_CHECK. If the incoming RX data does not match the expected RX data, an 8-bit counter is incremented. This error counter is indicated to the user through ERR\_COUNT port. This counter stops counting and count value is latched when it reaches to 255.

## Streaming RX Data Interface

Figure 2-12 shows the FRAME\_CHECK streaming user interface of the Aurora 8B/10B core, ports for RX data.



UG353\_03\_08\_030309

Figure 2-12: Aurora 8B/10B Core Streaming RX Data Interface (FRAME\_CHECK)

Table 2-7 lists the FRAME\_CHECK streaming RX data ports and their descriptions.

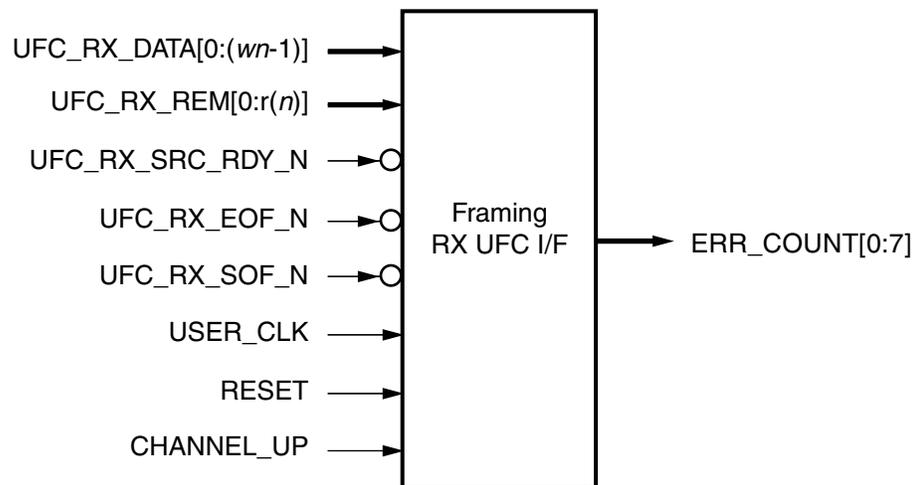
Table 2-7: FRAME\_CHECK Streaming User I/O Ports (RX)

Name	Direction	Description
RX_D[0:(wn-1)]	Input	Incoming data from channel partner (Ascending bit order).
RX_SRC_RDY_N	Input	Asserted (Low) when data and control signals from an Aurora 8B/10B core are valid. Deasserted (High) when data and/or control signals from an Aurora 8B/10B core should be ignored (active-Low)
ERR_COUNT[0:7]	Output	Count of the number of RX data words received by the frame checker that did not match with the expected value
CHANNEL_UP	Input	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data.
USER_CLK	Input	Parallel clock shared by the Aurora 8B/10B core and the user application.
RESET	Input	Resets the Aurora 8B/10B core (active-High).

In streaming mode, the incoming RX Data is compared against calculated RX data.

## Framing UFC RX Interface

Table 2-13 shows the FRAME\_CHECK framing UFC RX interface of the Aurora 8B/10B core, with LocalLink compliant ports for UFC RX data.



UG353\_03\_09\_030309

Figure 2-13: Aurora 8B/10B Core Framing UFC RX Interface (FRAME\_CHECK)

Table 2-8 lists the FRAME\_CHECK framing UFC RX data ports and their descriptions.

Table 2-8: FRAME\_CHECK Framing UFC User I/O Ports (RX)

Name	Direction	Description
UFC_RX_DATA [0:(wn-1)]	Input	Incoming UFC message data from the channel partner.
UFC_RX_REM [0:r(n)]	Input	Specifies the number of valid bytes of data presented on the UFC_RX_DATA port on the last word of a UFC message. Valid only when UFC_RX_EOF_N is asserted. $n = 16$ bytes max. REM bus widths are given by [0:r(n)], where $r(n) = \text{ceiling} [\{\log_2(n)\}-1]$ .
UFC_RX_SRC_RDY_N	Input	Asserted when the values on the UFC_RX_DATA port is valid. When this signal is not asserted, all values on the UFC_RX_DATA port should be ignored (active-Low).
UFC_RX_SOF_N	Input	Signals the start of the incoming UFC message (active-Low).
UFC_RX_EOF_N	Input	Signals the end of the incoming UFC message (active-Low).
ERR_COUNT[0:7]	Output	Count of the number of RX UFC data words received by the frame checker that did not match with the expected value
CHANNEL_UP	Input	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data.
USER_CLK	Input	Parallel clock shared by the Aurora 8B/10B core and the user application.
RESET	Input	Resets the Aurora 8B/10B core (active-High).

The expected UFC RX data is computed by LFSR. Error checking and counter logic is similar to that of [“Framing RX Data Interface,”](#) page 34. If the incoming RX UFC data does not match the expected RX UFC data, an 8-bit counter is incremented. This error counter is indicated to the user through ERR\_COUNT port.

## Framing NFC RX Interface

Figure 2-14 shows the FRAME\_CHECK framing NFC RX interface of the Aurora 8B10B core with ports.

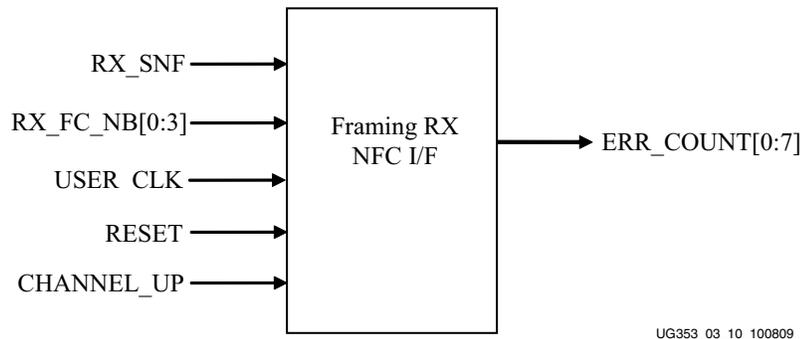


Figure 2-14: Aurora 8B10B Core Framing NFC RX Interface (FRAME\_CHECK)

Table 2-9 lists the FRAME\_CHECK framing NFC RX ports and their descriptions.

Table 2-9: Aurora 8B10B Core Framing NFC RX Interface (FRAME\_CHECK)

Name	Direction	Description
RX_SNF	Input	Signals the start of the incoming NFC message.
RX_FC_NB[0:3]	Input	Received NFC message PAUSE value
CHANNEL_UP	Input	Asserted when Aurora channel initialization is complete and channel is ready to send data.
USER_CLK	Input	Parallel clock shared by the Aurora 8B10B core and the user application.
RESET	Input	Resets the Aurora core (active-High).
ERROR_COUNT[0:7]	Output	Incremented when mismatch detected between the expected PAUSE value and the PAUSE value received by the FRAME_CHECK.

FRAME\_GEN transmits NFC message with predefined PAUSE value each time. This PAUSE value is compared with PAUSE value received by FRAME\_CHECK. If the incoming PAUSE value does not match with expected PAUSE value, an 8-bit counter will be incremented. This error counter is indicated to the user through ERR\_COUNT port.

Because cores are generated one at a time, simulating simplex cores requires additional steps (except for RX/TX Simplex cores, which can be connected to themselves). To simulate a simplex TX or simplex RX core, perform the following steps:

1. Generate the core for simulation.
2. Generate a complimentary simplex core. Go to the implement directory of the first core generated.
3. Set the environment variable SIMPLEX\_PARTNER to point to the directory for the complementary core.
4. Run the script according to the instructions in the *LogiCORE IP Aurora 8B/10B v5.1 Getting Started Guide*.

**Note:** The top-level module name of the simplex design and simplex partner design should be similar. For example, if the top-level module name of the TX simplex design is `simplex_201_tx`, then the top-level module name of the simplex partner should be `rx_simplex_201_tx`.

## Designing with the Core

This section provides a general description of how to use the Aurora 8B/10B core in your designs

### General Design Guidelines

This section describes the steps required to turn an Aurora 8B/10B core into a fully functioning design with user-application logic. It is important to note that not all implementations require all of the design steps listed here. Follow the logic design guidelines in this manual carefully.

### Use the Example Design as a Starting Point

Each instance of an Aurora 8B/10B core created by CORE Generator software is delivered with an example design that can be simulated and implemented in FPGA. This design can be used as a starting point for your own design or can be used to troubleshoot the user application, if necessary.

### Know the Degree of Difficulty

Aurora 8B/10B design is challenging to implement in any technology, and the degree of difficulty is further influenced by

- Maximum system clock frequency
- Targeted device architecture
- Nature of the user application

All Aurora 8B/10B implementations require careful attention to system performance requirements. Pipelining, logic mappings, placement constraints and logic duplications are all methods that help boost system performance.

### Keep It Registered

To simplify timing and increase system performance in an FPGA design, keep all inputs and outputs registered between the user application and the core. This means that all inputs and outputs from user application should come from, or connect to a flip-flop. Registering signals may not be possible for all paths, but doing so simplifies timing analysis and makes it easier for the Xilinx tools to place-and-route the design.

### Recognize Timing Critical Signals

The UCF provided with the example design for the core identifies the critical signals and the timing constraints that should be applied.

### Use Supported Design Flows

The core is delivered as Verilog or VHDL source code. The example implementation scripts provided currently use XST as synthesis tool for the example design that is delivered with the core. Other synthesis tools may also be used.

## Make Only Allowed Modifications

The Aurora 8B/10B core is not user modifiable. Any modifications may have adverse effects on the system timings and protocol compliance. Supported user configurations of the Aurora 8B/10B core can only be made by selecting options from CORE Generator tool.

# User Interface

---

## Introduction

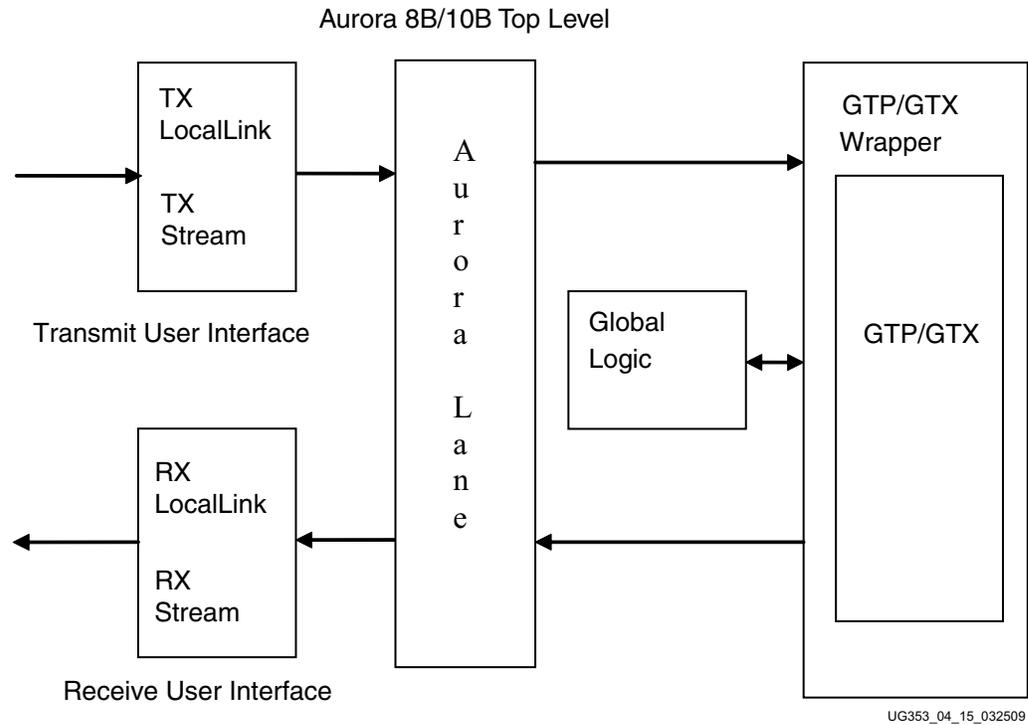
An Aurora 8B/10B core can be generated with either a *framing* or *streaming* user data interface. In addition, flow control options are available for designs with framing interfaces. See [Chapter 4, “Flow Control.”](#)

The framing user interface complies with the *LocalLink Interface Specification*. It comprises the signals necessary for transmitting and receiving framed user data. The streaming interface allows users to send data without special frame delimiters. It is simple to operate and uses fewer resources than framing.

## Top Level Architecture

Aurora 8B/10B top level (block level) file instantiates Aurora 8B/10B lane module, TX and RX LocalLink modules, global logic module, and wrapper for GTP/GTX transceiver. This top level wrapper file is instantiated in the example design file together with clock, reset circuit and frame generator and checker modules.

[Figure 3-1, page 42](#) shows Aurora 8B/10B top level for a duplex configuration. The top level file is the starting point for a user design.



*Figure 3-1: Top-Level Architecture*

The following sections describe the streaming and framing interface in details. User interface logic should be designed to comply with the timing requirement of the respective interface as explained in the subsequent sections.

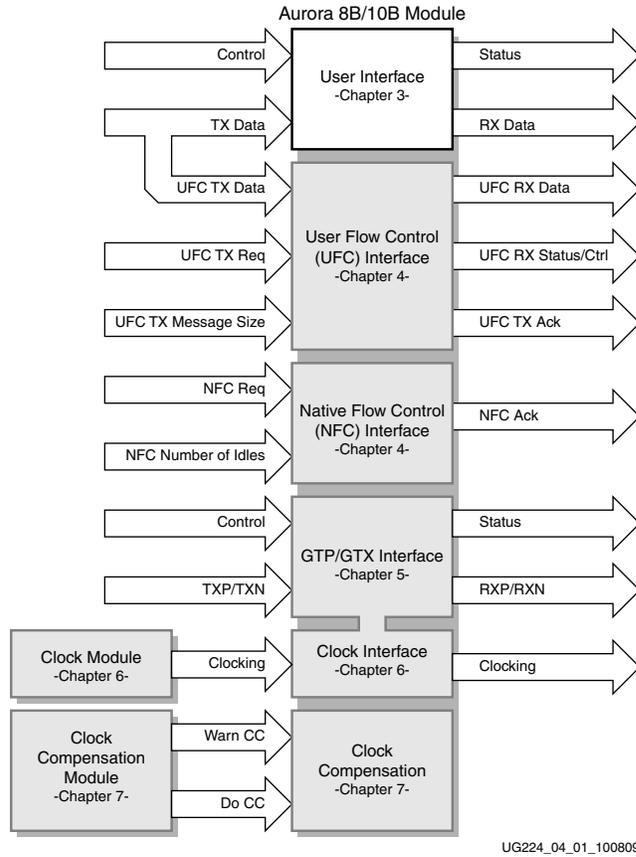


Figure 3-2: Top-Level User Interface

**Note:** The user interface signals vary depending upon the selections made when generating an Aurora 8B/10B core in the CORE Generator™ software.

## Framing Interface

Figure 3-3 shows the framing user interface of the Aurora 8B/10B core, with LocalLink-compliant ports for TX and RX data.

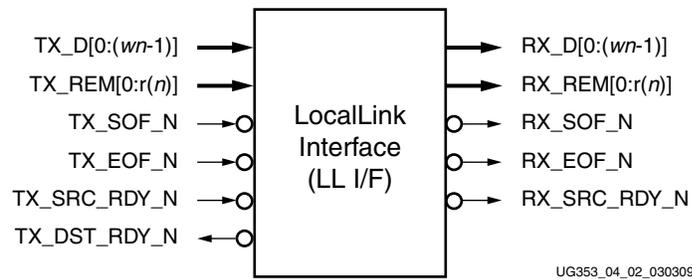


Figure 3-3: Aurora 8B/10B Core Framing Interface (LocalLink)

## LocalLink TX Ports

Table 3-1 lists port descriptions for LocalLink TX data ports. These ports are included on full-duplex, simplex TX, and simplex Both framing cores.

Table 3-1: LocalLink User I/O Ports (TX)

Name	Direction	Description
TX_D[0:(wn-1)]	Input	Outgoing data (Ascending bit order). <ul style="list-style-type: none"> <li><math>w</math> takes values 16 or 32 depending on whether width is 2-byte or 4-byte</li> <li><math>n</math> is the number of lanes</li> </ul>
TX_DST_RDY_N	Output	Asserted (Low) during clock edges when signals from the source will be accepted (if TX_SRC_RDY_N is also asserted). Deasserted (High) on clock edges when signals from the source will be ignored.
TX_EOF_N	Input	Signals the end of the frame (active-Low).
TX_REM[0:r(n)]	Input	Specifies the number of valid bytes in the last data beat; valid only while TX_EOF_N is asserted. REM bus widths are given by [0:r(n)], where $r(n) = \text{ceiling}[\{\log_2(n)\}-1]$ .
TX_SOF_N	Input	Signals the start of the outgoing channel frame (active-Low).
TX_SRC_RDY_N	Input	Asserted (Low) when LocalLink signals from the source are valid. Deasserted (High) when LocalLink control signals and/or data from the source should be ignored (active-Low).

## LocalLink RX Ports

Table 3-2 lists port descriptions for LocalLink RX data ports. These ports are included on full-duplex, simplex RX, and simplex Both framing cores.

Table 3-2: LocalLink User I/O Ports (RX)

Name	Direction	Description
RX_D[0:(wn-1)]	Output	Incoming data from channel partner (Ascending bit order).
RX_EOF_N	Output	Signals the end of the incoming frame (active-Low, asserted for a single user clock cycle). Ignored when RX_SRC_RDY_N is de-asserted (High)
RX_REM[0:r(n)]	Output	Specifies the number of valid bytes in the last data beat; valid only when RX_EOF_N is asserted. REM bus widths are given by [0:r(n)], where $r(n) = \text{ceiling}[\{\log_2(n)\}-1]$ .
RX_SOF_N	Output	Signals the start of the incoming frame (active-Low, asserted for a single user clock cycle). Ignored when RX_SRC_RDY_N is de-asserted (High)
RX_SRC_RDY_N	Output	Asserted (Low) when data and control signals from an Aurora 8B/10B core are valid. Deasserted (High) when data and/or control signals from an Aurora 8B/10B core should be ignored (active-Low).

To transmit data, the user manipulates control signals to cause the core to do the following:

- Take data from the user on the TX\_D bus
- Encapsulate and stripe the data across lanes in the Aurora 8B/10B channel (TX\_SOF\_N, TX\_EOF\_N)
- Pause data (that is, insert idles) (TX\_SRC\_RDY\_N)

When the core receives data, it does the following:

- Detects and discards control bytes (idles, clock compensation, SCP, ECP)
- Asserts framing signals (RX\_SOF\_N, RX\_EOF\_N)
- Recovers data from the lanes
- Assembles data for presentation to the user on the RX\_D bus

## LocalLink Bit Ordering

The *LocalLink Interface Specification* allows both ascending and descending bit ordering. Aurora 8B/10B cores use ascending ordering. They transmit and receive the most significant bit of the most significant byte first. Figure 3-4 shows the organization of an  $n$ -byte example of the LocalLink data interfaces of an Aurora 8B/10B core.

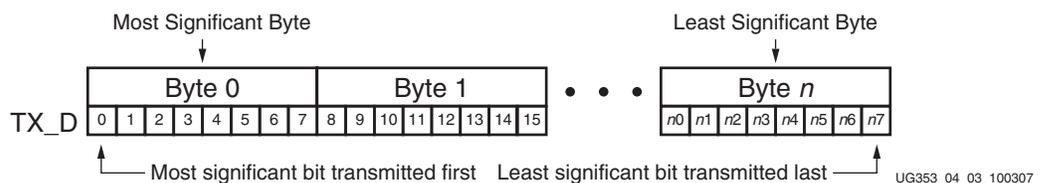


Figure 3-4: LocalLink Interface Bit Ordering

UG353\_04\_03\_100307

## Transmitting Data

LocalLink is a synchronous interface. The Aurora 8B/10B core samples the data on the interface only on the positive edge of USER\_CLK, and only on the cycles when both TX\_DST\_RDY\_N and TX\_SRC\_RDY\_N are asserted (Low).

When LocalLink signals are sampled, they are only considered valid if TX\_SRC\_RDY\_N is asserted. The user application can deassert TX\_SRC\_RDY\_N on any clock cycle; this will cause Aurora 8B/10B to ignore the LocalLink input for that cycle. If this occurs in the middle of a frame, idle symbols are sent through the Aurora 8B/10B channel, which eventually result in a idle cycles during the frame when it is received at the RX user interface.

LocalLink data is only valid when it is framed. Data outside of a frame is ignored. To start a frame, assert TX\_SOF\_N while the first word of data is on the TX\_D port. To end a frame, assert TX\_EOF\_N while the last word (or partial word) of data is on the TX\_D port.

**Note:** In the case of frames that are a single word long or less, TX\_SOF\_N and TX\_EOF\_N are asserted simultaneously.

## Data Remainder

LocalLink allows the last word of a frame to be a partial word. This lets a frame contain any number of bytes, regardless of the word size. The TX\_REM bus is used to indicate the number of valid bytes in the final word of the frame. The bus is only used when TX\_EOF\_N is asserted. Aurora 8B/10B uses encoded REM values. REM is the binary encoding of the number of valid bytes minus 1. A zero REM value indicates the left-most byte in the TX\_D port (the MSB) is the only valid byte in the word. Table 3-3 shows the mapping between TX\_REM values and valid byte counts for the TX\_D port.

Table 3-3: TX Data Remainder Values

TX_REM Value	Number of Valid Bytes
0	1
1	2
2	3
3	4
⋮	⋮
⋮	⋮
⋮	⋮
$n$	$n+1$

## Aurora 8B/10B Frames

The TX\_LL submodule translates each user frame that it receives through the TX interface to an Aurora 8B/10B frame. The 2-byte SCP code group is added to the beginning of the frame data to indicate the start of frame, and a 2-byte ECP set is sent after the frame ends to indicate the end of frame. Idle code groups are inserted whenever data is not available. Code groups are 8B/10B encoded byte pairs. All data in Aurora 8B/10B is sent as code groups, so user frames with an odd number of bytes have a control character called PAD appended to the end of the frame to fill out the final code group. Table 3-4 shows a typical Aurora 8B/10B frame with an even number of data bytes.

### Length

The user controls the channel frame length by manipulation of the TX\_SOF\_N and TX\_EOF\_N signals. The Aurora 8B/10B core responds with start-of-frame and end-of-frame ordered sets, /SCP/ and /ECP/ respectively, as shown in Table 3-4.

Table 3-4: Typical Channel Frame

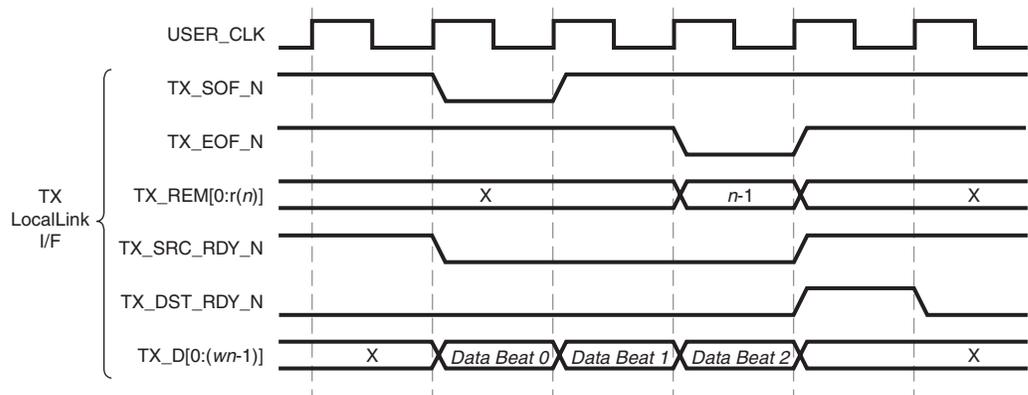
/SCP/ <sub>1</sub>	/SCP/ <sub>2</sub>	Data Byte 0	Data Byte 1	Data Byte 2	...	Data Byte <i>n</i> -1	Data Byte <i>n</i>	/ECP/ <sub>1</sub>	/ECP/ <sub>2</sub>
--------------------	--------------------	----------------	----------------	----------------	-----	--------------------------	-----------------------	--------------------	--------------------

### Example A: Simple Data Transfer

Figure 3-5 shows an example of a simple data transfer on a LocalLink interface that is *n*-bytes wide. In this case, the amount of data being sent is *3n* bytes and so requires three data beats. TX\_DST\_RDY\_N is asserted, indicating that the LocalLink interface is already ready to transmit data. When the Aurora 8B/10B core is not sending data, it sends idle sequences.

To begin the data transfer, the user asserts the TX\_SOF\_N concurrently with TX\_SRC\_RDY\_N and the first *n* bytes of the user frame. Since TX\_DST\_RDY\_N is already asserted, data transfer begins on the next clock edge. An /SCP/ ordered set is placed on the first two bytes of the channel to indicate the start of the frame. Then the first *n*-2 data bytes are placed on the channel. Because of the offset required for the /SCP/, the last two bytes in each data beat are always delayed one cycle and transmitted on the first two bytes of the next beat of the channel.

To end the data transfer, the user asserts TX\_EOF\_N, TX\_SRC\_RDY\_N, the last data bytes, and the appropriate value on the TX\_REM bus. In this example, TX\_REM is set to *n*-1 to indicate that all bytes are valid in the last data beat. One clock cycle after TX\_EOF\_N is asserted, the LocalLink interface deasserts TX\_DST\_RDY\_N and uses the gap in the data flow to send the final offset data bytes and the /ECP/ ordered set, indicating the end of the frame. TX\_DST\_RDY\_N is reasserted on the next cycle so that more data transfers can continue. As long as there is no new data, the Aurora 8B/10B core sends idles.



UG353\_04\_04\_100809

Figure 3-5: Simple Data Transfer

### Example B: Data Transfer with Pad

Figure 3-6 shows an example of a  $(3n-1)$ -byte data transfer that requires the use of a pad. Since there is an odd number of data bytes, the Aurora 8B/10B core appends a pad character at the end of the Aurora 8B/10B frame, as required by the protocol. A transfer of  $3n-1$  data bytes requires two full  $n$ -byte data words and one partial data word. In this example, TX\_REM is set to  $n-2$  to indicate  $n-1$  valid bytes in the last data word.

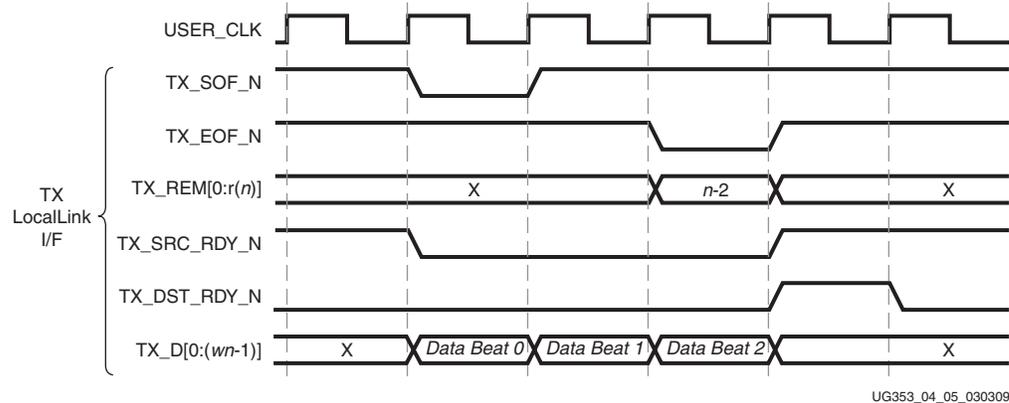


Figure 3-6: Data Transfer with Pad

### Example C: Data Transfer with Pause

Figure 3-7 shows how a user can pause data transmission during a frame transfer. In this example, the user is sending  $3n$  bytes of data, and pauses the data flow after the first  $n$  bytes. After the first data word, the user deasserts TX\_SRC\_RDY\_N, causing the TX Aurora 8B/10B core to ignore all data on the bus and transmit idles instead. The offset data from the first data word in the previous cycle still is transmitted on lane 0, but the next data word is replaced by idle characters. The pause continues until TX\_SRC\_RDY\_N is deasserted.

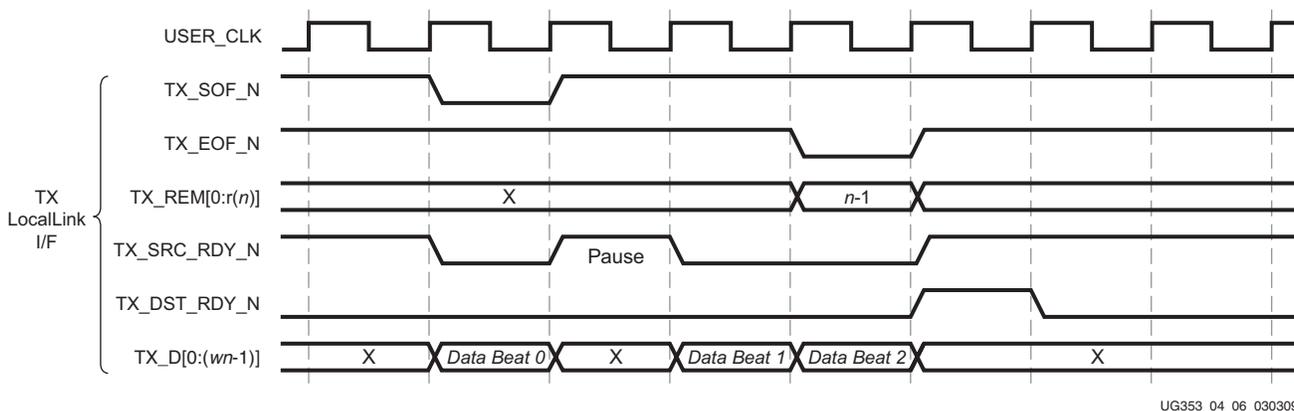


Figure 3-7: Data Transfer with Pause

### Example D: Data Transfer with Clock Compensation

The Aurora 8B/10B core automatically interrupts data transmission when it sends clock compensation sequences. The clock compensation sequence imposes 12 bytes of overhead per lane every 10,000 bytes.

Figure 3-8 shows how the Aurora 8B/10B core pauses data transmission during the clock compensation<sup>(1)</sup> sequence.

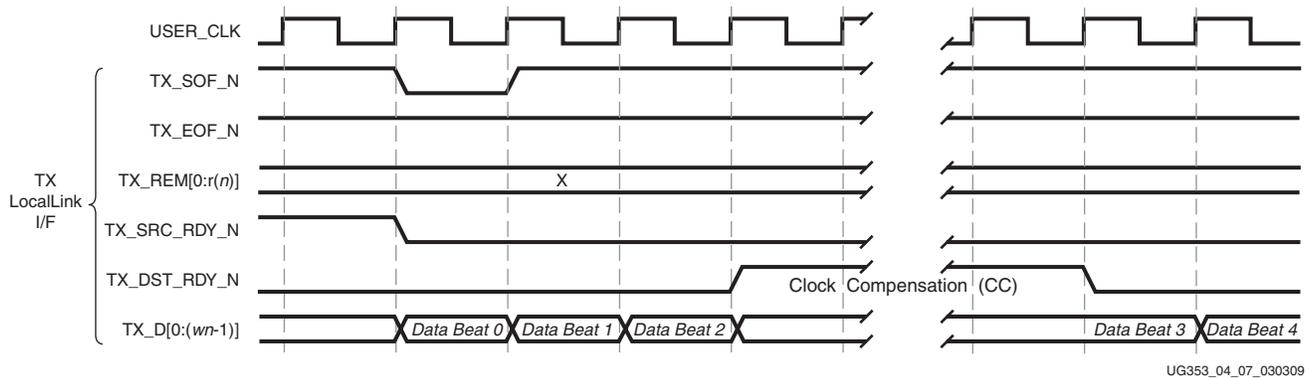


Figure 3-8: Data Transfer Paused by Clock Compensation

### TX Interface Example

This section illustrates a simple example of how a user might design an interface between the user's transmit FIFO and the LocalLink interface of an Aurora 8B/10B core.

To review, in order to transmit data, the user asserts TX\_SOF\_N and TX\_SRC\_RDY\_N. TX\_DST\_RDY\_N indicates that the data on the TX\_D bus will be transmitted on the next rising edge of the clock, assuming TX\_SRC\_RDY\_N remains asserted.

Figure 3-9 is a diagram of a typical connection between an Aurora 8B/10B core and the user's data source (in this example, a FIFO), including the simple logic needed to generate TX\_SOF\_N, TX\_SRC\_RDY\_N, and TX\_EOF\_N from typical FIFO buffer status signals. While RESET is false, the example application waits for a FIFO to fill and then it generates the TX\_SOF\_N and TX\_SRC\_RDY\_N signals. These two signals cause the Aurora 8B/10B core to start reading the FIFO by asserting the TX\_DST\_RDY\_N signal.

The Aurora 8B/10B core encapsulates the FIFO data and transmits it until the FIFO is empty. At this point, the example application tells the Aurora 8B/10B core to end the transmission using the TX\_EOF\_N signal.

1. Because of the need for clock compensation every 10,000 bytes per lane (5,000 clocks for 2-byte per lane designs; 2,500 clocks for 4-byte per lane designs), a user cannot continuously transmit data nor can data be continuously received. During clock compensation, data transfer is suspended for six clock periods.

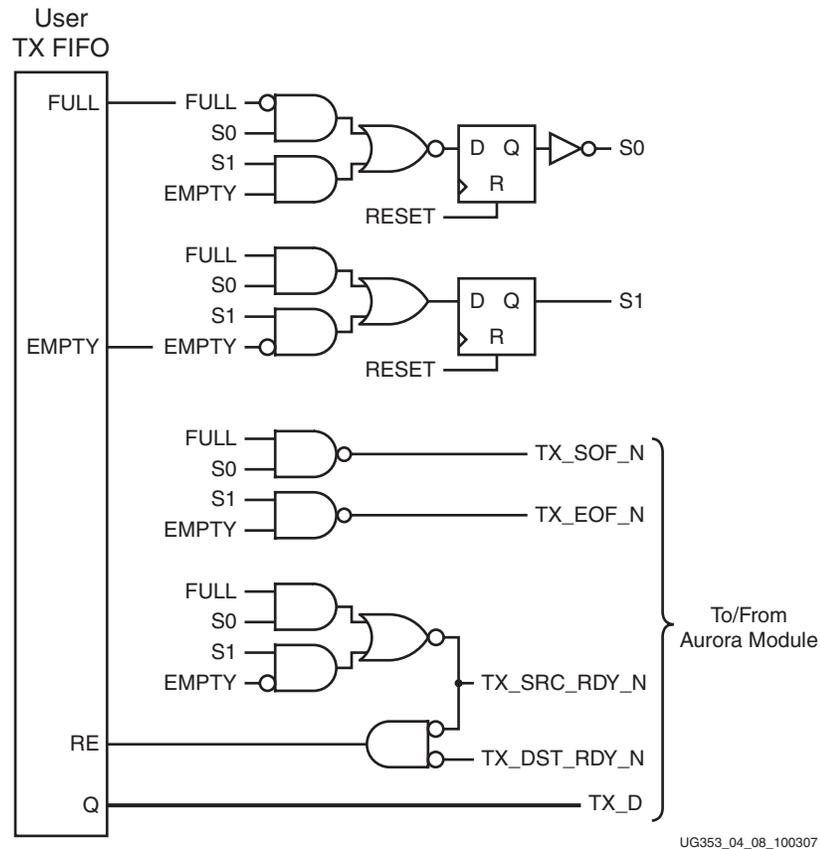


Figure 3-9: Transmitting Data

## Receiving Data

When the Aurora 8B/10B core receives an Aurora 8B/10B frame, it presents it to the user through the RX LocalLink interface after discarding the framing characters, idles, and clock compensation sequences.

The RX\_LL submodule has no built in elastic buffer for user data. As a result, there is no RX\_DST\_RDY\_N signal on the RX LocalLink interface. The only way for the user application to control the flow of data from an Aurora 8B/10B channel is to use one of the core's optional flow control features. In most cases, a FIFO should be added to the RX data path to ensure no data is lost while flow control messages are in transit.

The Aurora 8B/10B core asserts the RX\_SRC\_RDY\_N signal when the signals on its RX LocalLink interface are valid. Applications should ignore any values on the RX LocalLink ports sampled while RX\_SRC\_RDY\_N is deasserted (High).

RX\_SOF\_N is asserted concurrently with the first word of each frame from the Aurora 8B/10B core. RX\_EOF\_N is asserted concurrently with the last word or partial word of each frame. The RX\_REM port indicates the number of valid bytes in the final word of each frame. It uses the same encoding as TX\_REM and is only valid when RX\_EOF\_N is asserted.

The Aurora 8B/10B core can deassert RX\_SRC\_RDY\_N anytime, even during a frame. The timing of the RX\_SRC\_RDY\_N deassertions is independent of the way the data was transmitted. The core can occasionally deassert RX\_SRC\_RDY\_N even if the frame was

originally transmitted without pauses. These pauses are a result of the framing character stripping and left alignment process, as the core attempts to process each frame with as little latency as possible.

“Example A: Data Reception with Pause” shows the reception of a typical Aurora 8B/10B frame.

### Example A: Data Reception with Pause

Figure 3-10 shows an example of  $3n$  bytes of received data interrupted by a pause. Data is presented on the RX\_D bus. When the first  $n$  bytes are placed on the bus, the RX\_SOF\_N and RX\_SRC\_RDY\_N outputs are asserted to indicate that data is ready for the user. On the clock cycle following the first data beat, the core deasserts RX\_SRC\_RDY\_N, indicating to the user that there is a pause in the data flow.

After the pause, the core asserts RX\_SRC\_RDY\_N and continues to assemble the remaining data on the RX\_D bus. At the end of the frame, the core asserts RX\_EOF\_N. The core also computes the value of RX\_REM bus and presents it to the user based on the total number of valid bytes in the final word of the frame.

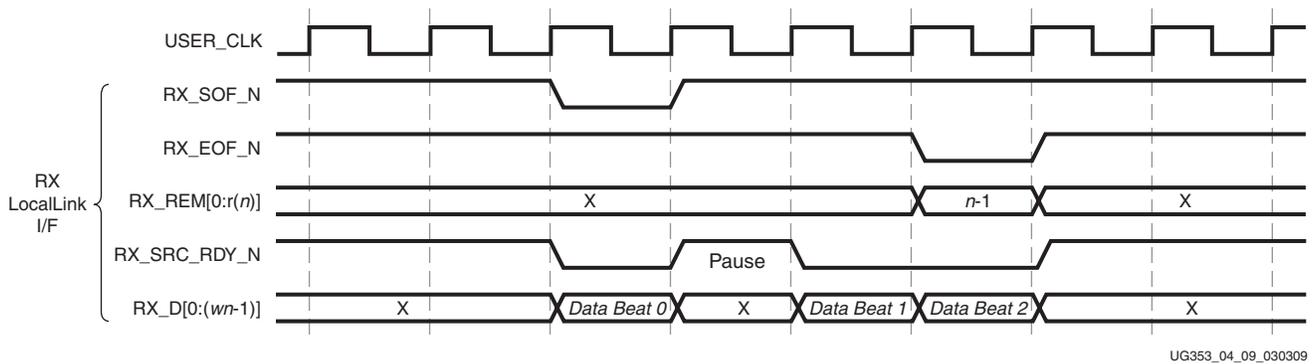


Figure 3-10: Data Reception with Pause

### RX Interface Example

Figure 3-11 is a simple example of how a user might design an interface between the LocalLink interface of an Aurora 8B/10B core and a FIFO. To receive data, the user monitors the RX\_SRC\_RDY\_N signal. When valid data is present on the RX\_D port, RX\_SRC\_RDY\_N is asserted. Because the inverse of RX\_SRC\_RDY\_N is connected to the FIFO’s WE port, the data and framing signals and REM value are written to the FIFO.

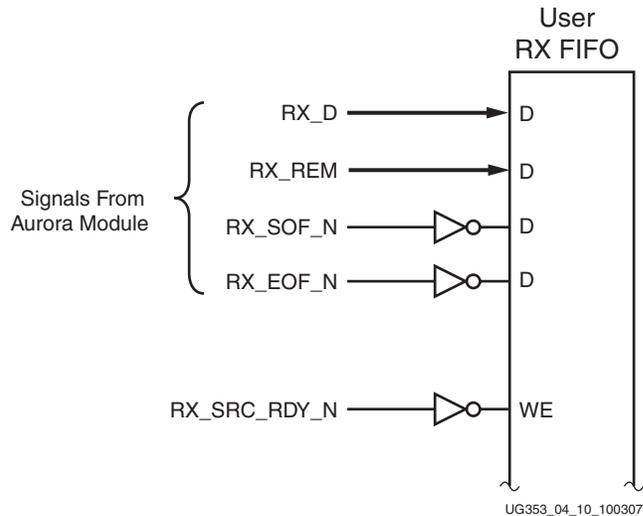


Figure 3-11: Receiving Data

## Framing Efficiency

There are two factors that affect framing efficiency in the Aurora 8B/10B core:

- Size of the frame
- Width of the data path

The CC sequence, which uses 12 bytes on every lane every 10,000 bytes, consumes about 0.12% of the total channel bandwidth.

All bytes in Aurora 8B/10B are sent in 2-byte code groups. Aurora 8B/10B frames with an even number of bytes have four bytes of overhead, two bytes for SCP (start of frame) and two bytes for ECP (end of frame). Aurora 8B/10B frames with an odd number of bytes have five bytes of overhead, four bytes of framing overhead plus an additional byte for the pad byte that is sent to fill the second byte of the code group carrying the last byte of data in the frame.

Like many parallel interfaces, LocalLink processes data from only one frame at a time. The core must drop data when it arrives on the GTP/GTX transceiver interface at the same time as data from a previous cycle. The *LocalLink Interface Specification* includes advanced options for handling multiple frames on a single cycle, but these options are not implemented in this core.

The core transmits frame delimiters only in specific lanes of the channel. SCP is only transmitted in the left-most (most-significant) lane, and ECP is only transmitted in the right-most (least-significant) lane. Any space in the channel between the last code group with data and the ECP code group is padded with idles. The result is reduced resource cost for the design, at the expense of a minimal additional throughput cost. Though SCP and ECP could be optimized for additional throughput, the single frame per cycle limitation imposed by the user interface would make this improvement unusable in most cases.

Use the formula shown in [Figure 3-12](#) to calculate the efficiency for a design of any number of lanes, any width of interface, and frames of any number of bytes.

**Note:** This formula includes the overhead for clock compensation.

$$E = \frac{100n}{n + 4 + 0.5 + \text{IDLEs} + \frac{12n}{9,988}}$$

Where:

- E = The average efficiency of a specified PDU
- n = Number of user data bytes
- 12n/9,988 = Clock correction overhead
- 4 = The overhead of SCP + ECP
- 0.5 = Average PAD overhead
- IDLEs = The overhead for IDLEs = (w/2)-1  
(w = The interface width)

UG353\_04\_11\_100307

**Figure 3-12: Formula for Calculating Overhead**

## Example

Table 3-5 is an example calculated from the formula given in Figure 3-12. It shows the efficiency for an 8-byte, 4-lane channel and illustrates that the efficiency increases as the length of channel frames increases.

**Table 3-5: Efficiency Example**

User Data Bytes	Efficiency
100	92.92%
1,000	99.14%
10,000	99.81%

Table 3-6 shows the overhead in an 8-byte, 4-lane channel when transmitting 256 bytes of frame data across the four lanes. The resulting data unit is 264 bytes long due to start and end characters, and due to the idles necessary to fill out the lanes. This amounts to 3.03% of overhead in the transmitter. In addition, a 12-byte clock compensation sequence occurs on each lane every 10,000 bytes, which adds a small amount more to the overhead. The receiver can handle a slightly more efficient data stream because it does not require any idle pattern.

**Table 3-6: Typical Overhead for Transmitting 256 Data Bytes**

Lane	Clock	Function	Character or Data Byte	
			Byte 1	Byte 2
0	1	Start of channel frame	/SCP/ <sub>1</sub>	/SCP/ <sub>2</sub>
1	1	Channel frame data	D0	D1
2	1	Channel frame data	D2	D3
3	1	Channel frame data	D4	D5
		.		
		.		
		.		
0	33	Channel frame data	D254	D255
1	33	Transmit idles	/I/	/I/
2	33	Transmit idles	/I/	/I/
3	33	End of channel frame	/ECP/ <sub>1</sub>	/ECP/ <sub>2</sub>

Table 3-7 shows the overhead that occurs with each value of TX\_REM.

Table 3-7: TX\_REM Value and Corresponding Bytes of Overhead

TX_REM Bus Value	SCP	Pad	ECP	Idles	Total
0	2	1	2	6	11
1		0			10
2		1		4	9
3		0			8
4		1		2	7
5		0			6
6		1		0	5
7		0			4

## Streaming Interface

Figure 3-13 shows an example of an Aurora 8B/10B core configured with a streaming user interface.

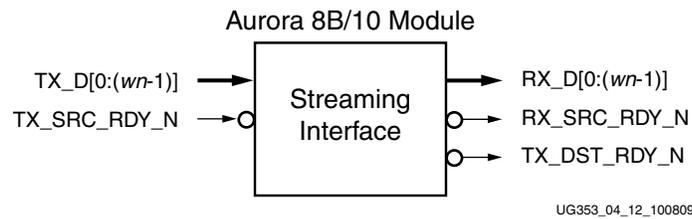


Figure 3-13: Aurora 8B/10B Core Streaming User Interface

## Streaming TX Ports

Table 3-8 lists the streaming TX data ports. These ports are included on full-duplex, simplex TX, and simplex Both framing cores.

Table 3-8: Streaming User I/O Ports (TX)

Name	Direction	Description
TX_D[0:(wn-1)]	Input	Outgoing data (Ascending bit order).
TX_DST_RDY_N	Output	Asserted (Low) during clock edges when signals from the source will be accepted (if TX_SRC_RDY_N is also asserted). Deasserted (High) on clock edges when signals from the source will be ignored.
TX_SRC_RDY_N	Input	Asserted (Low) when LocalLink signals from the source are valid. Deasserted (High) when LocalLink control signals and/or data from the source should be ignored (active-Low).

## Streaming RX Ports

Table 3-9 lists the streaming RX data ports. These ports are included on full-duplex, simplex RX, and simplex Both framing cores.

Table 3-9: Streaming User I/O Ports (RX)

Name	Direction	Description
RX_D[0:(wn-1)]	Output	Incoming data from channel partner (Ascending bit order).
RX_SRC_RDY_N	Output	Asserted (Low) when data and control signals from an Aurora 8B/10B core are valid. Deasserted (High) when data and/or control signals from an Aurora 8B/10B core should be ignored (active-Low).

## Transmitting and Receiving Data

The streaming interface allows the Aurora 8B/10B channel to be used as a pipe. Words written into the TX side of the channel are delivered, in order after some latency, to the RX side. After initialization, the channel is always available for writing, except when the DO\_CC signal is asserted to send clock compensation sequences. Applications transmit data through the TX\_D port, and use the TX\_SRC\_RDY\_N port to indicate when the data is valid (asserted Low). The Aurora 8B/10B core will deassert TX\_DST\_RDY\_N (High) when the channel is not ready to receive data. Otherwise, TX\_DST\_RDY\_N will remain asserted.

When TX\_SRC\_RDY\_N is deasserted, gaps are created between words. These gaps are preserved, except when clock compensation sequences are being transmitted. Clock compensation sequences are replicated or deleted by the GTP/GTX transceiver to make up for frequency differences between the two sides of the Aurora 8B/10B channel. As a result, gaps created when DO\_CC is asserted can shrink and grow. For details on the DO\_CC signal, see [Chapter 7, “Clock Compensation.”](#)

When data arrives at the RX side of the Aurora 8B/10B channel it is presented on the RX\_D bus and RX\_SRC\_RDY is asserted. The data must be read immediately or it is lost. If this is unacceptable, a buffer must be connected to the RX interface to hold the data until it can be used.

Figure 3-14 shows a typical example of streaming data. The example begins with neither of the ready signals asserted, indicating that both the user logic and the Aurora 8B/10B core are not ready to transfer data. During the next clock cycle, the Aurora 8B/10B core indicates that it is ready to transfer data by asserting TX\_DST\_RDY\_N. One cycle later, the user logic indicates that it is ready to transfer data by asserting the TX\_D bus and the TX\_SRC\_RDY\_N signal. Because both ready signals are now asserted, data D0 is transferred from the user logic to the Aurora 8B/10B core. Data D1 is transferred on the following clock cycle. In this example, the Aurora 8B/10B core deasserts its ready signal, TX\_DST\_RDY\_N, and no data is transferred until the next clock cycle when, once again, the TX\_DST\_RDY\_N signal is asserted. Then the user deasserts TX\_SRC\_RDY\_N on the next clock cycle, and no data is transferred until both ready signals are asserted.

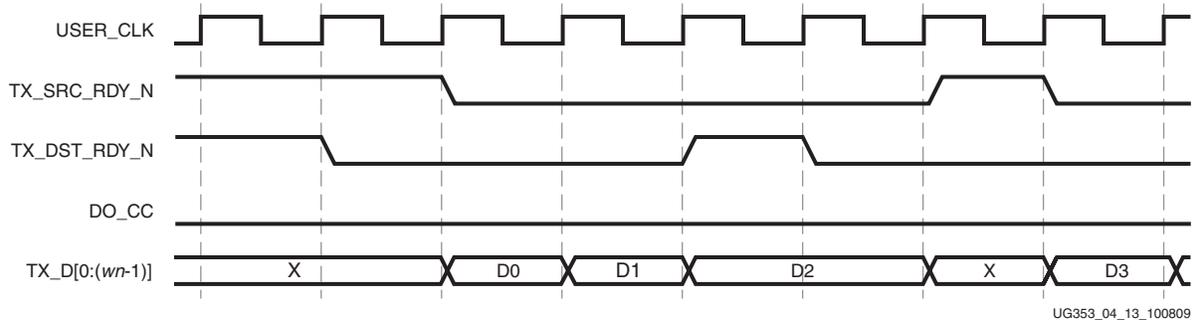


Figure 3-14: Typical Streaming Data Transfer

Figure 3-15 shows the receiving end of the data transfer that is shown in Figure 3-14.

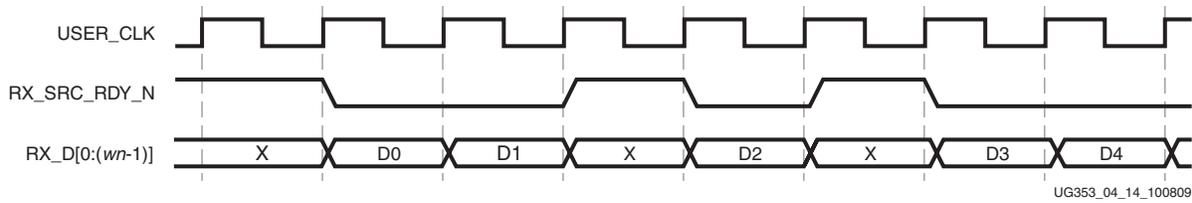


Figure 3-15: Typical Data Reception



# Flow Control

## Introduction

This chapter explains how to use Aurora 8B/10B flow control. Two flow control interfaces are available as options on cores that use a framing interface. *Native flow control* (NFC) is used for regulating the data transmission rate at the receiving end a full-duplex channel. *User flow control* (UFC) is used to accommodate high priority messages for control operations.

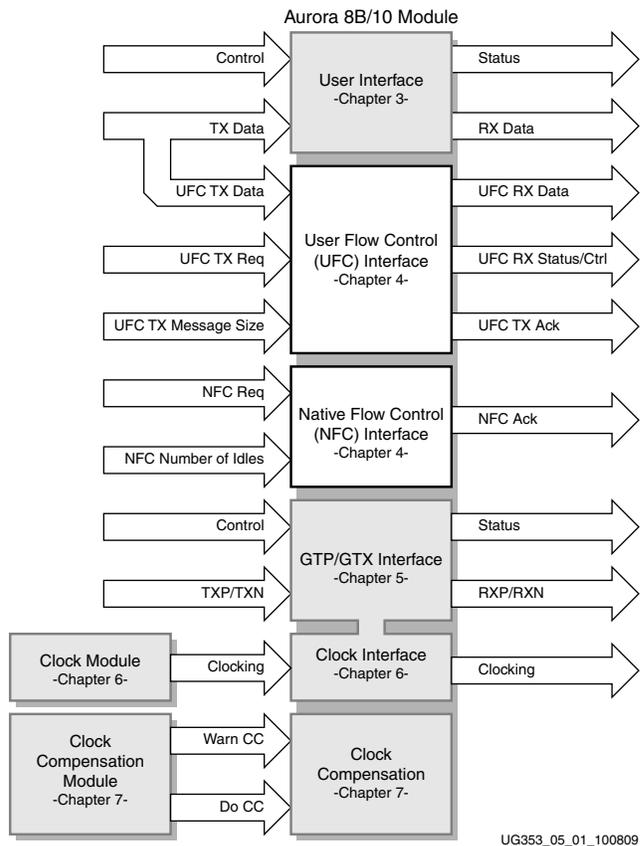


Figure 4-1: Top-Level Flow Control

## Native Flow Control

Table 4-1 shows the codes for native flow control (NFC).

Table 4-1: NFC Codes

NFC_NB	Idle Cycles Requested
0000	0 (XON)
0001	2
0010	4
0011	8
0100	16
0101	32
0110	64
0111	128
1000	256
1001 to 1110	Reserved
1111	Infinite (XOFF)

Table 4-2 lists the ports for the NFC interface available only in full-duplex Aurora 8B/10B cores.

Table 4-2: NFC I/O Ports

Name	Direction	Description
NFC_ACK_N	Output	Asserted when an Aurora 8B/10B core accepts an NFC request (active-Low).
NFC_NB[0:3]	Input	Indicates the number of PAUSE idles the channel partner must send when it receives the NFC message. Must be held until NFC_ACK_N is asserted.
NFC_REQ_N	Input	Asserted to request an NFC message be sent to the channel partner (active-Low). Must be held until NFC_ACK_N is asserted.

The Aurora 8B/10B protocol includes native flow control (NFC) to allow receivers to control the rate at which data is sent to them by specifying a number of idle data beats that must be placed into the data stream. The data flow can even be turned off completely by requesting that the transmitter temporarily send only idles (XOFF). NFC is typically used to prevent FIFO overflow conditions. For detailed explanation of NFC operation and NFC codes, see the *Aurora 8B/10B Protocol Specification*.

To send an NFC message to a channel partner, the user application asserts NFC\_REQ\_N and writes an NFC code to NFC\_NB. The NFC code indicates the minimum number of idle cycles the channel partner should insert in its TX data stream. The user application must hold NFC\_REQ\_N and NFC\_NB until NFC\_ACK\_N is asserted on a positive USER\_CLK edge, indicating the Aurora 8B/10B core will transmit the NFC message. Aurora 8B/10B cores cannot transmit data while sending NFC messages. TX\_DST\_RDY\_N is always deasserted on the cycle following an NFC\_ACK\_N assertion.

### Example A: Transmitting an NFC Message

Figure 4-2 shows an example of the transmit timing when the user sends an NFC message to a channel partner.

**Note:** TX\_DST\_RDY\_N is deasserted for one cycle<sup>(1)</sup> to create the gap in the data flow in which the NFC message is placed.

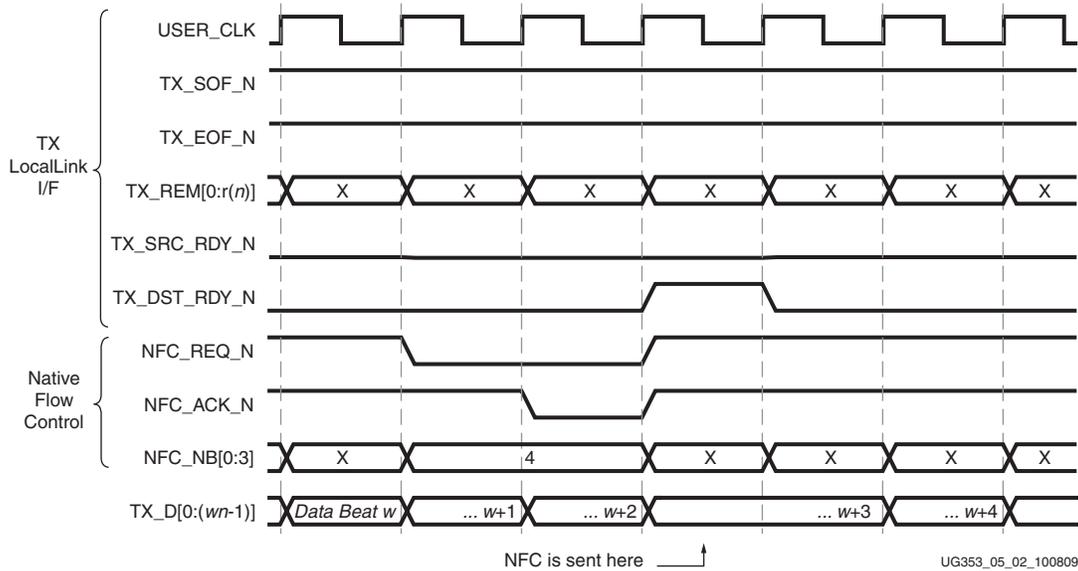


Figure 4-2: Transmitting an NFC Message

### Example B: Receiving a Message with NFC Idles Inserted

Figure 4-3 shows an example of the signals on the TX user interface when an NFC message is received. In this case, the NFC message has a code of 0001, requesting two data beats of idles. The core deasserts TX\_DST\_RDY\_N on the user interface until enough idles have been sent to satisfy the request. In this example, the core is operating in immediate NFC mode. Aurora 8B/10B cores can also operate in completion mode, where NFC idles are only inserted between frames. If a completion mode core receives an NFC message while it is transmitting a frame, it finishes transmitting the frame before deasserting TX\_DST\_RDY\_N to insert idles.

1. Assumes that  $n$  is at least 2.

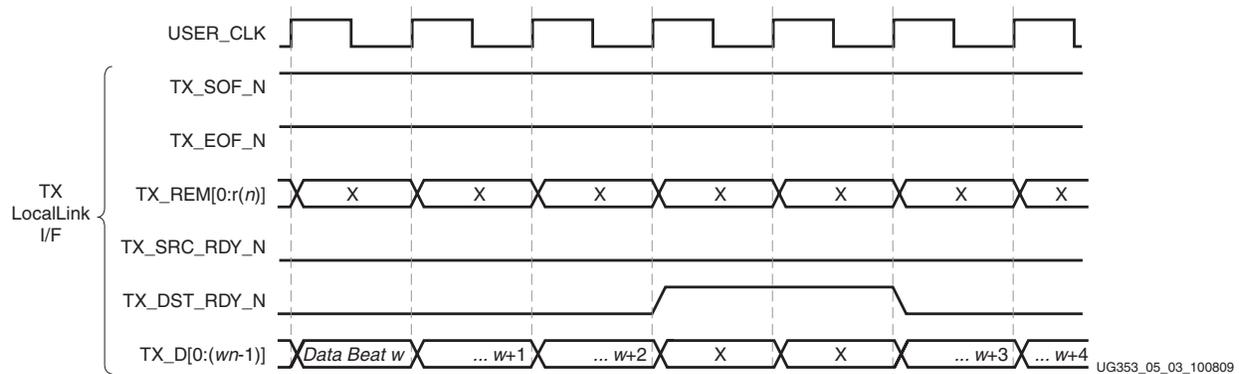


Figure 4-3: Transmitting a Message with NFC Idles Inserted

## User Flow Control

The Aurora 8B/10B protocol includes user flow control (UFC) to allow channel partners to send control information using a separate in-band channel. The user can send short UFC messages to the core's channel partner without waiting for the end of a frame in progress. The UFC message shares the channel with regular frame data, but has a higher priority.

Table 4-3 describes the ports for the UFC interface.

Table 4-3: UFC I/O Ports

Name	Direction	Description
UFC_TX_REQ_N	Input	Asserted to request a UFC message be sent to the channel partner (active-Low). Must be held until UFC_TX_ACK_N is asserted. Do not assert this signal unless the entire UFC message is ready to be sent; a UFC message cannot be interrupted once it has started.
UFC_TX_MS[0:2]	Input	Specifies the size of the UFC message that will be sent. The SIZE encoding is a value between 0 and 7. See Table 4-4, page 63.
UFC_TX_ACK_N	Output	Asserted when an Aurora 8B/10B core is ready to read the contents of the UFC message (active-Low). On the cycle after the ACK signal is asserted, data on the TX_D port will be treated as UFC data. TX_D data continues to be used to fill the UFC message until enough cycles have passed to send the complete message. Unused bytes from a UFC cycle are discarded.
UFC_RX_DATA[0:(8n-1)]	Output	Incoming UFC message data from the channel partner ( $n = 16$ bytes max).
UFC_RX_SRC_RDY_N	Output	Asserted when the values on the UFC_RX ports are valid. When this signal is not asserted, all values on the UFC_RX ports should be ignored (active-Low).
UFC_RX_SOF_N	Output	Signals the start of the incoming UFC message (active-Low).
UFC_RX_EOF_N	Output	Signals the end of the incoming UFC message (active-Low).
UFC_RX_REM[0:r(n)]	Output	Specifies the number of valid bytes of data presented on the UFC_RX_DATA port on the last word of a UFC message. Valid only when UFC_RX_EOF_N is asserted. $n = 16$ bytes max. REM bus widths are given by $[0:r(n)]$ , where $r(n) = \text{ceiling}[\{\log_2(n)\}-1]$ .

## Transmitting UFC Messages

UFC messages can carry an even number of data bytes from 2 to 16. The user application specifies the length of the message by driving a SIZE code on the UFC\_TX\_MS port. Table 4-4 shows the legal SIZE code values for UFC.

Table 4-4: SIZE Encoding

SIZE Field Contents	UFC Message Size
000	2 bytes
001	4 bytes
010	6 bytes
011	8 bytes
100	10 bytes
101	12 bytes
110	14 bytes
111	16 bytes

To send a UFC message, the user application asserts UFC\_TX\_REQ\_N while driving the UFC\_TX\_MS port with the desired SIZE code. UFC\_TX\_REQ\_N must be held until the Aurora 8B/10B core asserts the UFC\_TX\_ACK\_N signal, indicating that the core is ready to send the UFC message. The data for the UFC message must be placed on the TX\_D port of the data interface, starting on the first cycle after UFC\_TX\_ACK\_N is asserted. The core deasserts TX\_DST\_RDY\_N while the TX\_D port is being used for UFC data.

Figure 4-4 shows a useful circuit for switching TX\_D from sending regular data to UFC data.

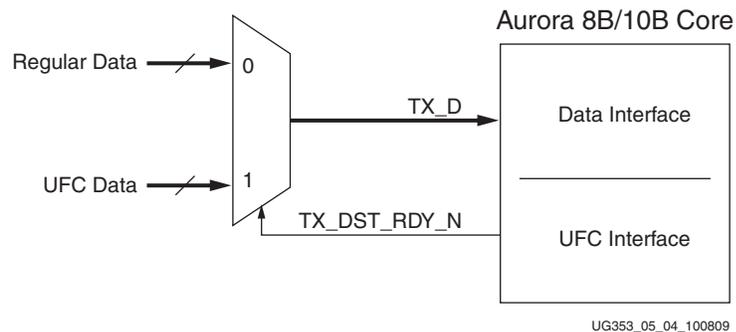


Figure 4-4: Data Switching Circuit

Table 4-5 shows the number of cycles required to transmit UFC messages of different sizes based on the width of the LocalLink data interface. UFC messages should never be started until all message data is available. Unlike regular data, UFC messages cannot be interrupted after UFC\_TX\_ACK\_N has been asserted.

Table 4-5: Number of Data Beats Required to Transmit UFC Messages

UFC Message	UFC_TX_MS Value	LL I/F Width	Number of Data Beats	LL I/F Width	Number of Data Beats
2 Bytes	0	2 Bytes	1	10 Bytes	1
4 Bytes	1		2		
6 Bytes	2		3		
8 Bytes	3		4		
10 Bytes	4		5		2
12 Bytes	5		6		
14 Bytes	6		7		
16 Bytes	7		8		
2 Bytes	0	4 Bytes	1	12 Bytes	1
4 Bytes	1		2		
6 Bytes	2		3		
8 Bytes	3		4		
10 Bytes	4		2		
12 Bytes	5				
14 Bytes	6				
16 Bytes	7				
2 Bytes	0	6 Bytes	1	14 Bytes	1
4 Bytes	1		2		
6 Bytes	2				
8 Bytes	3				
10 Bytes	4		3		
12 Bytes	5				
14 Bytes	6				
16 Bytes	7				

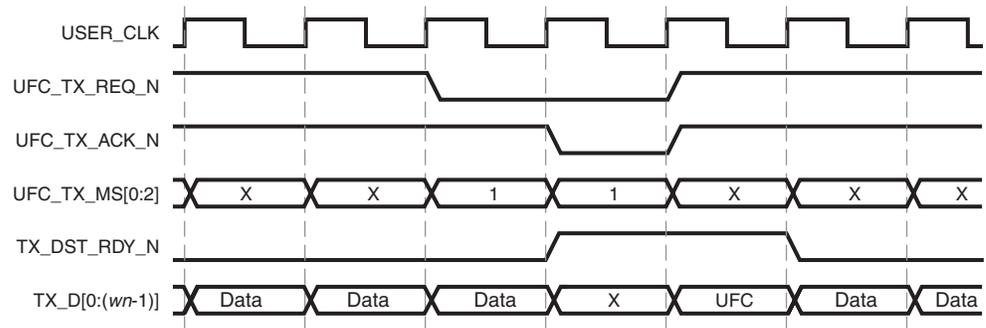
Table 4-5: Number of Data Beats Required to Transmit UFC Messages (Cont'd)

UFC Message	UFC_TX_MS Value	LL I/F Width	Number of Data Beats	LL I/F Width	Number of Data Beats
2 Bytes	0	8 Bytes	1	16 Bytes or more	1
4 Bytes	1				
6 Bytes	2				
8 Bytes	3				
10 Bytes	4		2		
12 Bytes	5				
14 Bytes	6				
16 Bytes	7				

### Example A: Transmitting a Single-Cycle UFC Message

The procedure for transmitting a single cycle UFC message is shown in Figure 4-5. In this case a 4-byte message is being sent on a 2-byte interface.

**Note:** TX\_DST\_RDY\_N is deasserted for two cycles. Aurora 8B/10B cores use this gap in the data flow to transmit the UFC header and message data.



UG353\_05\_05\_100809

Figure 4-5: Transmitting a Single-Cycle UFC Message

### Example B: Transmitting a Multi-Cycle UFC Message

The procedure for transmitting a two-cycle UFC message is shown in Figure 4-6. In this case the user application is sending a 16-byte message using a 2-byte interface.

TX\_DST\_RDY\_N is asserted for three cycles; one cycle for the UFC header which is sent during the UFC\_TX\_ACK\_N cycle, and two cycles for data.

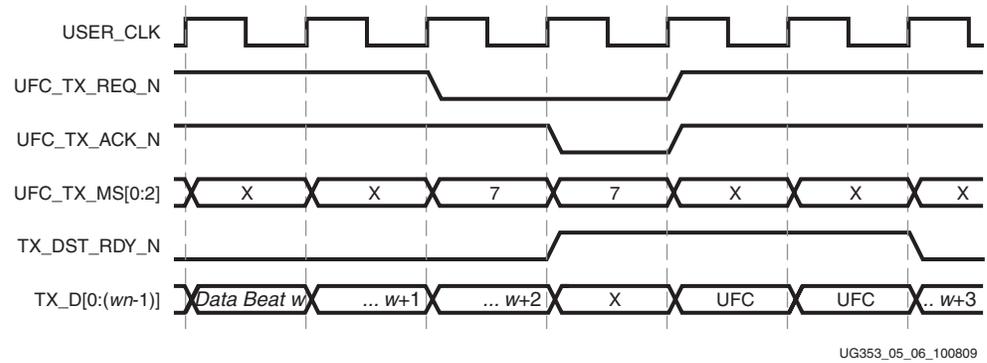


Figure 4-6: Transmitting a Multi-Cycle UFC Message

## Receiving User Flow Control Messages

When the Aurora 8B/10B core receives a UFC message, it passes the data from the message to the user application through a dedicated UFC LocalLink interface. The data is presented on the UFC\_RX\_DATA port; UFC\_RX\_SOF\_N indicates the start of the message data and UFC\_RX\_EOF\_N indicates the end. UFC\_RX\_REM is used to show the number of valid bytes on UFC\_RX\_DATA during the last cycle of the message (for example, while UFC\_RX\_EOF\_N is asserted). Signals on the UFC\_RX LocalLink interface are only valid when UFC\_RX\_SRC\_RDY\_N is asserted.

### Example C: Receiving a Single-Cycle UFC Message

Figure 4-7 shows an Aurora 8B/10B core with a 4-byte data interface receiving a 4-byte UFC message. The core presents this data to the user application by asserting UFC\_RX\_SRC\_RDY\_N, UFC\_RX\_SOF\_N and UFC\_RX\_EOF\_N to indicate a single cycle frame. The UFC\_RX\_REM bus is set to 3, indicating only the four most significant bytes of the interface are valid.

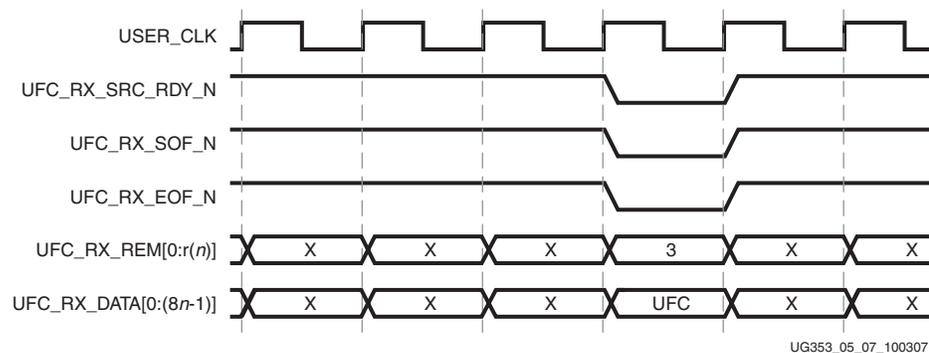


Figure 4-7: Receiving a Single-Cycle UFC Message

### Example D: Receiving a Multi-Cycle UFC Message

Figure 4-8 shows an Aurora 8B/10B core with a 4-byte interface receiving an 8-byte message.

**Note:** The resulting frame is two cycles long, with UFC\_RX\_REM set to 3 on the second cycle indicating that all eight bytes of the data are valid.

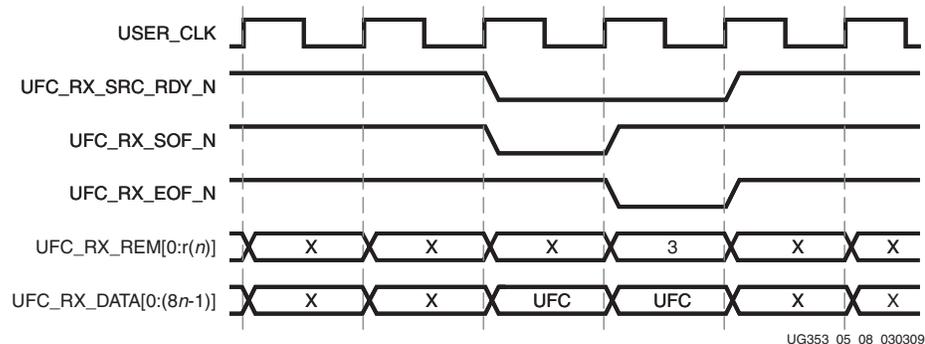


Figure 4-8: Receiving a Multi-Cycle UFC Message



# *Status, Control, and the GTP/GTX Block Interface*

---

## **Introduction**

The status and control ports of the Aurora 8B/10B core allow user applications to monitor the Aurora 8B/10B channel and use built-in features of the GTP/GTX transceivers. Aurora 8B/10B cores can be configured as full-duplex or simplex modules. Full-duplex modules provide high-speed TX and RX links. Simplex modules provide a link in only one direction and are initialized using sideband ports.

This chapter provides diagrams and port descriptions for the Aurora 8B/10B core's status and control interface, along with the GTP/GTX transceiver serial I/O interface and the sideband initialization ports that are used exclusively for simplex modules.

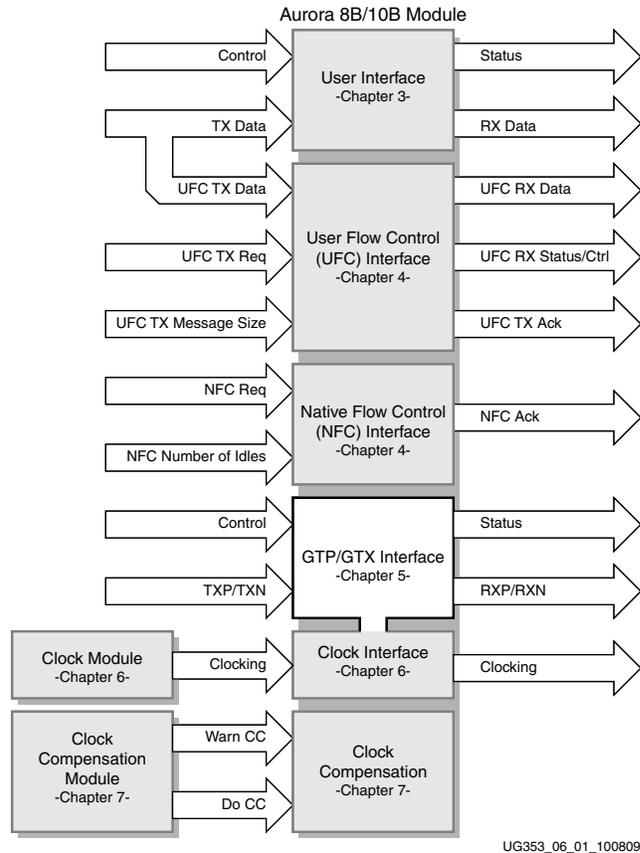


Figure 5-1: Top-Level GTP/GTX Block Interface

## Full-Duplex Cores

### Full-Duplex Status and Control Ports

Full-duplex cores provide a TX and an RX Aurora 8B/10B channel connection. Figure 5-2 shows the status and control interface for a full-duplex Aurora 8B/10B core. Table 5-1 describes the function of each of the ports in the interface.

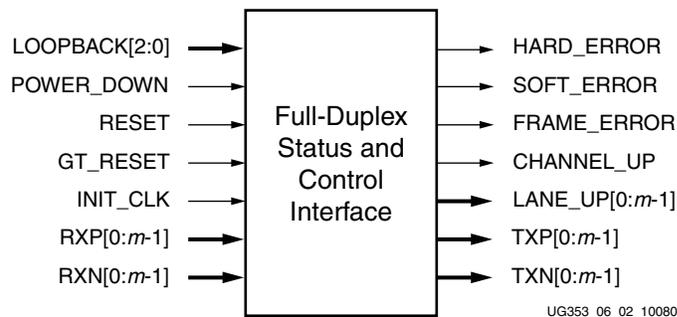


Figure 5-2: Status and Control Interface for Full-Duplex Cores

**Table 5-1: Status and Control Ports for Full-Duplex Cores**

Name	Direction	Description
CHANNEL_UP	Output	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data. The Aurora 8B/10B core cannot receive data before CHANNEL_UP.
LANE_UP[0:m-1]	Output	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High). The Aurora 8B/10B core can only receive data after all LANE_UP signals are High.
FRAME_ERROR	Output	Channel frame/protocol error detected. This port is active-High and is asserted for a single clock.
HARD_ERROR	Output	Hard error detected. (Active-High, asserted until Aurora 8B/10B core resets). See <a href="#">"Error Signals in Full-Duplex Cores,"</a> page 72 for more details.
LOOPBACK[2:0]	Input	The LOOPBACK[2:0] port selects between the normal operation mode and the different loopback modes. See the <i>Vertex-5 FPGA RocketIO GTP Transceiver User Guide</i> , <i>Vertex-5 FPGA RocketIO GTX Transceiver User Guide</i> , <i>Vertex-6 FPGA GTX Transceivers User Guide</i> , and <i>Spartan-6 FPGA GTP Transceivers User Guide</i> for details about loopback. See <a href="#">"Related Xilinx Documents"</a> in Chapter 1.
POWER_DOWN	Input	Drives the power-down input of the GTP/GTX transceiver (active-High).
RESET	Input	Resets the Aurora 8B/10B core (active-High). This signal must be synchronous to USER_CLK and must be asserted for at least one USER_CLK cycle.
SOFT_ERROR	Output	Soft error detected in the incoming serial stream. See <a href="#">"Error Signals in Full-Duplex Cores,"</a> page 72 for more details. (Active-High, asserted for a single clock).
RXP[0:m-1]	Input	Positive differential serial data input pin.
RXN[0:m-1]	Input	Negative differential serial data input pin.
TXP[0:m-1]	Output	Positive differential serial data output pin.
TXN[0:m-1]	Output	Negative differential serial data output pin.
GT_RESET	Input	The reset signal for the PMA modules in the transceivers is connected to the top level through a debouncer. The GT_RESET should be asserted (active-High) when the module is first powered up in hardware. This systematically resets all PCS and PMA subcomponents of the transceiver. The signal is debounced using the INIT_CLK. See the "Reset" section in the respective transceiver user guide for further details.

Table 5-1: Status and Control Ports for Full-Duplex Cores (Cont'd)

Name	Direction	Description
INIT_CLK	Input	INIT_CLK is used to register and debounce the GT_RESET signal in cores targeted for the Virtex-5 device. INIT_CLK is required since USER_CLK stops when GT_RESET is asserted. INIT_CLK should be set to a slow rate, preferably slower than the reference clock. INIT_CLK is a board clock. For example, the ML523 board has a 50 MHz crystal oscillator and it is constrained for this frequency by default in the <component_name>_example_design.ucf file. Users need to update this clock constraint with respect to their board clock frequency.

**Notes:**

1.  $m$  is the number of GTP/GTX transceivers

## Error Signals in Full-Duplex Cores

Equipment problems and channel noise can cause errors during Aurora 8B/10B channel operation. 8B/10B encoding allows the Aurora 8B/10B core to detect all single bit errors and most multi-bit errors that occur in the channel. The core reports these errors by asserting the SOFT\_ERROR signal on every cycle they are detected.

The core also monitors each GTP/GTX transceiver for hardware errors such as buffer overflow and loss of lock. The core reports hardware errors by asserting the HARD\_ERROR signal. Catastrophic hardware errors can also manifest themselves as bursts of soft errors. The core uses the leaky bucket algorithm described in the *Aurora 8B/10B Protocol Specification* to detect large numbers of soft errors occurring in a short period of time, and will assert the HARD\_ERROR signal when it detects them.

Whenever a hard error is detected, the Aurora 8B/10B core automatically resets itself and attempts to reinitialize. In most cases, this will allow the Aurora 8B/10B channel to be reestablished as soon as the hardware issue that caused the hard error is resolved. Soft errors do not lead to a reset unless enough of them occur in a short period of time to trigger the Aurora 8B/10B leaky bucket algorithm.

Aurora 8B/10B cores with a LocalLink data interface can also detect errors in Aurora 8B/10B frames. Errors of this type include frames with no data, consecutive Start of Frame symbols, and consecutive End of Frame symbols. When the core detects a frame problem, it asserts the FRAME\_ERROR signal. This signal is usually asserted close to a SOFT\_ERROR assertion, with soft errors being the main cause of frame errors.

Table 5-2 summarizes the error conditions the Aurora 8B/10B core can detect and the error signals used to alert the user application.

Table 5-2: Error Signals in Full-Duplex Cores

Signal	Description
HARD_ERROR	<p>TX Overflow/Underflow: The elastic buffer for TX data overflows or underflows. This can occur when the user clock and the reference clock sources are not running at the same frequency.</p> <p>RX Overflow/Underflow: The elastic buffer for RX data overflows or underflows. This can occur when the clock source frequencies for the two channel partners are not within 200 ppm.</p> <p>Bad Control Character: The protocol engine attempts to send a bad control character. This is an indication of design corruption or catastrophic failure.</p> <p>Soft Errors: There are too many soft errors within a short period of time. The Aurora 8B/10B protocol defines a leaky bucket algorithm for determining the acceptable number of soft errors within a given time period. When this number is exceeded, the physical connection may be too poor for communication using the current voltage swing and pre-emphasis settings.</p>
SOFT_ERROR	<p>Invalid Code: The 10-bit code received from the channel partner was not a valid code in the 8B/10B table. This usually means a bit was corrupted in transit, causing a good code to become unrecognizable. Typically, this will also result in a frame error or corruption of the current channel frame.</p> <p>Disparity Error: The 10-bit code received from the channel partner did not have the correct disparity. This error is also usually caused by corruption of a good code in transit, and can result in a frame error or bad data if it occurs while a frame is being sent.</p>
FRAME_ERROR	<p>Truncated Frame: A channel frame is started without ending the previous channel frame, or a channel frame is ended without being started.</p> <p>Invalid Control Character: The protocol engine receives a control character that it does not recognize.</p> <p>No Data in Frame: A channel frame is received with no data.</p>

## Full-Duplex Initialization

Full-duplex cores initialize automatically after power up, reset, or hard error. Full-duplex modules on each side of the channel perform the Aurora 8B/10B initialization procedure until the channel is ready for use. The LANE\_UP bus indicates which lanes in the channel have finished the lane initialization portion of the initialization procedure. This signal can be used to help debug equipment problems in a multi-lane channel. CHANNEL\_UP is asserted only after the core completes the entire initialization procedure.

Aurora 8B/10B cores cannot receive data before CHANNEL\_UP is asserted. Only the RX\_SRC\_RDY\_N signal on the user interface should be used to qualify incoming data. CHANNEL\_UP can be inverted and used to reset modules that drive the TX side of a full-duplex channel, since no transmission can occur until after CHANNEL\_UP. If user application modules need to be reset before data reception, one of the LANE\_UP signals can be inverted and used. Data cannot be received until after all the LANE\_UP signals are asserted.

## Simplex Cores

### Simplex TX Status and Control Ports

Simplex TX cores allow user applications to transmit data to a simplex RX core. They have no RX connection. [Figure 5-3](#) shows the status and control interface for a simplex TX core. [Table 5-3](#) describes the function of each of the ports in the interface.

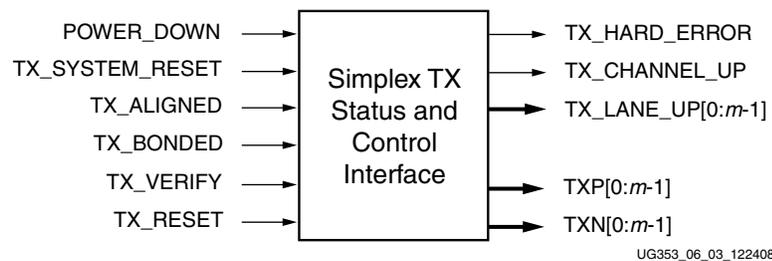


Figure 5-3: Status and Control Interface for Simplex TX Core

Table 5-3: Status and Control Ports for Simplex TX Cores

Name	Direction	Description
TX_ALIGNED	Input	Asserted when RX channel partner has completed lane initialization for all lanes. Typically connected to RX_ALIGNED.
TX_BONDED	Input	Asserted when RX channel partner has completed channel bonding. Not needed for single-lane channels. Typically connected to RX_BONDED.
TX_VERIFY	Input	Asserted when RX channel partner has completed verification. Typically connected to RX_VERIFY.

Table 5-3: Status and Control Ports for Simplex TX Cores (Cont'd)

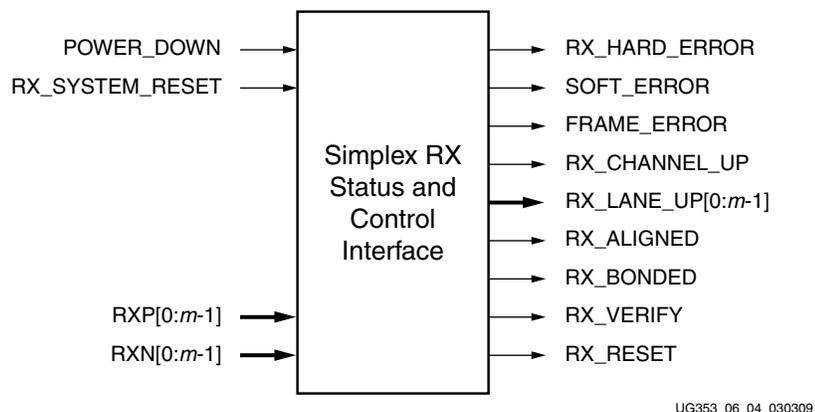
Name	Direction	Description
TX_RESET	Input	Asserted when reset is required because of initialization status of RX channel partner. This signal must be synchronous to USER_CLK and must be asserted for at least one USER_CLK cycle. Typically connected to RX_RESET.
TX_CHANNEL_UP	Output	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data. The Aurora 8B/10B core cannot receive data before TX_CHANNEL_UP.
TX_LANE_UP[0:m-1]	Output	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High).
TX_HARD_ERROR	Output	Hard error detected. (Active-High, asserted until Aurora 8B/10B core resets). See “Error Signals in Simplex Cores,” page 80 for more details.
POWER_DOWN	Input	Drives the powerdown input of the GTP/GTX transceiver (active-High).
TX_SYSTEM_RESET	Input	Resets the Aurora 8B/10B core (active-High).
TXP[0:m-1]	Output	Positive differential serial data output pin.
TXN[0:m-1]	Output	Negative differential serial data output pin.

**Notes:**

1. *m* is the number of GTP/GTX transceivers.

## Simplex RX Status and Control Ports

Simplex RX cores allow user applications to receive data from a simplex TX core. Figure 5-4 shows the status and control interface for a simplex RX core. Table 5-4, page 76 describes the function of each of the ports in the interface.



UG353\_06\_04\_030309

Figure 5-4: Status and Control Interface for Simplex RX Core

Table 5-4: Status and Control Ports for Simplex RX Cores

Name	Direction	Description
RX_ALIGNED	Output	Asserted when RX module has completed lane initialization. Typically connected to TX_ALIGNED.
RX_BONDED	Output	Asserted when RX module has completed channel bonding. Not used for single-lane channels. Typically connected to TX_BONDED.
RX_VERIFY	Output	Asserted when RX module has completed verification. Typically connected to TX_VERIFY.
RX_RESET	Output	Asserted when the RX module needs the TX module to restart initialization. Typically connected to TX_RESET.
RX_CHANNEL_UP	Output	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data. The Aurora 8B/10B core cannot receive data before RX_CHANNEL_UP.
RX_LANE_UP[0:m-1]	Output	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High). The Aurora 8B/10B core can only receive data after all RX_LANE_UP signals are High.
FRAME_ERROR	Output	Channel frame/protocol error detected. This port is active-High and is asserted for a single clock.
RX_HARD_ERROR	Output	Hard error detected. (Active-High, asserted until Aurora 8B/10B core resets). See <a href="#">“Error Signals in Simplex Cores,”</a> page 80 for more details.
POWER_DOWN	Input	Drives the power-down input of the GTP/GTX transceiver (active-High).
RX_SYSTEM_RESET	Input	Resets the Aurora 8B/10B core (active-High).
SOFT_ERROR	Output	Soft error detected in the incoming serial stream. See <a href="#">“Error Signals in Simplex Cores,”</a> page 80 for more details. (Active-High, asserted for a single clock).
RXP[0:m-1]	Input	Positive differential serial data input pin.
RXN[0:m-1]	Input	Negative differential serial data input pin.

**Notes:**

1.  $m$  is the number of GTP/GTX transceivers.
2. RX\_ALIGNED, RX\_BONDED, RX\_VERIFY, and RX\_RESET are available as output signals even when the simplex partner is timer based, but functionally these signals are not required.

## Simplex Both (RX and TX) Status and Control Ports

Simplex Both cores consist of a simplex TX core and a simplex RX core sharing the same set of GTP/GTX transceivers. Like a full-duplex core, the simplex Both core transmits and receives data. Two key differences are as follows:

- The TX and RX sides of the simplex Both core initialize and run independently of each other, unlike the duplex core, where both TX and RX must be operational for either direction to work.
- Simplex Both cores only connect to other simplex cores, while full-duplex cores only connect to other full-duplex cores. The TX side of a simplex Both core connects to a simplex RX core, or to the RX side of a simplex Both core; the RX side of a simplex Both core connects to a simplex TX core, or to the TX side of a simplex Both core.

Figure 5-5 shows the status and control interface for a simplex Both core. Table 5-5, page 78 describes the function of each of the ports in the interface.

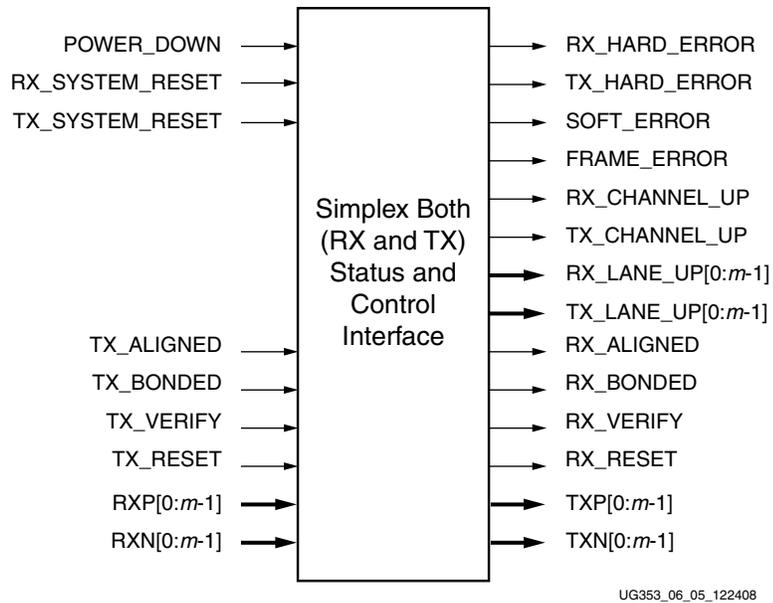


Figure 5-5: Status and Control Interface for Simplex Both Cores

Table 5-5: Status and Control Ports for Simplex Both Cores

Name	Direction	Description
TX_ALIGNED	Input	Asserted when RX channel partner has completed lane initialization for all lanes. Typically connected to RX_ALIGNED.
RX_ALIGNED	Output	Asserted when RX module has completed lane initialization. Typically connected to TX_ALIGNED.
TX_BONDED	Input	Asserted when RX channel partner has completed channel bonding. Not needed for single-lane channels. Typically connected to RX_BONDED.
RX_BONDED	Output	Asserted when RX module has completed channel bonding. Not used for single-lane channels. Typically connected to TX_BONDED.
TX_VERIFY	Input	Asserted when RX channel partner has completed verification. Typically connected to RX_VERIFY.
RX_VERIFY	Output	Asserted when RX module has completed verification. Typically connected to TX_VERIFY.
TX_RESET	Input	Asserted when reset is required because of initialization status of RX channel partner. Typically connected to RX_RESET.
RX_RESET	Output	Asserted when the RX module needs the TX module to restart initialization. Typically connected to TX_RESET.
RX_CHANNEL_UP	Output	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to receive data. The Aurora 8B/10B core cannot receive data before CHANNEL_UP.
TX_CHANNEL_UP	Input	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data. The Aurora 8B/10B core cannot receive data before CHANNEL_UP.
RX_LANE_UP[0:m-1]	Output	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High). The Aurora 8B/10B core can only receive data after all LANE_UP signals are High.
TX_LANE_UP[0:m-1]	Input	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High). The Aurora 8B/10B core can only transmit data after all LANE_UP signals are High.
FRAME_ERROR	Output	Channel frame/protocol error detected. This port is active-High and is asserted for a single clock.
RX_HARD_ERROR TX_HARD_ERROR	Output	Hard error detected. (Active-High, asserted until Aurora 8B/10B core resets). See <a href="#">“Error Signals in Simplex Cores,” page 80</a> for more details.

Table 5-5: Status and Control Ports for Simplex Both Cores (Cont'd)

Name	Direction	Description
POWER_DOWN	Input	Drives the power-down input of the GTP/GTX transceiver (active-High).
RX_SYSTEM_RESET TX_SYSTEM_RESET	Input	Resets the Aurora 8B/10B core (active-High).
SOFT_ERROR	Output	Soft error detected in the incoming serial stream. See <a href="#">"Error Signals in Simplex Cores,"</a> page 80 for more details. (Active-High, asserted for a single clock).
RXP[0:m-1]	Input	Positive differential serial data input pin.
RXN[0:m-1]	Input	Negative differential serial data input pin.
TXP[0:m-1]	Output	Positive differential serial data output pin.
TXN[0:m-1]	Output	Negative differential serial data output pin.

**Notes:**

1.  $m$  is the number of GTP/GTX transceivers.

## Error Signals in Simplex Cores

The 8B/10B encoding allows RX simplex cores and the RX sides of simplex Both cores to detect all single bit errors and most multi-bit errors in a simplex channel. The cores report these errors by asserting the SOFT\_ERROR signal on every cycle an error is detected. TX simplex cores do not include a SOFT\_ERROR port. All transmit data is assumed correct at transmission unless there is an equipment problem.

All simplex cores monitor their GTP/GTX transceivers for hardware errors such as buffer overflow and loss of lock. Hardware errors on the TX side of the channel are reported by asserting the TX\_HARD\_ERROR signal; RX side hard errors are reported using the RX\_HARD\_ERROR signal. Simplex RX and simplex Both cores use the Aurora 8B/10B protocol's leaky bucket algorithm to evaluate bursts of soft errors. If too many soft errors occur in a short span of time, RX\_HARD\_ERROR is asserted.

Whenever a hard error is detected, the Aurora 8B/10B core automatically resets itself and attempts to reinitialize. In simplex Both cores, TX hard errors reset only the TX side, and RX errors reset only the RX side. Resetting allows the Aurora 8B/10B channel to be re-established as soon as the hardware issue that caused the hard error is resolved in most cases. Soft errors do not lead to a reset unless enough of them occur in a short period of time to trigger the Aurora 8B/10B leaky bucket algorithm.

Simplex RX and simplex Both cores with a LocalLink data interface can also detect errors in Aurora 8B/10B frames when they are received. Errors of this type include frames with no data, consecutive Start of Frame symbols, and consecutive End of Frame symbols. When the core detects a frame problem, it asserts the FRAME\_ERROR signal. This signal will usually be asserted close to a SOFT\_ERROR assertion, as soft errors are the main cause of frame errors. Simplex TX modules do not use the FRAME\_ERROR port.

[Table 5-6](#) summarizes the error conditions simplex Aurora 8B/10B cores can detect and the error signals uses to alert the user application.

**Table 5-6: Error Signals in Simplex Cores**

Signal	Description	TX	RX	Both
HARD_ERROR	TX Overflow/Underflow: The elastic buffer for TX data overflows or underflows. This can occur when the user clock and the reference clock sources are not running at the same frequency.	x		x
	RX Overflow/Underflow: The elastic buffer for RX data overflows or underflows. This can occur when the clock source frequencies for the two channel partners are not within 200 ppm.		x	x
	Bad Control Character: The protocol engine attempts to send a bad control character. This is an indication of design corruption or catastrophic failure.	x		x
	Soft Errors: There are too many soft errors within a short period of time. The Aurora 8B/10B protocol defines a leaky bucket algorithm for determining the acceptable number of soft errors within a given time period. When this number is exceeded, the physical connection may be too poor for communication using the current voltage swing and pre-emphasis settings.		x	x

Table 5-6: Error Signals in Simplex Cores (Cont'd)

Signal	Description	TX	RX	Both
SOFT_ERROR	Invalid Code: The 10-bit code received from the channel partner was not a valid code in the 8B/10B table. This usually means a bit was corrupted in transit, causing a good code to become unrecognizable. Typically, this will also result in a frame error or corruption of the current channel frame.		x	x
	Disparity Error: The 10-bit code received from the channel partner did not have the correct disparity. This error is also usually caused by corruption of a good code in transit, and can result in a frame error or bad data if it occurs while a frame is being sent.		x	x
	No Data in Frame: A channel frame is received with no data.		x	x
FRAME_ERROR	Truncated Frame: A channel frame is started without ending the previous channel frame, or a channel frame is ended without being started.	x		x
	Invalid Control Character: The protocol engine receives a control character that it does not recognize.		x	x
	Invalid UFC Message Length: A UFC message is received with an invalid length.		x	x

## Simplex Initialization

Simplex cores do not depend on signals from an Aurora 8B/10B channel for initialization. Instead, the TX and RX sides of simplex channels communicate their initialization state through a set of sideband initialization signals. The initialization ports are called ALIGNED, BONDED, VERIFY, and RESET; one set for the TX side with a TX\_ prefix, and one set for the RX side with an RX\_ prefix. The bonded port is only used for multi-lane cores.

There are two ways to initialize a simplex module using the sideband initialization signals:

- Send the information from the RX sideband initialization ports to the TX sideband initialization ports
- Drive the TX sideband initialization ports independently of the RX sideband initialization ports using timed initialization intervals

Both initialization methods are described in the following sections.

### Using a Back Channel

If there is no communication channel available from the RX side of the connection to the TX side, using a back channel is the safest way to initialize and maintain a simplex channel. There are very few requirements on the back channel; it need only deliver messages to the TX side to indicate which of the sideband initialization signals is asserted when the signals change.

The `aurora_example` design included in the `example_design` directory with simplex Aurora 8B/10B cores shows a simple side channel that uses 3 or 4 I/O pins on the device.

## Using Timers

For some systems a back channel is not possible. In these cases, serial channels can be initialized by driving the TX simplex initialization with a set of timers. The timers must be designed carefully to meet the needs of the system since the average time for initialization depends on many channel specific conditions such as clock rate, channel latency, skew between lanes, and noise.

Some of the initialization logic in the Aurora 8B/10B module uses watchdog timers to prevent deadlock. These watchdog timers are used on the RX side of the channel, and can interfere with the proper operation of TX initialization timers. If the RX simplex module goes from ALIGNED, BONDED or VERIFY, to RESET, make sure that it is not because the TX logic spend too much time in one of those states. If a particularly long timer is required to meet the needs of the system, the watchdog timers can be adjusted by editing the `lane_init_sm` module and the `channel_init_sm` module. For most cases, this should not be necessary and is not recommended.

Aurora 8B/10B channels normally reinitialize only in the case of failure. When there is no back channel available, event-triggered re-initialization is impossible for most errors since it is usually the RX side that detects a failure and the TX side that must handle it. The solution for this problem is to make timer-driven TX simplex modules reinitialize on a regular basis. If a catastrophic error occurs, the channel will be reset and running again once the next re-initialization period arrives. System designers should balance the average time required for re-initialization against the maximum time their system can tolerate an inoperative channel to determine the optimum re-initialization period for their systems.

## Reset and Power Down

### Reset

The reset signals on the control and status interface are used to set the Aurora 8B/10B core to a known starting state. Resetting the core stops any channels that are currently operating; after reset, the core attempts to initialize a new channel.

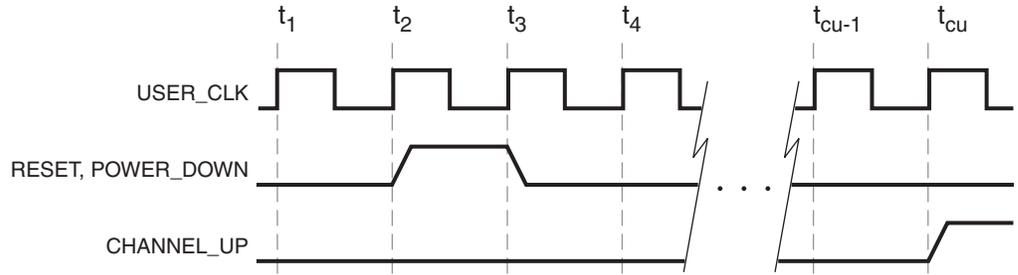
On full-duplex modules, the RESET signal resets both the TX and RX sides of the channel when asserted on the positive edge of USER\_CLK. On simplex modules, the resets for the TX and RX channels are separate. TX\_SYSTEM\_RESET resets TX channels; RX\_SYSTEM\_RESET resets RX channels. The TX\_SYSTEM\_RESET is separate from the TX\_RESET and RX\_RESET signals used on the simplex sideband interface.

### Power Down

This is an active-High signal. When POWER\_DOWN is asserted, the GTP/GTX transceivers in the Aurora 8B/10B core are turned off, putting them into a non-operating low-power mode. When POWER\_DOWN is deasserted, the core automatically resets. Be careful when asserting this signal on cores that use TX\_OUT\_CLK (see the [Chapter 6, “Clock Interface and Clocking”](#)). TX\_OUT\_CLK will stop when the GTP/GTX transceivers are powered down. See the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide*, the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*, the *Virtex-6 FPGA GTX Transceivers User Guide*, and the *Spartan-6 FPGA GTP Transceivers User Guide* for the device you are using for details about powering down GTP/GTX transceivers.

## Timing

Figure 5-6 shows the timing for the RESET and POWER\_DOWN signals. In a quiet environment,  $t_{CU}$  is generally less than 800 clocks; in a noisy environment,  $t_{CU}$  can be much longer.



UG353\_06\_06\_100307

Figure 5-6: Reset and Power Down Timing



## *Clock Interface and Clocking*

---

### **Introduction**

Good clocking is critical for the correct operation of the Aurora 8B/10B core. The core requires a high-quality, low-jitter reference clock to drive the high-speed TX clock and clock recovery circuits in the GTP/GTX transceiver. It also requires at least one frequency locked parallel clock for synchronous operation with the user application.

The Virtex<sup>®</sup>-5 FPGA GTP/GTX architecture has a pair of transceivers in each GTP/GTX\_DUAL tile and the Virtex-6 FPGA has four GTX transceivers in a Quad. The Spartan<sup>®</sup>-6 FPGA GTP architecture has a pair of transceivers in each GTPA1\_DUAL tile. The Virtex-5 FPGA has a shared PMA PLL architecture inside the GTP/GTX\_DUAL tile, and each reference clock sources both channels. The Virtex-6 FPGA GTX Quad has individual PLLs for both TX and RX portion of the transceivers. The Spartan-6 FPGA has individual PLLs for each transceiver in a GTPA1\_DUAL tile. The reference clock is used to produce the PLL clock, which is divided to make individual TX and RX serial clocks and parallel clocks in each GTP/GTX transceiver.

Each Aurora 8B/10B core is generated in the example\_design directory that includes a design called aurora\_example. This design by instantiating the generated Aurora 8B/10B core, demonstrates a working clock configuration of the core. First-time users should examine the aurora example design and use it as a template when connecting the clock interface.

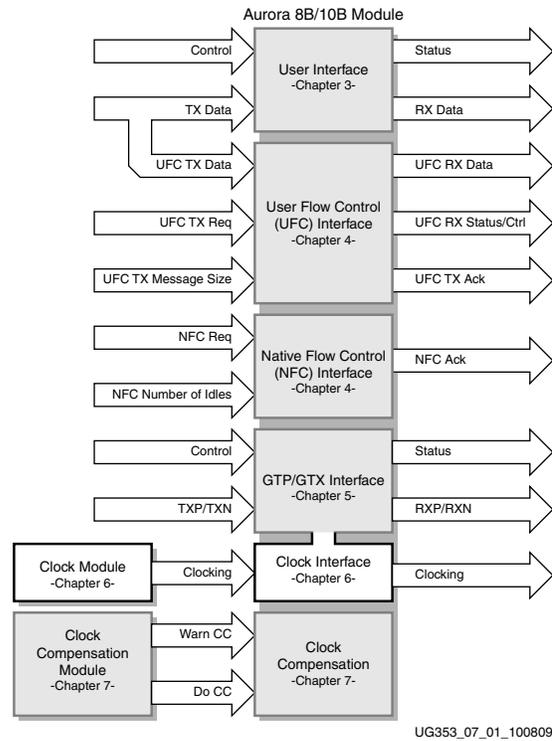


Figure 6-1: Top-Level Clocking

## Clock Interface Ports for GTP/GTX Transceiver Cores

Table 6-1 describes the Aurora 8B/10B core clock ports.

**Table 6-1: Clock Ports for a GTP/GTX Aurora 8B/10B Core**

<b>Clock Ports</b>	<b>Direction</b>	<b>Description</b>
PLL_NOT_LOCKED	Input	If a PLL is used to generate clocks for the Aurora 8B/10B core, the PLL_NOT_LOCKED signal should be connected to the inverse of the PLL's LOCKED signal. The clock module provided with the Aurora 8B/10B core uses the PLL for clock division. The PLL_NOT_LOCKED signal from the clock module should be connected to the PLL_NOT_LOCKED signal on the Aurora 8B/10B core. If PLL is not used to generate clock signals for the Aurora 8B/10B core, tie PLL_NOT_LOCKED to ground.
USER_CLK	Input	Parallel clock shared by the Aurora 8B/10B core and the user application. In Aurora 8B/10B cores, USER_CLK and SYNC_CLK are the output of a PLL whose input is derived from TX_OUT_CLK. These clock generations are available in <component_name>_clock_module file. The Spartan-6 FPGA uses the GTPCLKOUT port to derive USER_CLK and SYNC_CLK outputs. USER_CLK goes as TXUSRCLK2 input to the transceiver tile. Refer to the respective transceiver user guide for more information.
SYNC_CLK	Input	Parallel clock used by the internal synchronization logic of the GTP/GTX transceivers in the Aurora 8B/10B core. SYNC_CLK goes as TXUSRCLK input to the transceiver tile. Refer to the respective transceiver user guide for more information.
GT_REFCLK	Input	GT_REFCLK (CLKP/CLKN) is a dedicated external clock generated from an oscillator. This clock is fed through IBUFDS. To minimize the number of oscillators, the GTP/GTX transceiver architecture has a NORTH/SOUTH clock routing matrix using CLKP/CLKN. A GTP/GTX_DUAL tile shares its clock with its neighbors using dedicated clock routing resources.

## Reference Clocks for Virtex-5 FPGA GTP/GTX Transceiver Designs

In Virtex-5 FPGA transceiver designs, the reference clock is GTPD/GTXD, which is a differential input clock for each GTP/GTX\_DUAL. In Virtex-6 FPGA transceiver designs, the reference clock is GTXQ, which is a differential input clock for each GTX\_QUAD. The reference clock for GTP/GTX\_DUAL is provided through the CLKIN port.

The two possible use models for distributing a reference clock to drive the CLKIN port are:

- “Clocking from an External Source”
- “Clocking from a Neighboring GTP/GTX\_DUAL Tile”

## Clocking from an External Source

Each GTP/GTX\_DUAL tile has a pair of dedicated pins that can be connected to an external clock source. To use these pins, a IBUFDS primitive is instantiated. In the user constraints file (.ucf), the IBUFDS input pins are set to the dedicated clock pins for the tile. In the design, the output of the IBUFDS is connected to the CLKIN port.

Each GTP/GTX\_DUAL takes differential GTPD/GTXD clock inputs, which are directly bonded to the FPGA pins. GTPD/GTXD clock should be used for high data rate applications (1 Gb/s or higher). For multilane Aurora 8B/10B designs, GTPD/GTXD clock of any GTP/GTX\_DUAL can be used as the reference clock for the Aurora 8B/10B design. Using a low-jitter oscillator delivers a high-quality clock suitable for top-speed operation. [Figure 6-2](#) shows a differential GTP/GTX\_DUAL clock pin pair sourced by an external oscillator on the board.

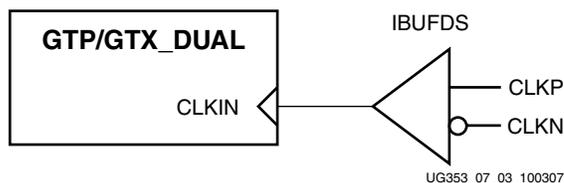


Figure 6-2: Single GTP/GTX\_DUAL Tile Clocked Externally

## Clocking from a Neighboring GTP/GTX\_DUAL Tile

The external clock from one tile can be used to drive the CLKIN ports of neighboring tiles. The example in Figure 6-3 uses the clock from one GTP/GTX\_DUAL tile to clock six neighboring tiles. A GTP/GTX\_DUAL tile shares its clock with its neighbors using dedicated clock routing resources. GREFCLK cannot be shared using these resources.

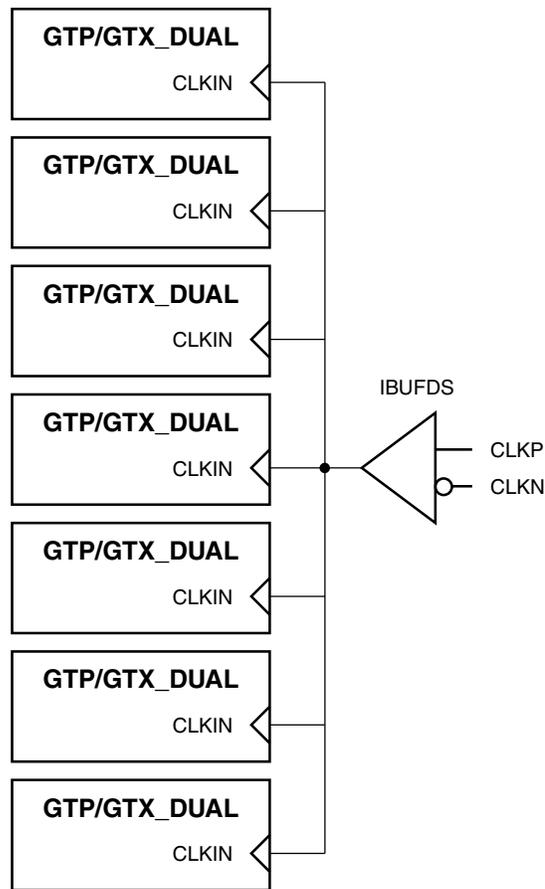


Figure 6-3: Multiple Tiles with a Shared Reference Clock

To keep the jitter of this configuration within the jitter margin for high-speed designs, the following rules must be observed:

1. The number of GTP/GTX\_DUAL tiles above the sourcing GTP/GTX\_DUAL tile must not exceed three.
2. The number of GTP/GTX\_DUAL tiles below the sourcing GTP/GTX\_DUAL tile must not exceed three.
3. The total number of GTP/GTX\_DUAL tiles sourced by the external clock pin pair (CLKN/CLKP) must not exceed seven.

The maximum number of GTP/GTX transceivers that can be sourced by a single clock pin pair translates to a maximum of 14 pairs. Designs with more than 14 GTP/GTX transceivers require the use of multiple external clock pins to ensure that the rules for controlling jitter are followed. When multiple clock pins are used, it is recommended that a low-skew LVDS buffer be used externally to drive all the pins from the same low-jitter oscillator. This is critical when several GTP/GTX transceivers are combined to form a single channel.

**Note:** In the current release, the CORE Generator™ software GUI does not allow selecting an external clock from a tile in which no GTP/GTX transceivers are selected.

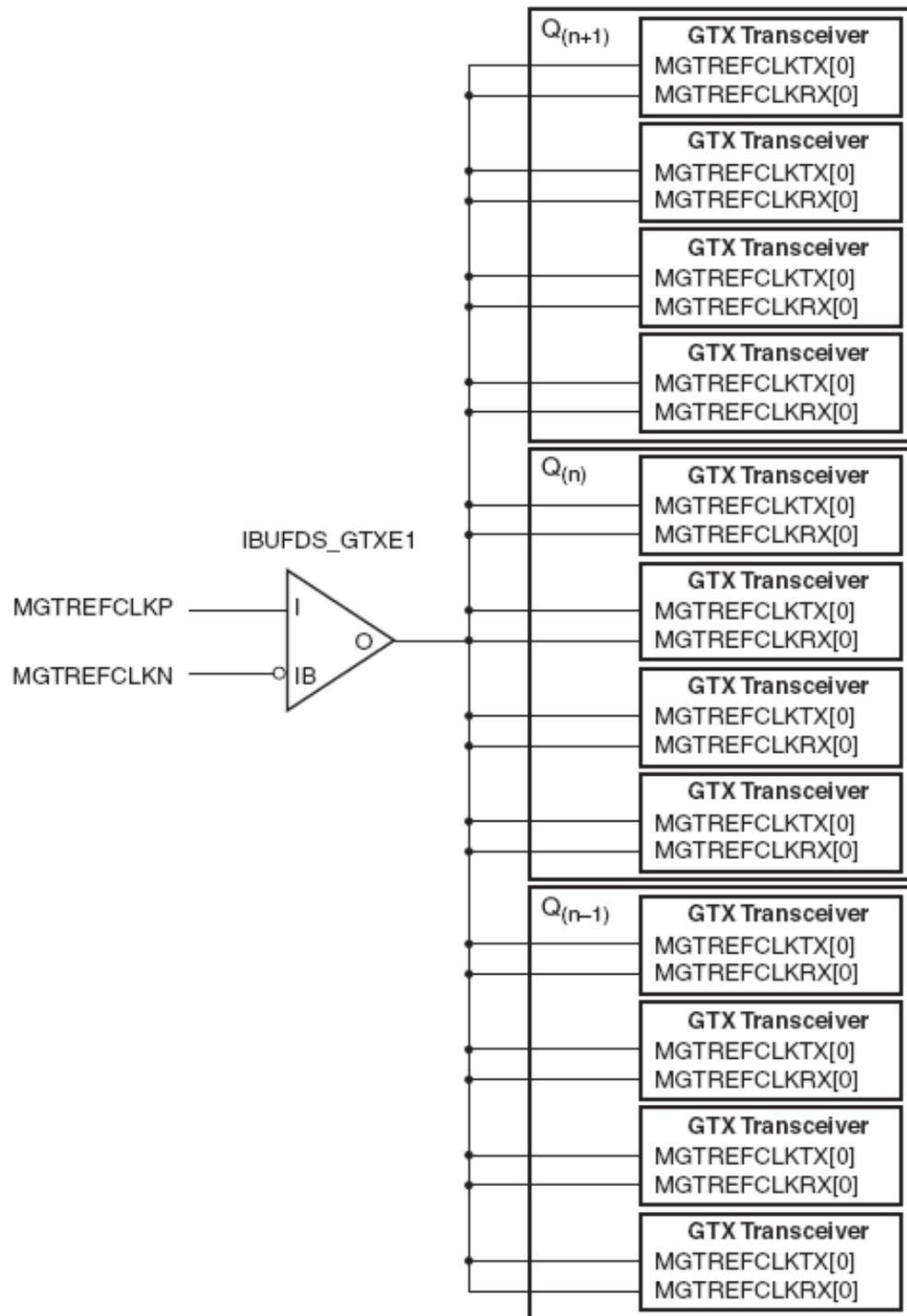
## Clocking from a Neighboring GTX\_QUAD Tile for Virtex-6 FPGA Designs

The Xilinx implementation tools will make necessary adjustments to the north-south routing shown in [Figure 6-4, page 91](#) as well as pin swapping necessary to GTXE1 clock inputs in order to route clocks from one Quad to another when required.

The following rules must be observed when sharing a reference clock to ensure that jitter margins for high-speed designs are met:

1. The number of GTX Quads above the sourcing Quad must not exceed one.
2. The number of GTX Quads below the sourcing Quad must not exceed one.
3. The total number of GTX Quads sourced by an external clock pin pair (MGTREFCLKN/MGTREFCLKP) must not exceed three or 12 GTXE1 transceivers.

The maximum number of GTX transceivers that can be sourced by a single clock pin pair is 12. Designs with more than 12 transceivers require the use of multiple external clock pins to ensure that the rules for controlling jitter are followed. When multiple clock pins are used, an external buffer can be used to drive them from the same oscillator.



UG353\_07\_05\_032309

Figure 6-4: North-South Routing Adjustments in Virtex-6 FPGAs

## Reference Clocks for Spartan-6 FPGA GTP Transceiver Designs

In Spartan-6 FPGA transceiver designs, the reference clock is GTPD, which is a differential input clock for each GTPA1\_DUAL. The reference clock for GTPA1\_DUAL is provided through the CLK00 and CLK01 ports. The two possible use models for distributing a reference clock to drive the CLK00 and CLK01 ports are:

- “Clocking from an External Source”
- “Clocking from a Neighboring GTPA1\_DUAL Tile”

### Clocking from an External Source

Each GTPA1\_DUAL tile has a pair of dedicated pins that can be connected to an external clock source. To use these pins, a IBUFDS primitive is instantiated. In the user constraints file (.ucf), the IBUFDS input pins are set to the dedicated clock pins for the tile. In the design, the output of the IBUFDS is connected to the CLK00 and CLK01 ports. Each GTPA1\_DUAL takes differential GTPD clock inputs, which are directly bonded to the FPGA pins. For multilane Aurora 8B/10B designs, GTPD clock of any GTPA1\_DUAL can be used as the reference clock for the Aurora 8B/10B design. Using a low-jitter oscillator delivers a high-quality clock suitable for top-speed operation. Figure 6-5 shows a differential GTPA1\_DUAL clock pin pair sourced by an external oscillator on the board. This clocking mechanism is used for Spartan-6 single lane and 2-lane designs.

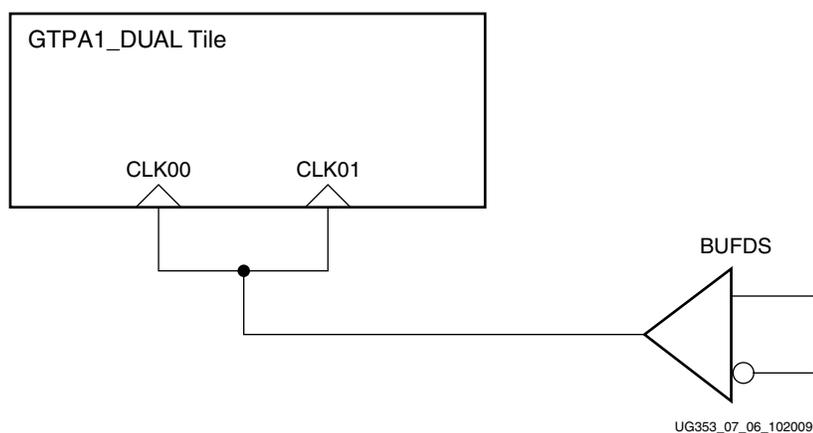
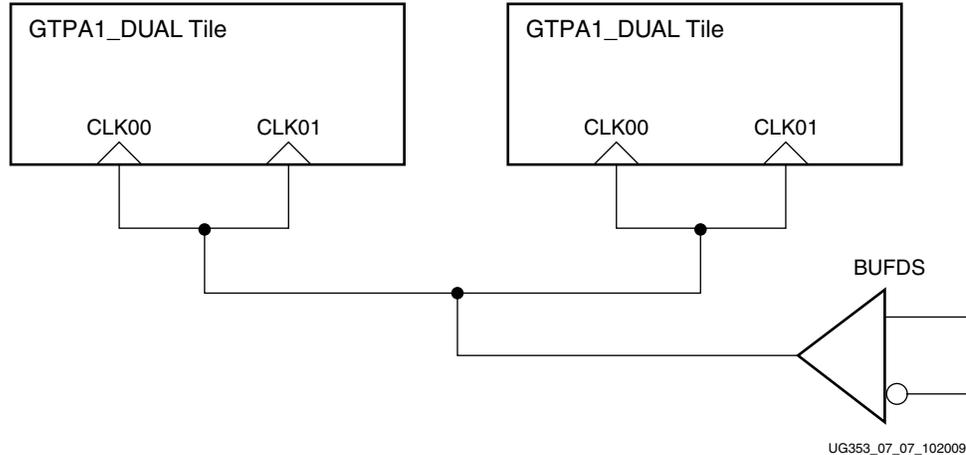


Figure 6-5: Single GTPA1\_DUAL Tile Clocked Externally

## Clocking from a Neighboring GTPA1\_DUAL Tile

The external clock from one tile can be used to drive the CLK00 and CLK01 ports of neighboring tiles.

The example in [Figure 6-6](#) uses the clock from one GTPA1\_DUAL tile to clock neighboring tiles. A GTPA1\_DUAL tile shares its clock with its neighbors using dedicated clock routing resources. This clocking mechanism is used for Spartan-6 FPGA 4-lane design.



UG353\_07\_07\_102009

Figure 6-6: GTPA1\_DUAL Tiles with Shared Reference Clock

## Clock Rates for GTP/GTX Transceiver Designs

GTP/GTX transceivers support a wide range of serial rates. The attributes used to configure the GTP/GTX transceivers in the Aurora 8B/10B core for a specific line rate are kept in the `transceiver_wrapper` module for simulation. These attributes are set automatically by the CORE Generator software in response to the line rate and reference clock selections made in the Configuration GUI window for the core. Manual edits of the attributes are not recommended, but are possible using the recommendations in the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide*, the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*, the *Virtex-6 FPGA GTX Transceivers User Guide*, and the *Spartan-6 FPGA GTP Transceivers User Guide*.

# *Clock Compensation*

---

## **Introduction**

Clock compensation is a feature that allows up to  $\pm 100$  ppm difference in the reference clock frequencies used on each side of an Aurora 8B/10B channel. This feature is used in systems where a separate reference clock source is used for each device connected by the channel, and where the same USER\_CLK is used for transmitting and receiving data.

The Aurora 8B/10B core's clock compensation interface enables full control over the core's clock compensation features. A standard clock compensation module is generated with the Aurora 8B/10B core to provide Aurora 8B/10B-compliant clock compensation for systems using separate reference clock sources; users with special clock compensation requirements can drive the interface with custom logic. If the same reference clock source is used for both sides of the channel, the interface can be tied to ground to disable clock compensation.

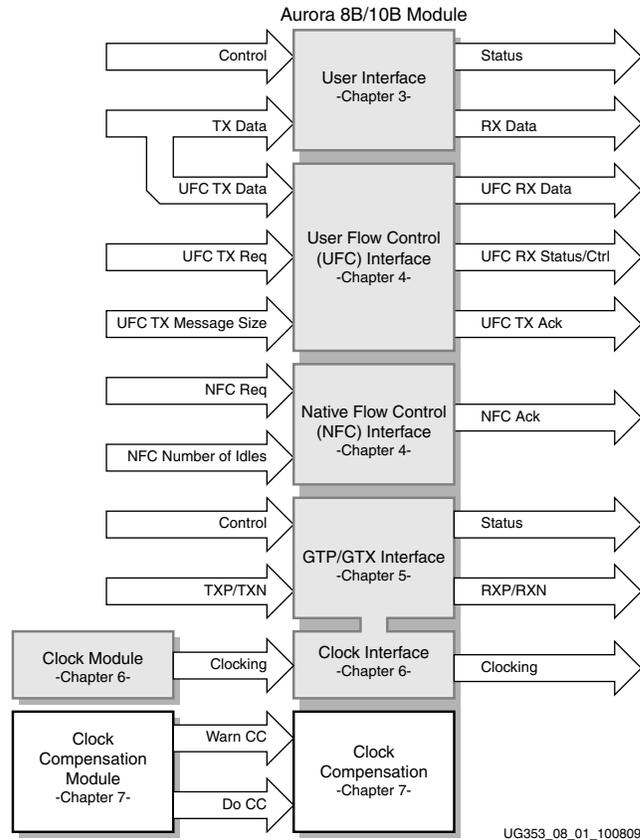


Figure 7-1: Top-Level Clock Compensation

## Clock Compensation Interface

All Aurora 8B/10B cores include a clock compensation interface for controlling the transmission of clock compensation sequences. [Table 7-1](#) describes the function of the clock compensation interface ports.

Table 7-1: Clock Compensation I/O Ports

Name	Direction	Description
DO_CC	Input	The Aurora 8B/10B core sends CC sequences on all lanes on every clock cycle when this signal is asserted. Connects to the DO_CC output on the CC module.
WARN_CC	Input	The Aurora 8B/10B core will not acknowledge UFC requests while this signal is asserted. It is used to prevent UFC messages from starting too close to CC events. Connects to the WARN_CC output on the CC module.

[Figure 7-2](#) and [Figure 7-3](#) are waveform diagrams showing how the DO\_CC signal works.

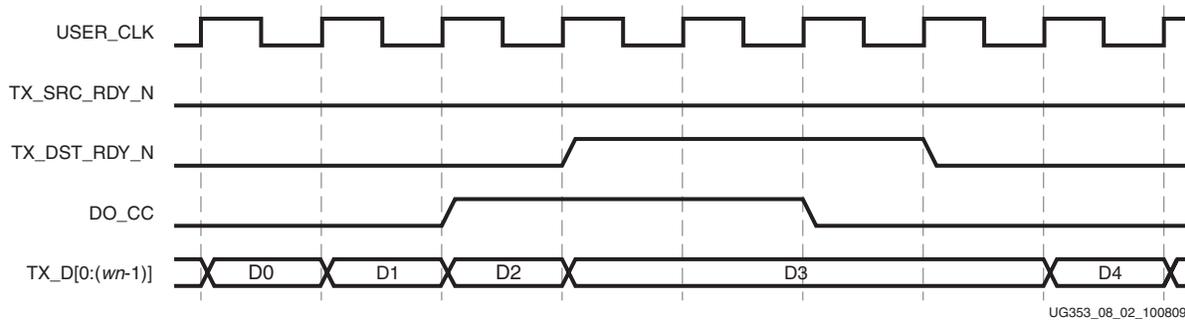


Figure 7-2: Streaming Data with Clock Compensation Inserted

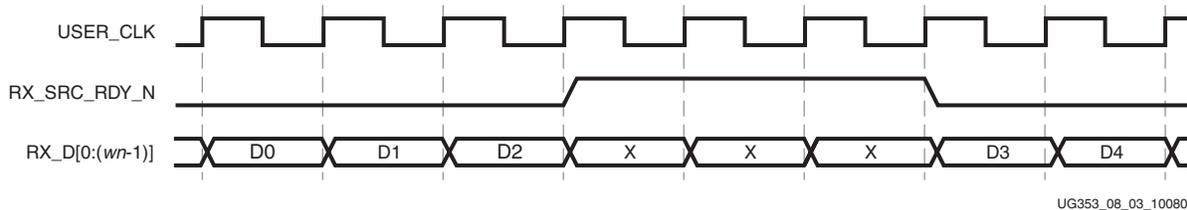


Figure 7-3: Data Reception Interrupted by Clock Compensation

The Aurora 8B/10B protocol specifies a clock compensation mechanism that allows up to  $\pm 100$  ppm difference between reference clocks on each side of an Aurora 8B/10B channel. To perform Aurora 8B/10B-compliant clock compensation, DO\_CC must be asserted for several cycles every clock compensation period. The duration of the DO\_CC assertion and the length of time between assertions is determined based on the width of the GTP/GTX transceiver data interface. While DO\_CC is asserted, TX\_DST\_RDY\_N on the user interface for modules with TX while the channel is being used to transmit clock compensation sequences. Table 7-2 shows the required durations and periods for 2-byte and 4-byte wide lanes.

Table 7-2: Clock Compensation Cycles

Lane Width	USER_CLK Cycles Between DO_CC	DO_CC Duration (USER_CLK cycles)
2	5000	6
4	3000	3

WARN\_CC is for cores with user flow control (UFC) and/or native flow control (NFC). Driving this signal before DO\_CC is asserted prevents the UFC interface from acknowledging and sending UFC messages too close to a clock correction sequence. This precaution is necessary because data corruption occurs when CC sequences and UFC messages overlap. The number of lookahead cycles required to prevent a 16-byte UFC message from colliding with a clock compensation sequence depends on the number of lanes in the channel and the width of each lane. Table 7-3 shows the number of lookahead cycles required for each combination of lane width, channel width, and maximum UFC message size.

Table 7-3: Lookahead Cycles

Data Interface Width	Max UFC Size	WARN_CC Lookahead
2	2	3
2	4	4
2	6	5
2	8	6
2	10	7
2	12	8
2	14	9
2	16	10
4	2-4	3
4	6-8	4
4	10-12	5
4	14-16	6
6	2-6	3
6	8-12	4
6	14-16	5
8	2-8	3
8	10-16	4
10	2-10	3
10	12-16	4
12	2-12	3
12	14-16	4
14	2-14	3
14	16	4
≥16	2-16	3

Native flow control message request will not be acknowledged during assertion of WARN\_CC and DO\_CC signals. This helps to prevent the collision of NFC message and clock compensation sequence.

To make Aurora 8B/10B compliance easy, a standard clock compensation module is generated along with each Aurora 8B/10B core from the CORE Generator™ software in the cc\_manager subdirectory. It automatically generates pulses to create Aurora 8B/10B compliant clock compensation sequences on the DO\_CC port and sufficiently early pulses on the WARN\_CC port to prevent UFC collisions with maximum-sized UFC messages. This module always be connected to the clock compensation port on the Aurora 8B/10B module, except in special cases. Table 7-4 shows the port description for the standard CC module.

Table 7-4: Standard CC I/O Port

Name	Direction	Description
WARN_CC	Output	Connect this port to the WARN_CC input of the Aurora 8B/10B core when using UFC.
DO_CC	Output	Connect this port to the DO_CC input of the Aurora 8B/10B core.
CHANNEL_UP	Input	Connect this port to the CHANNEL_UP output of a full-duplex core, or to the TX_CHANNEL_UP output of a simplex TX or a simplex Both port.

Clock compensation is not needed when both sides of the Aurora 8B/10B channel are being driven by the same clock (see [Figure 7-3, page 97](#)) because the reference clock frequencies on both sides of the module are locked. In this case, WARN\_CC and DO\_CC should both be tied to ground. Additionally, the CLK\_CORRECT\_USE attribute can be set to false in the transceiver interface module for the core. This can result in lower latencies for single lane modules.

Other special cases when the standard clock compensation module is not appropriate are possible. The DO\_CC port can be used to send clock compensation sequences at any time, for any duration to meet the needs of specific channels. The most common use of this feature is scheduling clock compensation events to occur outside of frames, or at specific times during a stream to avoid interrupting data flow. In general, customizing the clock compensation logic is not recommended, and when it is attempted, it should be performed with careful analysis, testing, and consideration of the following guidelines:

- Clock compensation sequences should last at least two cycles to ensure they are recognized by all receivers
- Be sure the duration and period selected is sufficient to correct for the maximum difference between the frequencies of the clocks that will be used
- Do not perform multiple clock correction sequences within eight cycles of one another
- Replacing long sequences of idles (>12 cycles) with CC sequences will result in increased EMI
- DO\_CC will have no effect until after CHANNEL\_UP; DO\_CC should be asserted immediately after CHANNEL\_UP since no clock compensation can occur during initialization



# Two Aurora 8B/10B Cores in the Virtex-5 Family Sharing a High-Speed Serial GTX Transceiver Tile

---

The high-speed serial GTX transceivers in the Virtex®-5 family have dual architecture in which two transceivers in each tile share a common PLL (see the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*). The Aurora 8B/10B core requires one transceiver per lane and the transceiver can be mapped to GTX0 or GTX1 in a tile. The unused transceiver is powered down during core generation. To use the powered down transceiver in another Aurora 8B/10B core sharing a common reference clock source, the user needs to bring out the unused transceiver ports to the Aurora 8B/10B GTX transceiver wrapper file.

The Aurora 8B/10B core instantiates the GTX\_DUAL primitive in <component name>\_tile.v[hd] file located in <component name>/example\_design/gt/. This file contains GTX0 and GTX1 specific ports together with shared tile ports. This file is hierarchically called in <component name>\_gtx\_wrapper.v[hd]. The user needs to bring out unused transceiver ports to the <component name>\_gtx\_wrapper.v[hd] file that is instantiated in the Aurora 8B/10B core's top level file. See [Figure A-1, page 103](#) for the file level hierarchy of the Aurora 8B/10B core.

## Shared Ports and Attributes

The Aurora 8B/10B core automatically sets shared PLL ports and attributes based on the CORE Generator™ software options. The reference clock input and the GTX transceiver reset to a tile is driven from the example design. The Aurora 8B/10B core derives the clock from TXOUTCLK0/1 and based on GTX0 or GTX1 selection, TXOUTCLK0 or TXOUTCLK1 is fed to the PLL input for generating the core's clock. The user can use the same TXOUTCLK source used in core 1 (See [Figure A-1, page 103](#)), or optionally can use the other TXOUCLK port for core 2. Because the reference clock and GTX transceiver reset ports are common for GTX0 and GTX1, application of the GTX transceiver reset resets both cores.

## Steps to Modify Transceiver Specific Ports and Attributes

When the user shares a tile across two Aurora 8B/10B cores, the shared PLL settings are not user modifiable. Do the following steps to modify GTX0 or GTX1 specific ports and attributes:

1. Generate an Aurora 8B/10B v5.1 core (core 1) using the CORE Generator software. Select transceiver locations such that GTX0 or GTX1 in a tile is used for the Aurora 8B/10B core.
2. Generate another Aurora 8B/10B core (core 2) such that the unused transceiver (GTX1 or GTX0) in core 1 is used for core 2.
3. Go to the directory <component name>/example\_design/gt/ of core 1.
4. Open the <component name>\_tile.v[hd] file.

This procedure assumes Line rate and GT reference clock are same for both generated Aurora 8B10B cores (core1 and core2).

## Steps to Bring Out Unused Transceiver Ports to the Aurora 8B/10B GTX Transceiver Wrapper File

Do the following steps to bring out unused transceiver ports to the Aurora 8B/10B <component name>\_gtx\_wrapper.v[hd] file:

1. Go to the directory <component name>/example\_design/gt/ of core 1.
2. Open the <component name>\_gtx\_wrapper.v[hd] file.
3. Search for GTX\_TILE\_INST in <component name>\_gtx\_wrapper.v[hd]
  - a. For a multi-lane Aurora 8B/10B core, search for <component name>\_TILE\_INST\_LANE*n* (*n*=1, 2, 3, . . .)
  - b. Select the instance in which one transceiver is used for the Aurora 8B/10B core and the other is powered down
4. If GTX0 is used for the Aurora 8B/10B core, attributes and ports with a "1" appended are hard coded to the default values.

If GTX1 is used for the Aurora 8B/10B core, attributes and ports with a "0" appended are hard coded to the default values.

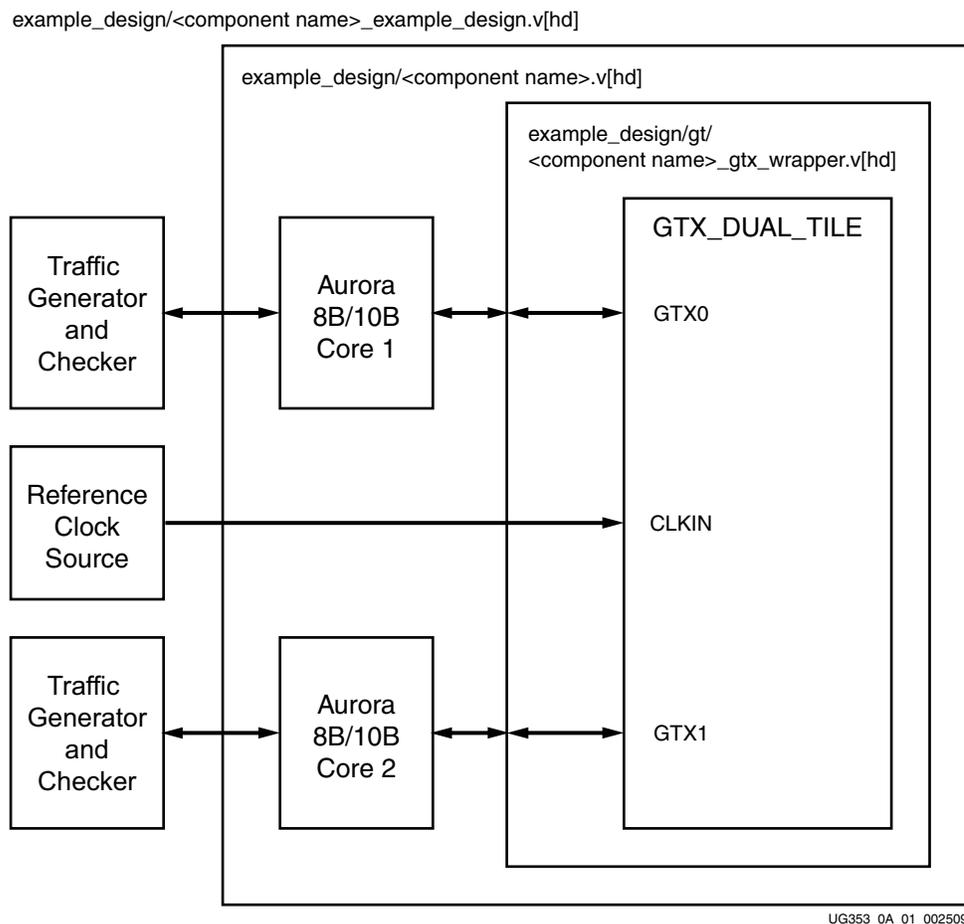
Port map all such hard-coded ports and attributes to new ports and attributes and declare them in the module list (Verilog) or entity list (VHDL).

## Steps to Instantiate Two Aurora 8B/10B Cores in the Top Level File

Use these steps to instantiate two Aurora 8B/10B cores in the top level file:

1. Go to the directory <component name>/example\_design of core 1.
2. Open <component name>.v[hd].
3. Search for gtx\_wrapper\_i instance. Add all new ports and attributes in [step 4, page 102](#) in the port map list.
4. Map these new ports to another Aurora 8B/10B core (see core 2 in [Figure A-1](#)).

[Figure A-1](#) shows two single lane Aurora 8B/10B cores sharing a tile in the Virtex-5 family.



*Figure A-1: Example Showing Two Single Lane Aurora 8B/10B Cores Sharing a Transceiver Tile in the Virtex-5 Family*

**Note:** The same procedure can be followed when a GTP transceiver device is selected. But instead of GTX transceiver references, GTP transceiver should be used in appropriate places.



## Handling Timing Errors

---

When implemented, the Aurora 8B/10B 5.1 core gives timing errors for certain configurations. The design parameters that affect the timing performance are:

- Line rate and transceiver datapath width (2/4 bytes)
- Number of unused transceivers between two selected transceivers

As a result of one or more of the above parameters, timing errors can occur because:

- CHBONDO does not meet timing
- RXCHARISCOMMA, RXCHARISK, and RXCHANISALIGNED do not meet timing
- RX\_SNF/bucket\_full\_r does not meet timing.
- RXRUNDISPERR, RXNOTINTABLEERR, TXDATA0/1, TXBUFSTATUS, RXDATA0/1, RX\_SCP, and reset\_lanes\_flop signals do not meet timing

### Solutions

The following suggestions can be attempted to meet timing.

Select the transceivers consecutively. This can be done by using Lane Assignment in the Aurora 8B/10B GUI during core generation. Most of the timing errors are due to unused transceivers and channel bonding signals connections among transceivers.

- Constrain the critical path signals in UCF:
  - Example:
    - ◆ NET aurora\_module\_i/gtx\_wrapper\_i/chbondo\* MAXDELAY=2.5 ns;
    - ◆ NET aurora\_module\_i/gtx\_wrapper\_i/chbonddone\* MAXDELAY=1.6 ns
- Use another set of USER\_CLK and SYNC\_CLK for two reference clock sources. Aurora 8B10B supports two transceiver reference clock sources. This can be identified as 'Clock Source 1' and 'Clock Source 2' in second page of Aurora 8B10B GUI. Aurora 8B10B core generates with USER\_CLK\_2 and SYNC\_CLK\_2 if Clock source 2 got enabled during core generation. USER\_CLK\_2 and SYNC\_CLK\_2 generation are similar to USER\_CLK and SYNC\_CLK and their generation can be found in CLOCK\_MODULE under example\_design/clock\_module directory. This provision is meant to meet timing and consumes two extra BUFG. This implementation can be removed if no timing errors are observed in the core to save BUFG resources.
- Follow timing optimization mode in ISE implementation. The following modifications are required:
  - ◆ Add these lines in xst.scr file in implement directory.
    - -opt\_mode Speed
    - -register\_balancing Yes

- -iob false
- ◆ Add the following extra switches to map and par commands in `implement.sh` and/or `implement.bat` file(s) presented in `implement` directory
  - -ol high -xe n
- Use `smartexplorer` which comes with ISE software. Refer ISE software documentation on how to use `smartexplorer`.
- In the worst case, reduce the line rate. This results in lower `USER_CLK` and `SYNC_CLK`.

# Performance and Core Latency

## Introduction

Latency through an Aurora 8B/10B core is caused by pipeline delays through the protocol engine (PE) and through the GTP/GTX transceivers. The PE pipeline delay increases as the LocalLink interface width increases. The GTP/GTX transceivers delays are fixed per the features of the GTP/GTX transceivers.

This section outlines expected latency for the Aurora 8B/10B core's LocalLink user interface in terms of USER\_CLK cycles for 2-byte-per-lane and 4-byte-per-lane designs. For the purposes of illustrating latency, the Aurora 8B/10B modules partitioned into GTP/GTX transceivers logic and protocol engine (PE) logic implemented in the FPGA fabric.

**Note:** These figures do not include the latency incurred due to the length of the serial connection between each side of the Aurora 8B/10B channel.

## Latency of the Frame path

Figure C-1 illustrates the latency of the frame path.

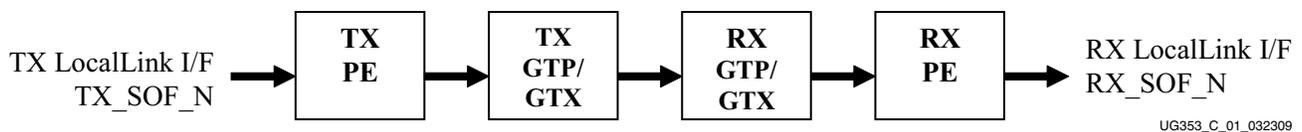


Figure C-1: Latency of the Frame Path

Maximum latency for a 2-byte designs from TX\_SOF\_N to RX\_SOF\_N is approximately 52 GTP/GTX transceivers clock cycles in simulation.

Maximum latency for a 4-byte designs from TX\_SOF\_N to RX\_SOF\_N is approximately 61 GTP/GTX transceivers clock cycles in simulation.

The pipeline delays are designed to maintain the clock speed.



# Generating a Virtex-5 FPGA GTP/GTX Wrapper File from the RocketIO GTP/GTX Transceiver Wizard

---

The transceiver attributes play a vital role in the functionality of the Aurora 8B/10B core. It is important to have the transceiver wrapper file updated. One way to achieve this is to generate the transceiver wrapper file from latest RocketIO™ GTP/ GTX Transceiver Wizards. The following steps explain the procedure to generate the transceiver wrapper file using the RocketIO GTP Wizard and/or the RocketIO GTX Wizard.

1. Using CORE Generator™ tool, run the latest version of RocketIO GTP/GTX Transceiver Wizard. Make the component name match the component name of the Aurora 8B/10B core.
2. Select transceivers and the clock source based on application requirement.
3. Select internal data width as 20 bits (for GTP datapath width is 10) for the Aurora 8B/10B protocol.
4. Select the protocol template based on the number of lanes
5. Select TX encoding as 8B/10B and data path width as 16/32. The same settings should be selected for RX.
6. Select the TXBUFSTATUS and RXCHARISCOMMA ports
7. Keep all other settings as default. Generate the core.
8. Replace the <component name>\_tile.v[hd] file in the example\_design/gt directory available in the Aurora 8B/10B core with the generated <component name>\_tile.v[hd] file from the RocketIO GTP/GTX transceiver Wizard.

The transceiver tile settings for Aurora 8B/10B core are up to date now.

**Note:** The same procedure can be followed for Virtex®-6 FPGA GTX transceiver and Spartan®-6 FPGA GTP transceiver. The Virtex-6 FPGA GTX Transceiver Wizard or the Spartan-6 FPGA GTP Transceiver Wizard should be used to generate the Virtex-6 FPGA GTX or Spartan-6 FPGA transceiver wrapper files respectively.