

# Virtex-5 GTP Aurora v2.8

## *User Guide*

UG224 (v1.4) October 10, 2007



---

Xilinx is disclosing this Document and Intellectual Property (hereinafter “the Design”) to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the Design 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. Any unauthorized use of the Design may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Design; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Design. Xilinx reserves the right to make changes, at any time, to the Design as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Design.

THE DESIGN IS PROVIDED “AS IS” WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DESIGN, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE DESIGN, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE DESIGN, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREUNDER FOR USE OF THE DESIGN. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems (“High-Risk Applications”). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.

© 2006-2007 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. 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
11/30/06	1.1	Initial Xilinx release.
03/01/07	1.2	LogiCORE GTP Aurora v2.6 release. LogiCORE GTP Aurora v2.7 release. <ul style="list-style-type: none"><li>• Replaced <i>sample</i> design with <i>example</i> design throughout for clarity</li><li>• Updated <a href="#">Figure90 TDTc0 307.570 TD-0.0003 Tc-, Figur.</a></li></ul>
05/17/07	1.3	

# Table of Contents

---

<b>Schedule of Figures</b>	7
<b>Schedule of Tables</b>	9
<b>Preface: About This Guide</b>	
Contents	11
Additional Resources	12
Conventions	12
Typographical	12
Online Document	13
<b>Chapter 1: Introduction</b>	
About the Core	15
Recommended Design Experience	15
Related Xilinx Documents	16
Additional Core Resources	16
Technical Support	16
Feedback	16
Core	16
Document	17
<b>Chapter 2: Installing and Licensing the Core</b>	
Before You Begin	19
System Requirements	19
Installing the Core	20
Automated Installation Using WebUpdate	20
Manual Installation	20
Obtaining Your License	21
Installing Your License File	21
<b>Chapter 3: Customizing the Aurora Core</b>	
Introduction	23
Using the IP Customizer	23
Page 1 of the IP Customizer	23
Page 2 of the IP Customizer	25
Page 3 of the IP Customizer	26
Aurora Project Directory Structure	27
Using the Build Script	28
Example Design Overview	29
Using the Testbench and Simulation Scripts	31

## Chapter 4: User Interface

<b>Introduction</b>	33
<b>Framing Interface</b>	34
LocalLink TX Ports	34
LocalLink RX Ports	35
LocalLink Bit Ordering	35
Transmitting Data	36
Receiving Data	41
Framing Efficiency	42
<b>Streaming Interface</b>	45
Streaming TX Ports	45
Streaming RX Ports	45
Transmitting and Receiving Data	46

## Chapter 5: Flow Control

<b>Introduction</b>	47
<b>Native Flow Control</b>	48
Example A: Transmitting an NFC Message	49
Example B: Receiving a Message with NFC Idles Inserted	49
<b>User Flow Control</b>	50
Transmitting UFC Messages	51
Receiving User Flow Control Messages	54
Example D: Receiving a Multi-Cycle UFC Message	54

## Chapter 6: Status, Control, and the GTP Block Interface

<b>Introduction</b>	55
<b>Full-Duplex Cores</b>	56
Full-Duplex Status and Control Ports	56
Error Signals in Full-Duplex Cores	57
Full-Duplex Initialization	59
<b>Simplex Cores</b>	59
Simplex TX Status and Control Ports	59
Simplex RX Status and Control Ports	61
Simplex Both (RX and TX) Status and Control Ports	62
Error Signals in Simplex Cores	64
Simplex Initialization	65
<b>Reset and Power Down</b>	67
Reset	67
Power Down	67
Timing	67

## **Chapter 7: Clock Interface and Clocking**

<b>Introduction</b> .....	69
<b>Clock Interface Ports for GTP Transceiver Cores</b> .....	70
<b>Reference Clocks for GTP Transceiver Designs</b> .....	71
Clocking from Inside the FPGA: GREFCLK .....	71
Clocking from an External Source .....	71
Clocking from a Neighboring GTP_DUAL Tile .....	72
<b>Clock Rates for GTP Transceiver Designs</b> .....	73

## **Chapter 8: Clock Compensation**

<b>Introduction</b> .....	75
<b>Clock Compensation Interface</b> .....	76



# Schedule of Figures

---

## Preface: About This Guide

## Chapter 1: Introduction

## Chapter 2: Installing and Licensing the Core

## Chapter 3: Customizing the Aurora Core

<i>Figure 3-1: Aurora IP Customizer, Page 1</i> .....	23
<i>Figure 3-2: Aurora IP Customizer, Page 2</i> .....	25
<i>Figure 3-3: Aurora IP Customizer, Page 3</i> .....	26
<i>Figure 3-4: Example Design</i> .....	29

## Chapter 4: User Interface

<i>Figure 4-1: Top-Level User Interface</i> .....	33
<i>Figure 4-2: Aurora Core Framing Interface (LocalLink)</i> .....	34
<i>Figure 4-3: LocalLink Interface Bit Ordering</i> .....	35
<i>Figure 4-4: Simple Data Transfer</i> .....	37
<i>Figure 4-5: Data Transfer with Pad</i> .....	38
<i>Figure 4-6: Data Transfer with Pause</i> .....	38
<i>Figure 4-7: Data Transfer Paused by Clock Compensation</i> .....	39
<i>Figure 4-8: Transmitting Data</i> .....	40
<i>Figure 4-9: Data Reception with Pause</i> .....	41
<i>Figure 4-10: Receiving Data</i> .....	42
<i>Figure 4-11: Formula for Calculating Overhead</i> .....	43
<i>Figure 4-12: Aurora Core Streaming User Interface</i> .....	45
<i>Figure 4-13: Typical Streaming Data Transfer</i> .....	46
<i>Figure 4-14: Typical Data Reception</i> .....	46

## Chapter 5: Flow Control

<i>Figure 5-1: Top-Level Flow Control</i> .....	47
<i>Figure 5-2: Transmitting an NFC Message</i> .....	49
<i>Figure 5-3: Transmitting a Message with NFC Idles Inserted</i> .....	49
<i>Figure 5-4: Data Switching Circuit</i> .....	51
<i>Figure 5-5: Transmitting a Single-Cycle UFC Message</i> .....	53
<i>Figure 5-6: Transmitting a Multi-Cycle UFC Message</i> .....	53
<i>Figure 5-7: Receiving a Single-Cycle UFC Message</i> .....	54
<i>Figure 5-8: Receiving a Multi-Cycle UFC Message</i> .....	54

## Chapter 6: Status, Control, and the GTP Block Interface

<i>Figure 6-1: Top-Level GTP Block Interface</i> . . . . .	55
<i>Figure 6-2: Status and Control Interface for Full-Duplex Cores</i> . . . . .	56
<i>Figure 6-3: Status and Control Interface for Simplex TX Core</i> . . . . .	59
<i>Figure 6-4: Status and Control Interface for Simplex RX Core</i> . . . . .	61
<i>Figure 6-5: Status and Control Interface for Simplex Both Cores</i> . . . . .	62
<i>Figure 6-6: Reset and Power Down Timing</i> . . . . .	67

## Chapter 7: Clock Interface and Clocking

<i>Figure 7-1: Top-Level Clocking</i> . . . . .	69
<i>Figure 7-2: Single GTP_DUAL Tile Clocked from the FPGA</i> . . . . .	71
<i>Figure 7-3: Single GTP_DUAL Tile Clocked Externally</i> . . . . .	71
<i>Figure 7-4: Multiple Tiles with a Shared Reference Clock</i> . . . . .	72

## Chapter 8: Clock Compensation

<i>Figure 8-1: Top-Level Clock Compensation</i> . . . . .	75
<i>Figure 8-2: Streaming Data with Clock Compensation Inserted</i> . . . . .	76
<i>Figure 8-3: Data Reception Interrupted by Clock Compensation</i> . . . . .	76



# Schedule of Tables

---

## Preface: About This Guide

## Chapter 1: Introduction

## Chapter 2: Installing and Licensing the Core

## Chapter 3: Customizing the Aurora Core

Table 3-1: Build Script Options .....	28
---------------------------------------	----

Table 3-2: Example Design I/O Ports .....	29
---	----

## Chapter 4: User Interface

Table 4-1: LocalLink User I/O Ports (TX) .....	34
--	----

Table 4-2: LocalLink User I/O Ports (RX) .....	35
--	----

Table 4-3: TX Data Remainder Values .....	36
---	----

Table 4-4: Typical Channel Frame .....	37
--	----

Table 4-5: Efficiency Example .....	43
-------------------------------------	----

Table 4-6: Typical Overhead for Transmitting 256 Data Bytes .....	44
---	----

Table 4-7: TX_REM Value and Corresponding Bytes of Overhead .....	44
---	----

Table 4-8: Streaming User I/O Ports (TX) .....	45
--	----

Table 4-9: Streaming User I/O Ports (RX) .....	45
--	----

## Chapter 5: Flow Control

Table 5-1: NFC Codes .....	48
----------------------------	----

Table 5-2: NFC I/O Ports .....	48
--------------------------------	----

Table 5-3: UFC I/O Ports .....	50
--------------------------------	----

Table 5-4: SIZE Encoding .....	51
--------------------------------	----

Table 5-5: Number of Data Beats Required to Transmit UFC Messages .....	52
---	----

## Chapter 6: Status, Control, and the GTP Block Interface

Table 6-1: Status and Control Ports for Full-Duplex Cores .....	56
---	----

Table 6-2: Error Signals in Full-Duplex Cores .....	58
---	----

Table 6-3: Status and Control Ports for Simplex TX Cores .....	60
--	----

Table 6-4: Status and Control Ports for Simplex RX Cores .....	61
--	----

Table 6-5: Status and Control Ports for Simplex "Both" Cores .....	63
--	----

Table 6-6: Error Signals in Simplex Cores .....	65
---	----

## Chapter 7: Clock Interface and Clocking

Table 7-1: Clock Ports for a GTP Aurora Core .....	70
--	----

## Chapter 8: Clock Compensation

<i>Table 8-1: Clock Compensation I/O Ports</i> .....	76
<i>Table 8-2: Clock Compensation Cycles</i> .....	77
<i>Table 8-3: Lookahead Cycles</i> .....	77
<i>Table 8-4: Standard CC I/O Port.</i> .....	78

# About This Guide

---

This user guide describes the function and operation of the LogiCORE™ GTP Aurora core for Virtex™-5 FPGAs, 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, “Installing and Licensing the Core”](#) provides information about installing and licensing the core.
- [Chapter 3, “Customizing the Aurora Core”](#) describes how to customize an Aurora core with the available parameters.
- [Chapter 4, “User Interface”](#) provides port descriptions for the user interface.
- [Chapter 5, “Flow Control”](#) describes the user flow control and native flow control options for sending and receiving data.
- [Chapter 6, “Status, Control, and the GTP Block Interface”](#) provides details on using the Aurora core's status and control ports to initialize and monitor the Aurora channel.
- [Chapter 7, “Clock Interface and Clocking”](#) describes how to connect FPGA clocking resources.
- [Chapter 8, “Clock Compensation”](#) covers Aurora clock compensation, and explains how to customize it for a given system.

## Additional Resources

For additional information, go to <http://www.xilinx.com/support>. The following table lists some of the resources you can access from this website or by using the provided URLs.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging <a href="http://www.xilinx.com/support/techsup/tutorials/index.htm">http://www.xilinx.com/support/techsup/tutorials/index.htm</a>
Answer Browser	Database of Xilinx solution records <a href="http://www.xilinx.com/xlnx/xil_ans_browser.jsp">http://www.xilinx.com/xlnx/xil_ans_browser.jsp</a>
Application Notes	Descriptions of device-specific design techniques and approaches <a href="http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=Application+Notes">http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=Application+Notes</a>
Data Sheets	Device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging <a href="http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp">http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp</a>
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues <a href="http://www.xilinx.com/support/troubleshoot/psolvers.htm">http://www.xilinx.com/support/troubleshoot/psolvers.htm</a>
Tech Tips	Latest news, design tips, and patch information for the Xilinx design environment <a href="http://www.xilinx.com/xlnx/xil_tt_home.jsp">http://www.xilinx.com/xlnx/xil_tt_home.jsp</a>

## 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 you enter in a syntactical statement	<b>ngdbuild</b> <i>design_name</i>
<i>Italic font</i>	Variables in a syntax statement for which you must supply values	<b>ngdbuild</b> <i>design_name</i>
	References to other manuals	See the <i>User Guide</i> for details.
	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

Convention	Meaning or Use	Example
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as <b>bus [7:0]</b> , they are required.	<b>ngdbuild</b> [ <i>option_name</i> ] <i>design_name</i>
Braces { }	A list of items from which you must choose one or more	<b>lowpwr</b> = { <b>on</b>   <b>off</b> }
Vertical bar	Separates items in a list of choices	<b>lowpwr</b> = { <b>on</b>   <b>off</b> }
Vertical ellipsis .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Omitted repetitive material	<b>allow block</b> block_name loc1 loc2 ... locn;
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	usr_teof_n is active low.

## Online Document

The following linking conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section “ <a href="#">Additional Resources</a> ” for details. Refer to “ <a href="#">Title Formats</a> ” in <a href="#">Chapter 1</a> for details.
Red text	Cross-reference link to a location in another document	See <a href="#">Figure 2-5</a> in the <i>Virtex-II Handbook</i> .
<a href="#">Blue, underlined text</a>	Hyperlink to a website (URL)	Go to <a href="http://www.xilinx.com">http://www.xilinx.com</a> for the latest speed files.

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section “ <a href="#">Additional Resources</a> ” for details. Refer to “ <a href="#">Title Formats</a> ” in <a href="#">Chapter 1</a> for details.
Red text	Cross-reference link to a location in another document	See <a href="#">Figure 2-5</a> in the <i>Virtex-II Platform FPGA User Guide</i> .
<a href="#">Blue, underlined text</a>	Hyperlink to a website (URL)	Go to <a href="http://www.xilinx.com">http://www.xilinx.com</a> for the latest speed files.



## Introduction

---

This chapter introduces the Aurora core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.

The LogiCORE GTP Aurora v2.8 core is a high-speed serial solution based on the Aurora protocol and Virtex-5 RocketIO™ 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 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 <http://www.xilinx.com/aurora>. For information about system requirements, installation, and licensing options, see [Chapter 2, “Installing and Licensing the Core.”](#)

### 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 6, “Status, Control, and the GTP Block Interface”](#) carefully, and consult the PCB design requirements information in the *Virtex-5 RocketIO GTP Transceiver 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:

- Documents located on the Aurora product page: <http://www.xilinx.com/aurora>
  - ♦ SP002: *Aurora Protocol Specification*
  - ♦ UG058: *Aurora Bus Functional Model User Guide*
- Documents located on the LocalLink product page: <http://www.xilinx.com/locallink>
  - ♦ SP006: *LocalLink Interface Specification*
- [UG196](#): *Virtex-5 RocketIO GTP Transceiver User Guide*
- ISE™ documentation  
[http://www.xilinx.com/support/sw\\_manuals/xilinx9/index.htm](http://www.xilinx.com/support/sw_manuals/xilinx9/index.htm)

## Additional Core Resources

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

- DS538: *(LogiCORE) Virtex-5 GTP Aurora v2.8 Data Sheet*
- UG223: *Virtex-5 GTP Aurora v2.8 Getting Started Guide*
- UG224: *Virtex-5 GTP Aurora v2.8 User Guide*
- GTP Aurora core Release Notes

## 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 GTP Aurora v2.8 license (see [Chapter 2, “Installing and Licensing the Core”](#)). Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow the guidelines in *Virtex-5 GTP Aurora v2.8 User Guide* and the *Virtex-5 GTP Aurora v2.8 Getting Started Guide*.

## Feedback

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

### Core

For comments or suggestions about the Aurora core, please submit a WebCase from [www.xilinx.com/support](http://www.xilinx.com/support). 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 this document, please submit a WebCase from [www.xilinx.com/support](http://www.xilinx.com/support). Be sure to include the following information:

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



# Installing and Licensing the Core

---

This chapter provides instructions for installing the Aurora core in the CORE Generator tool and how to obtain a free license to use the core.

## Before You Begin

Before installing the Wizard, you must have a MySupport account and the ISE 9.2i SP3 software installed on your system. If you have already completed these steps, go to [“Installing the Core,” page 20](#), otherwise, do the following:

1. Click **Login** at the top of the Xilinx home page, then follow the onscreen instructions to create a MySupport account.
2. Install the ISE 9.2i software and the applicable Service Pack. ISE Service Packs can be downloaded from [www.xilinx.com/support/download.htm](http://www.xilinx.com/support/download.htm)

## System Requirements

### Windows

- Windows XP Professional SP1, SP2, 32-bit, 64-bit
- Windows Vista Business 32-bit

### Solaris

- Sun Solaris 9, 10

### Linux

- Red Hat Enterprise WS 3.0/4.0/5.0 (32-bit or 64-bit)

### Software

- ISE 9.2i with applicable Service Pack

Check the release notes for the required Service Pack; ISE Service Packs can be downloaded from [http://www.xilinx.com/support/sw\\_manuals/xilinx9/index.htm](http://www.xilinx.com/support/sw_manuals/xilinx9/index.htm)

## Installing the Core

You can install the Wizard in two ways: using the CORE Generator WebUpdate facility, which lets you select from a list of updates, or by performing a manual installation after downloading the core from the web.

### Automated Installation Using WebUpdate

**Note:** To use this installation method behind a firewall, you must know your proxy settings. If necessary, contact your administrator to determine the proxy host address and port number before you begin.

1. From the main CORE Generator window, choose **Tools** → **Software Update...**
2. Click **OK** to close the CORE Generator tool and start WebUpdate.
3. If necessary, click **Advanced . . .** to specify a proxy host.
4. Click **Check for Updates**.
5. Ensure ISE 9.2i IP Update 2 is selected in the list of software updates.
6. Click **OK**.
7. WebUpdate downloads and installs the selected software updates. Restart your computer when the install is finished.
8. To confirm the installation, from the main CORE Generator window, choose **Help** → **About Xilinx CORE Generator**.
9. Look for the following lines in the About dialog box:  
Updates installed:  
ISE 9.2i IP Update 2

### Manual Installation

1. Close the CORE Generator application if it is running.
2. Download the ZIP file from the following location and save it to a temporary directory:  
[http://www.xilinx.com/xlnx/xil\\_sw\\_updates\\_home.jsp?update=ip&software=9.2i](http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp?update=ip&software=9.2i)  
**Note:** Before you can access this page and the files listed on it, you must be registered for CORE Generator IP Updates access.
3. Extract the ZIP archive file `ise_92i_ip_update2.zip` to a temporary location.

#### For Windows

- ♦ Extract the ZIP archive file using WinZip 7.0 SR-1 or later.

**Note:** When extracting the files using WinZip, you must check the **Use Folder Names** option.

#### For UNIX

- ♦ Extract the ZIP archive file using `unzip`.

**Note:** You might need system administrator privileges to install the update.

4. In the root level of the extracted directory structure, run the `setup (.exe)` executable to install the update.
5. Restart the CORE Generator tool; it automatically detects and displays the newly installed IP cores.
6. Determine whether the installation was successful by verifying that the new cores are visible in the main CORE Generator window.

## Obtaining Your License

To obtain your license for the Aurora core, perform the following steps:

- Navigate to the Aurora product page: <http://www.xilinx.com/aurora>
- Click the Aurora LogiCORE link at the bottom of the page
- Click **Order and Register**

Follow the onscreen instructions to review and electronically sign the Aurora License Agreement and download your license file for the Aurora core.

## Installing Your License File

After selecting a license option, an email will be sent to you that includes instructions for installing your license file. In addition, information about advanced licensing options and technical support is provided.



## Customizing the Aurora Core

### Introduction

The Aurora core can be customized to suit a wide variety of requirements using the CORE Generator tool. 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 IP Customizer is presented when the user selects the Aurora core in the CORE Generator tool. For help starting and using the CORE Generator tool, see the *CORE Generator Help* in the ISE documentation. Each numbered item in [Figure 3-1](#), [Figure 3-2](#), and [Figure 3-3](#) corresponds to a section that describes the purpose of the feature.

#### Page 1 of the IP Customizer

[Figure 3-1](#) shows Page 1 of the customizer. The left side displays a representative block diagram of the Aurora core as currently configured. The right side consists of user-configurable parameters.

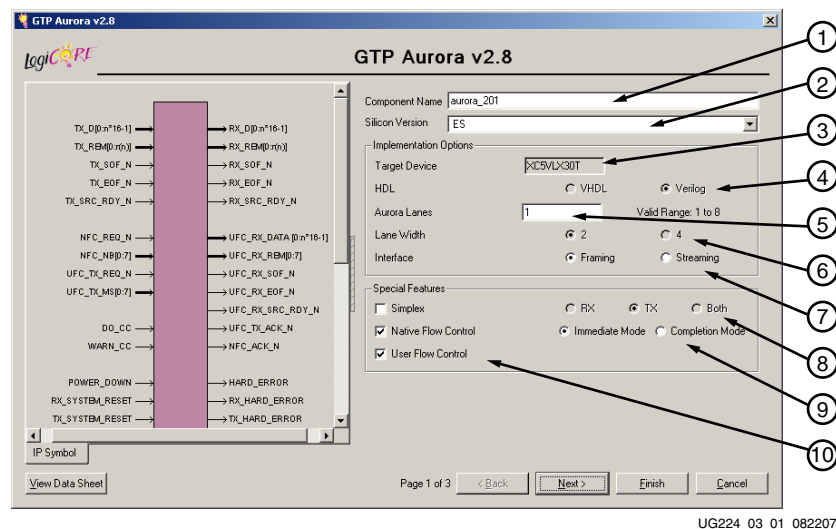


Figure 3-1: Aurora IP Customizer, Page 1

## 1: Component Name

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

**Default:** aurora\_201

## 2: Silicon Version

Select the respective silicon version for the target device.

- ES - Engineering sample.
- PRODUCTION - Production silicon.

**Default:** ES

## 3: Target Device

The device selected during project creation is displayed here.

## 4: HDL

The language selected during project creation is displayed here.

## 5: Aurora Lanes

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

**Default:** 1

## 6: Lane Width

The byte width of the GTP transceivers used in the core is fixed at 2.

## 7: Interface

Use these buttons to 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 channel. See [Chapter 4, “User Interface”](#) for more information.

**Default:** Framing

## 8: Simplex

Check this box to create a simplex core. Simplex Aurora cores have a single, unidirectional serial port that connects to a complementary simplex Aurora core. Three buttons representing the mode of the simplex core become active when the simplex checkbox is activated: RX only, TX only, or Both. Use these buttons to select the direction of the channel the Aurora core will support. Both creates two cores, one RX and one TX, that share a single GTP transceiver. See [Chapter 6, “Status, Control, and the GTP Block Interface”](#) for more information.

**Default:** Not checked (full-duplex mode)



## 9: Native Flow Control

Check this box to add native flow control to the core. Native flow control allows full-duplex receivers to regulate the rate of the data sent to them. Two buttons represent the native flow control mode, *Immediate Mode* or *Completion Mode*, for the core. These buttons are selectable only when Native Flow Control is checked. Immediate mode allows idle codes to be inserted within data frames while completion mode only inserts idle codes between complete data frames. See [Chapter 5, “Flow Control”](#) for more information.

**Default:** Immediate Mode

## 10: User Flow Control

Check this box to add user flow control to the core. User flow control allows applications to send each other brief, high-priority messages through the Aurora channel. See [Chapter 5, “Flow Control”](#) for more information.

**Default:** Not checked

## Page 2 of the IP Customizer

Figure 3-2 shows Page 2 of the customizer.

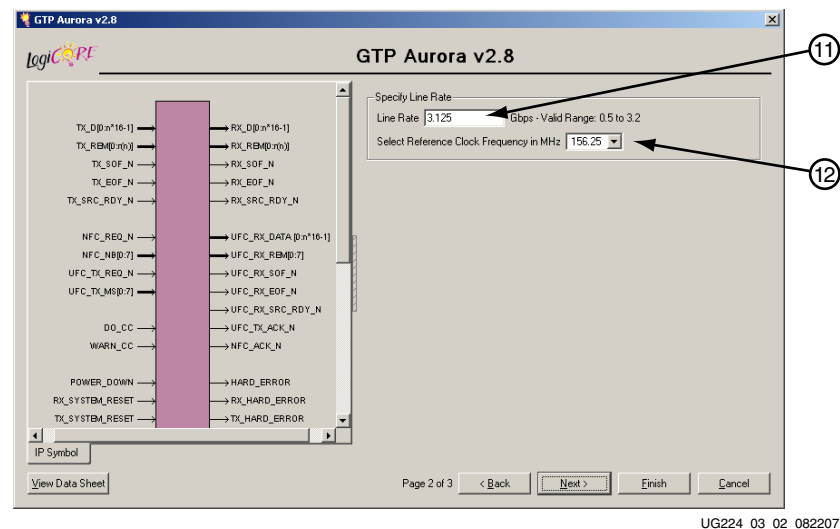


Figure 3-2: Aurora IP Customizer, Page 2

## 11: Line Rate

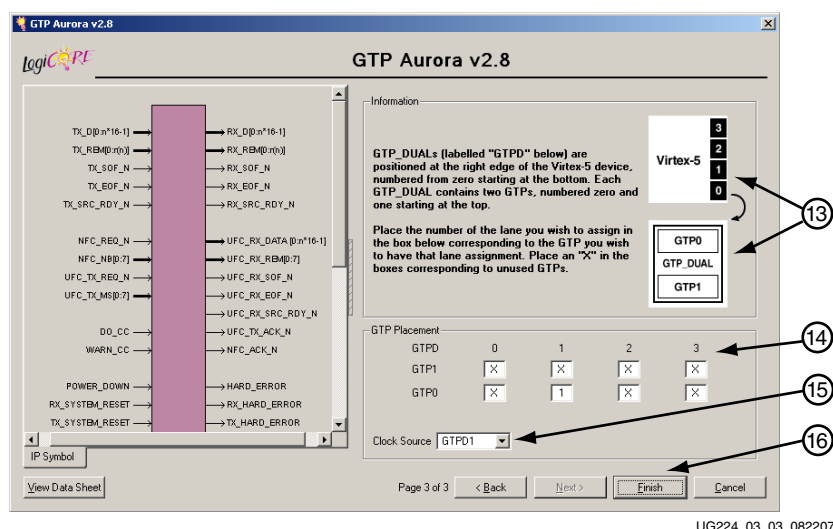
Enter a floating-point value in gigabits per second. The value entered must be within the valid range shown. This determines the un-encoded bit rate at which data will be transferred over the serial link. The aggregate data rate of the core is  $(0.8 \times \text{line rate}) \times \text{Aurora lanes}$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

## 12: Reference Clock Frequency

Select a reference clock frequency from the drop-down list. Reference clock frequencies are given in megahertz, 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.

## Page 3 of the IP Customizer

Figure 3-3 shows Page 3 of the customizer.



UG224\_03\_03\_082207

Figure 3-3: Aurora IP Customizer, Page 3

### 13: Information

This section details the layout of the GTP\_DUALs for the target device. Refer to the diagram displayed in the information area in Figure 3-3 when selecting the lane placement.

This area also includes a diagram of the details of a single GTP\_DUAL showing the GTP transceivers inside of it.

### 14: Lane Placement

Refer to the diagram in the information area in Figure 3-3. Each numbered column represents a GTP\_DUAL and each active box represents an available GTP transceiver. For each Aurora Lane in the core, starting with Lane 1, select a GTP transceiver and place the lane by typing its number in the GTP Placement box. If there are fewer lanes in this instance of the core than there are available GTP transceivers, put an X in the boxes for all the unused GTP transceivers.

### 15: Clock Source

Select a reference clock source for the GTP\_DUALs from the drop-down list in this section.

### 16: Finish Button

Click **Finish** to generate the core. The modules for the Aurora core are written to the CORE Generator project directory using the same name as the top level of the core. See “Aurora Project Directory Structure,” page 27.

## Aurora Project Directory Structure

The customized Aurora core is delivered as a set of HDL source modules in the selected language with supporting script and documentation files. These files are arranged in a predetermined directory structure under the project directory name provided to the CORE Generator system when the project is created, as shown below.

```

<top-level name>
|__readme.txt
|__aurora_gtp_ug224.pdf (user guide)
|__aurora_gtp_gsug223.pdf (getting started guide)
|__src
|   |__(top-level source file)
|   |__(remaining Aurora module source files)
|__ucf
|   |__<top-level name>.ucf (Aurora module design constraints)
|   |__aurora_example.ucf (Aurora example design constraints)
|__examples
|   |__(aurora_example source file)
|   |__(remaining source files for submodules used in the example
|       design, not including the cc_manager or the clock module)
|__scripts
|   |__(make_aurora build script)
|   |__(config setup shell script)
|   |__(simulation script for modelsim)
|__clock_module (optional)
|   |__(clock_module source file)
|__cc_manager (optional)
|   |__(standard_cc_module source file)
|__testbench
|   |__(testbench files for simulating aurora_example design)

```

## Using the Build Script

A PERL script called `make_aurora.pl` is delivered with the Aurora core in the `scripts` subdirectory. The script can be used to ease implementation of the Aurora core. Run the script to synthesize the Aurora core using XST. The script can also be run with the options shown in [Table 3-1](#) to synthesize using Synplify, 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:

```
xilperl make_aurora.pl <options>
```

**Table 3-1: Build Script Options**

Option	Description
<b>-b</b>	For XST synthesis, use this option to generate a bitstream file after place and route.
<b>-files</b>	Use this option to generate synthesis project files for the core without running the synthesis or implementation tools.
<b>-h</b>	Use this option to display a help message for the make script without running any tools.
<b>-m</b>	Use this option to run ngdbuild and map after synthesis is complete.
<b>-p</b>	Use this option to run place and route (par). The <b>-m</b> option is automatically used when <b>-p</b> is selected.
<b>-example</b>	Causes the <code>aurora_example</code> module to be used as the top level. Specify this option when building the example design.
<b>-synplify</b>	Use this option to invoke the Synplicity synthesis tool instead of XST. Specify with the <b>-files</b> option above to create a Synplicity project file without running the tool.
<b>-t</b>	Use this option to generate Trace report.

## Example Design Overview

Each core includes a example design (aurora\_example) that uses the core in a simple data transfer system. 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 3-4 is a block diagram of the example design for a full-duplex core. Table 3-2 describes the ports of the example design.

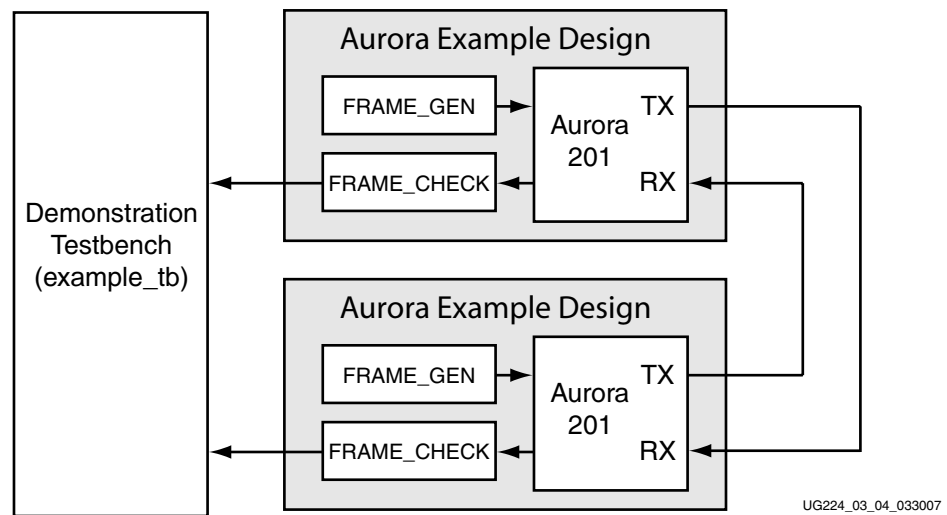


Figure 3-4: Example Design

The example designs uses all the interfaces of the core except for optional flow control. 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 scripts subdirectory, the example design can be used to quickly get an Aurora 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 core, such as the clocking interface.

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

Table 3-2: Example Design I/O Ports

Port	Direction	Description
RXN[0:m-1]	Input	Positive differential serial data input pin.
RNP[0:m-1]	Input	Negative differential serial data input pin.
TXN[0:m-1]	Output	Positive differential serial data output pin.
TXP[0:m-1]	Output	Positive differential serial data output pin.
ERROR_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.

<i>&lt;reference clock(s)&gt;</i>	Input	The reference clocks for the Aurora core are brought to the top level of the example design. See <a href="#">Chapter 7, “Clock Interface and Clocking”</a> for details about the reference clocks.
<i>&lt;core error signals&gt;</i>	Output	The error signals from the Aurora core's Status and Control interface are brought to the top level of the example design and registered. See

## Using the Testbench and Simulation Scripts

A testbench for simulating a pair of example design modules communicating with each other is included in the testbench subdirectory. A script is included in the testbench subdirectory to compile the complete test and prepare a waveform view in ModelSim. For instructions explaining how to run the simulation, see the *Virtex-5 GTP Aurora v2.8 Getting Started Guide*.

Because cores are generated one at a time, simulating simplex cores requires additional steps (except for simplex Both 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 scripts 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 *Virtex-5 GTP Aurora v2.8 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`.





## User Interface

### Introduction

An Aurora 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 5, “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.

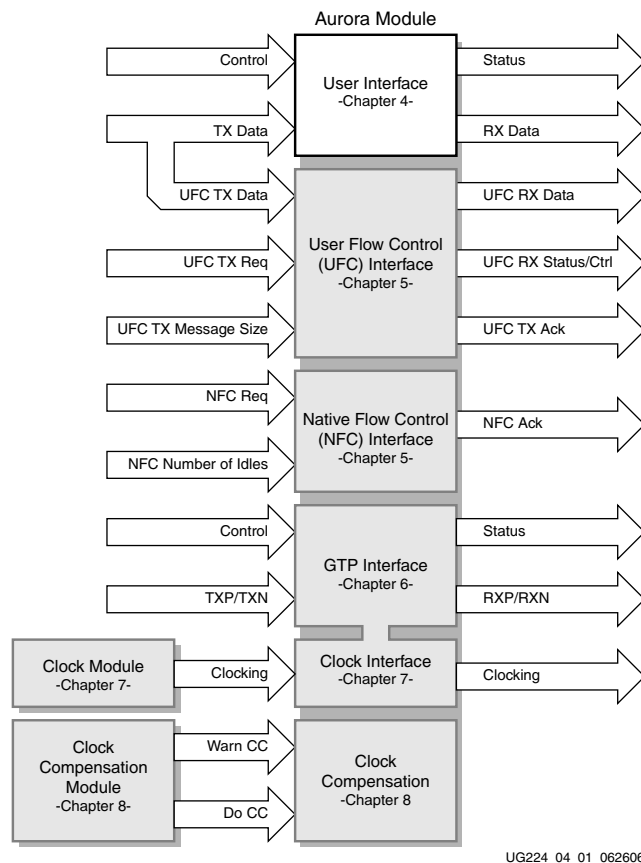


Figure 4-1: Top-Level User Interface

**Note:** The user interface signals vary depending upon the selections made when generating an Aurora core in the CORE Generator tool.

## Framing Interface

Figure 4-2 shows the framing user interface of the Aurora core, with LocalLink-compliant ports for TX and RX data.

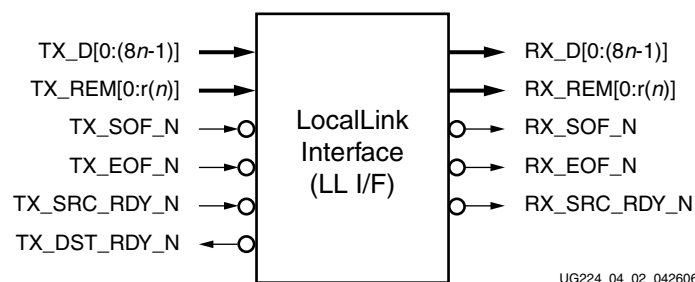


Figure 4-2: Aurora Core Framing Interface (LocalLink)

### LocalLink TX Ports

The tables in this section list port descriptions for LocalLink TX data ports. These ports are included on full-duplex, simplex TX, and simplex Both framing cores.

Table 4-1: LocalLink User I/O Ports (TX)

Name	Direction	Description
TX_D[0:(8n-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_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

The tables in this section list port descriptions for LocalLink RX data ports. These ports are included on full-duplex, simplex RX, and simplex Both framing cores.

**Table 4-2: LocalLink User I/O Ports (RX)**

Name	Direction	Description
RX_D[0:(8n-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).
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).
RX_SRC_RDY_N	Output	Asserted (low) when data and control signals from an Aurora core are valid. Deasserted (high) when data and/or control signals from an Aurora core should be ignored (active-Low).

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

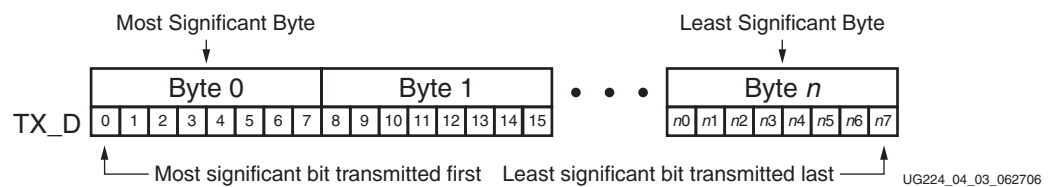
- Take data from the user on the TX\_D bus
- Encapsulate and stripe the data across lanes in the Aurora channel (TX\_SOF\_N, TX\_EOF\_N)
- Pause data (that is, insert idles) (TX\_SRC\_RDY\_N)

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 cores use ascending ordering. They transmit and receive the most significant bit of the most significant byte first. Figure 4-3 shows the organization of an  $n$ -byte example of the LocalLink data interfaces of an Aurora core.



**Figure 4-3: LocalLink Interface Bit Ordering**

UG224\_04\_03\_062706

## Transmitting Data

LocalLink is a synchronous interface. The Aurora 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 to ignore the LocalLink input for that cycle. If this occurs in the middle of a frame, idle symbols are sent through the Aurora 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 that 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 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 4-3](#) shows the mapping between TX\_REM values and valid byte counts for the TX\_D port.

**Table 4-3: TX Data Remainder Values**

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

## Aurora Frames

The TX\_LL submodule translates each user frame that it receives through the TX interface to an Aurora 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 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 4-4](#) shows a typical Aurora 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 core responds with start-of-frame and end-of-frame ordered sets, /SCP/ and /ECP/ respectively, as shown in Table 4-4.

Table 4-4: Typical Channel Frame

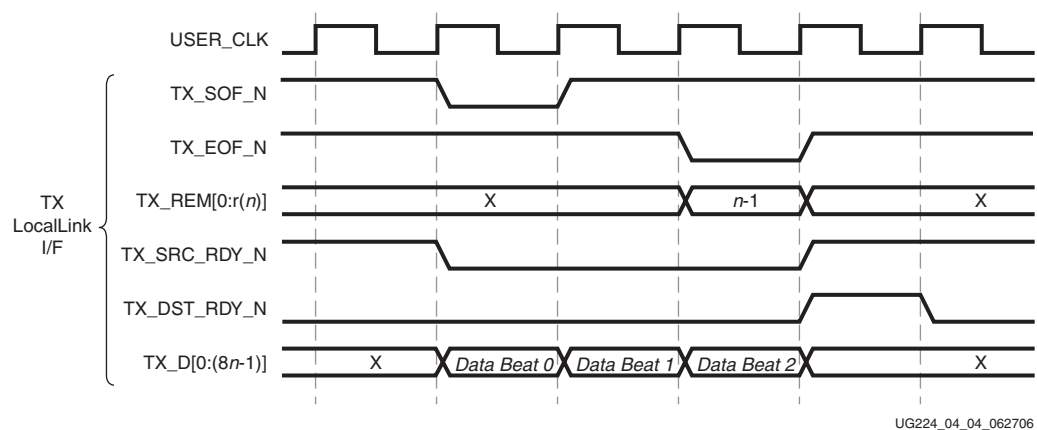
/SCP/ <sub>1</sub>	/SCP/ <sub>2</sub>	Data Byte 0	Data Byte 1	Data Byte 2	...	Data Byte <i>n</i> - 1	Data Byte <i>n</i>	/ECP/ <sub>1</sub>	/ECP/ <sub>2</sub>
--------------------	--------------------	----------------	----------------	----------------	-----	---------------------------	-----------------------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

## Example A: Simple Data Transfer

Figure 4-4 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 core is not sending data, it sends idle sequences.

To begin the data transfer, the user asserts 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 core sends idles.

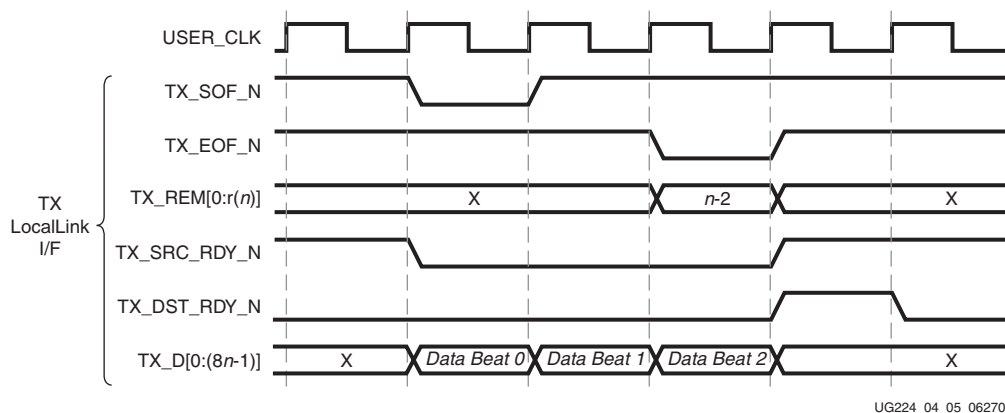


UG224\_04\_04\_062706

Figure 4-4: Simple Data Transfer

### Example B: Data Transfer with Pad

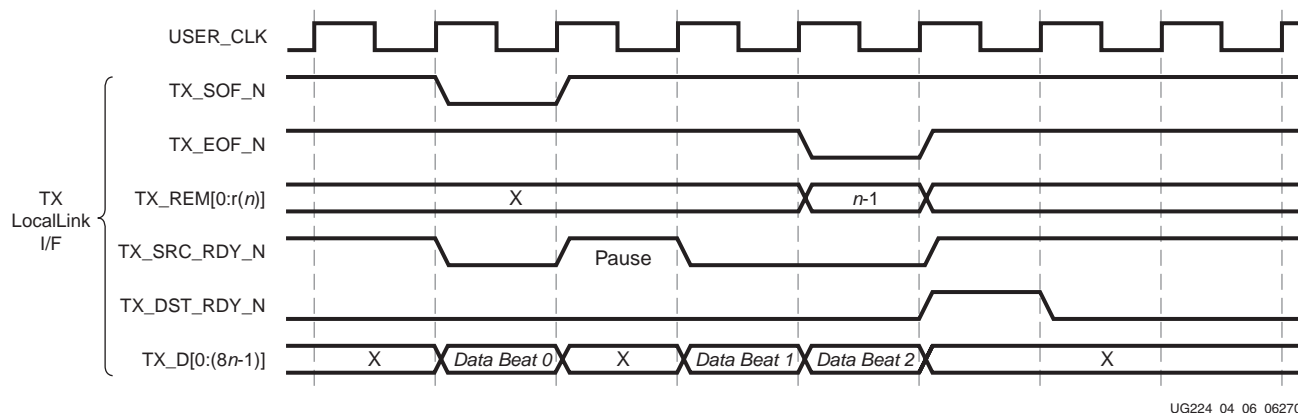
Figure 4-5 shows an example of a  $(3n-1)$ -byte data transfer that requires the use of a pad. Since there is an odd number of data bytes, the Aurora core appends a pad character at the end of the Aurora frame, as required by the protocol. A transfer of  $3n-1$  data bytes requires two full  $n$ -byte 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 4-5: Data Transfer with Pad**

### Example C: Data Transfer with Pause

Figure 4-6 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 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 4-6: Data Transfer with Pause**

## Example D: Data Transfer with Clock Compensation

The Aurora 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 4-7 shows how the Aurora core pauses data transmission during the clock compensation<sup>(1)</sup> sequence.

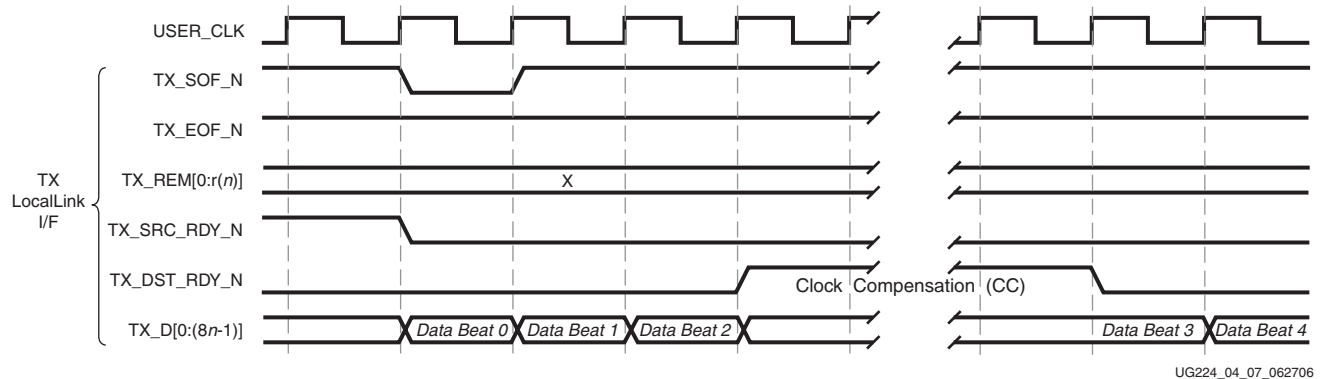


Figure 4-7: Data Transfer Paused by Clock Compensation

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.

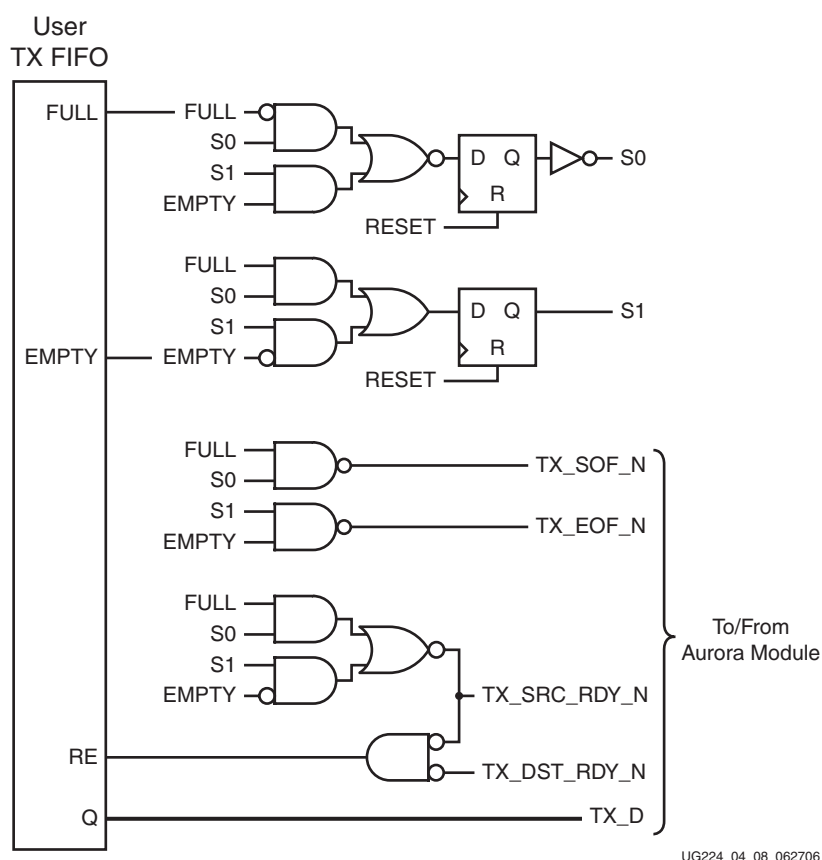
## TX Interface Example

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 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 4-8 is a diagram of a typical connection between an Aurora 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 core to start reading the FIFO by asserting the TX\_DST\_RDY\_N signal.

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



UG224\_04\_08\_062706

Figure 4-8: Transmitting Data



## Receiving Data

When the Aurora core receives an Aurora 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 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 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 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 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 frame.

### Example A: Data Reception with Pause

Figure 4-9 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.

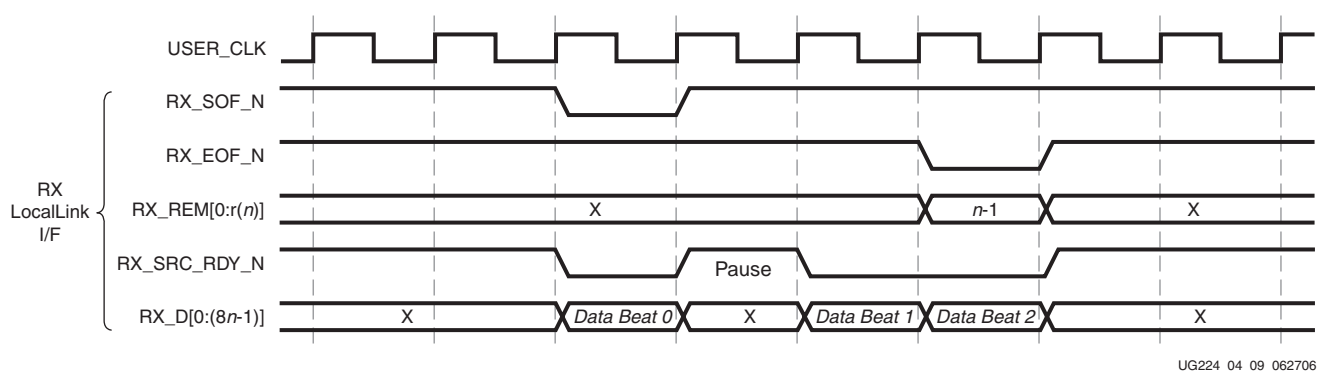


Figure 4-9: Data Reception with Pause

## RX Interface Example

Figure 4-10 is a simple example of how a user might design an interface between the LocalLink interface of an Aurora 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.

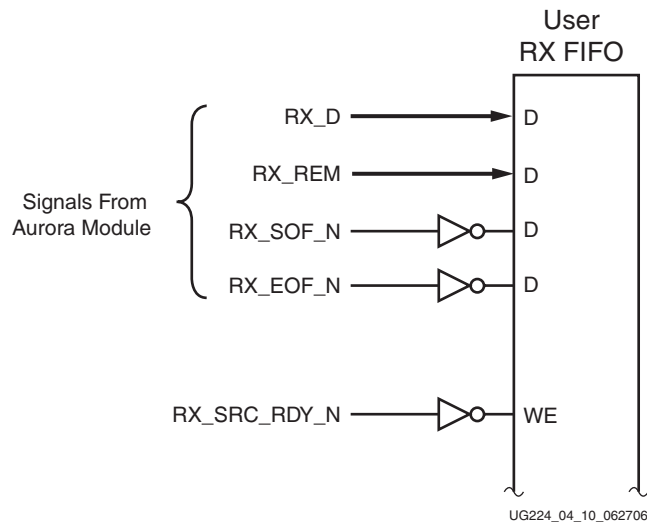


Figure 4-10: Receiving Data

## Framing Efficiency

There are two factors that affect framing efficiency in the Aurora 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 percent of the total channel bandwidth.

All bytes in Aurora are sent in 2-byte code groups. Aurora frames with an even number of bytes have 4 bytes of overhead, 2 bytes for SCP (start of frame) and 2 bytes for ECP (end of frame). Aurora frames with an odd number of bytes have 5 bytes of overhead, 4 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 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 4-11 to calculate the efficiency for a design of any number of lanes, any width of interface, and frames of any number of bytes. Note that 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)

UG224\_04\_11\_062806

Figure 4-11: Formula for Calculating Overhead

## Example

Table 4-5 is an example calculated from the formula given in Figure 4-11. 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 4-5: Efficiency Example

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

Table 4-6 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 percent 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 4-6: 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 4-7 shows the overhead that occurs with each value of TX\_REM.

**Table 4-7: 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 4-12 shows an example of an Aurora core configured with a streaming user interface.

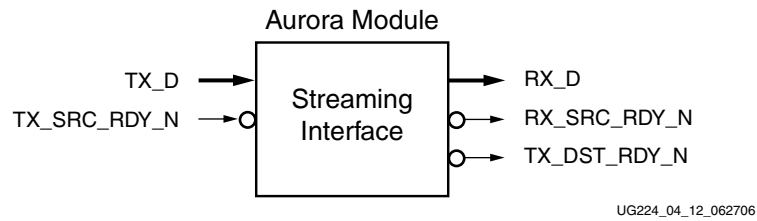


Figure 4-12: Aurora Core Streaming User Interface

### Streaming TX Ports

Table 4-8 lists the streaming TX data ports. These ports are included on full-duplex, simplex TX, and simplex Both framing cores.

Table 4-8: Streaming User I/O Ports (TX)

Name	Direction	Description
TX_D[0:(8n-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 4-9 lists the streaming RX data ports. These ports are included on full-duplex, simplex RX, and simplex Both framing cores.

Table 4-9: Streaming User I/O Ports (RX)

Name	Direction	Description
RX_D[0:(8n-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 core are valid. Deasserted (high) when data and/or control signals from an Aurora core should be ignored (active-Low).

## Transmitting and Receiving Data

The streaming interface allows the Aurora 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 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 transceiver to make up for frequency differences between the two sides of the Aurora channel. As a result, gaps created when DO\_CC is asserted can shrink and grow. For details on the DO\_CC signal, see [Chapter 8, “Clock Compensation.”](#)

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

[Figure 4-13](#) 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 core are not ready to transfer data. During the next clock cycle, the Aurora 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 core. Data D1 is transferred on the following clock cycle. In this example, the Aurora 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.

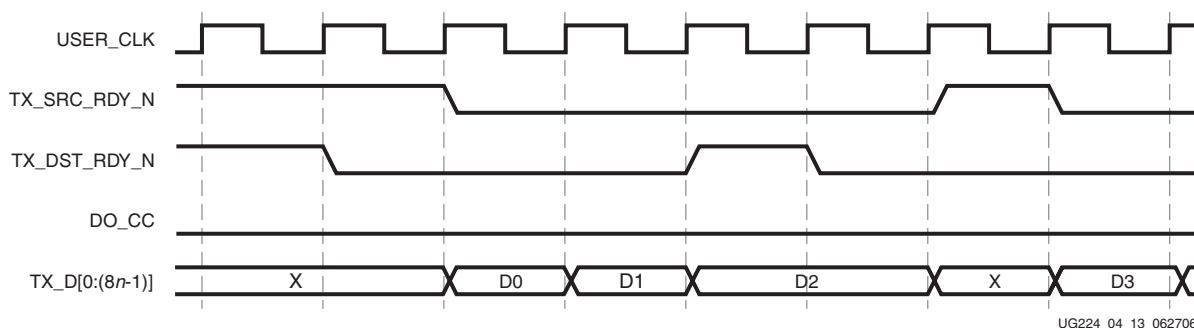


Figure 4-13: Typical Streaming Data Transfer

[Figure 4-14](#) shows the receiving end of the data transfer that is shown in [Figure 4-13](#).

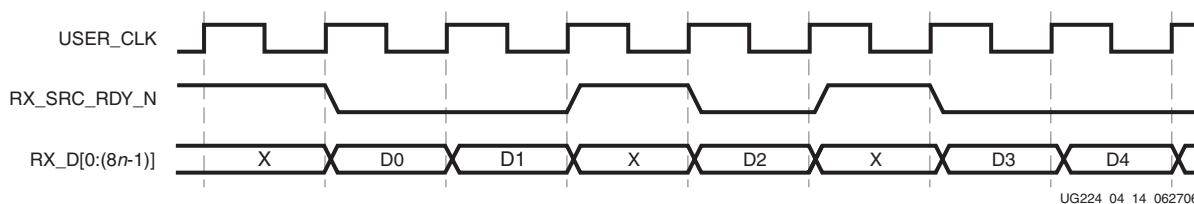


Figure 4-14: Typical Data Reception

## Flow Control

### Introduction

This chapter explains how to use Aurora 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.

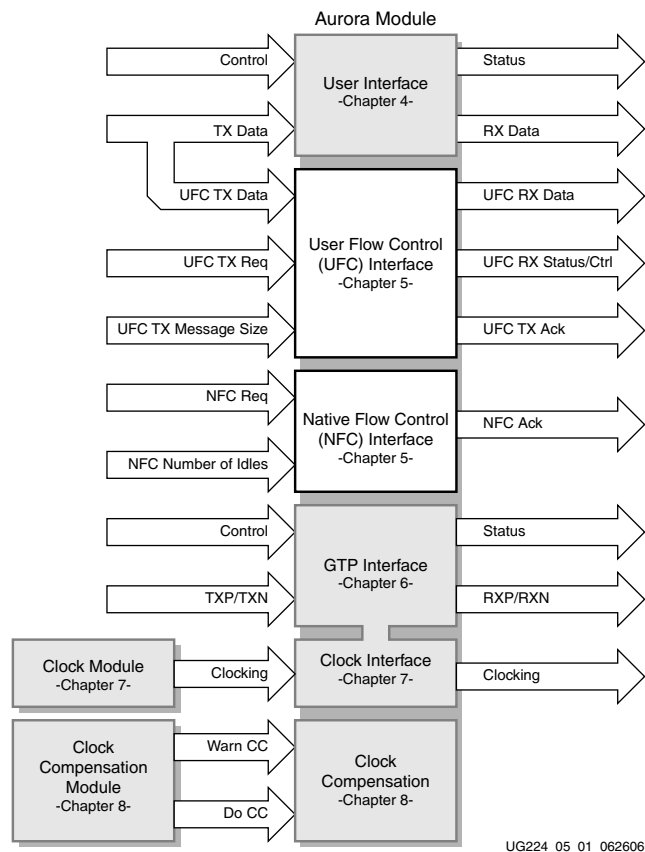


Figure 5-1: Top-Level Flow Control

## Native Flow Control

Table 5-1 shows the codes for native flow control (NFC).

Table 5-1: 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 5-2 lists the ports for the NFC interface available only in full-duplex Aurora cores.

Table 5-2: NFC I/O Ports

Name	Direction	Description
NFC_ACK_N	Output	Asserted when an Aurora 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 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 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 core will transmit the NFC message. Aurora 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 5-2 shows an example of the transmit timing when the user sends an NFC message to a channel partner. Note that 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.

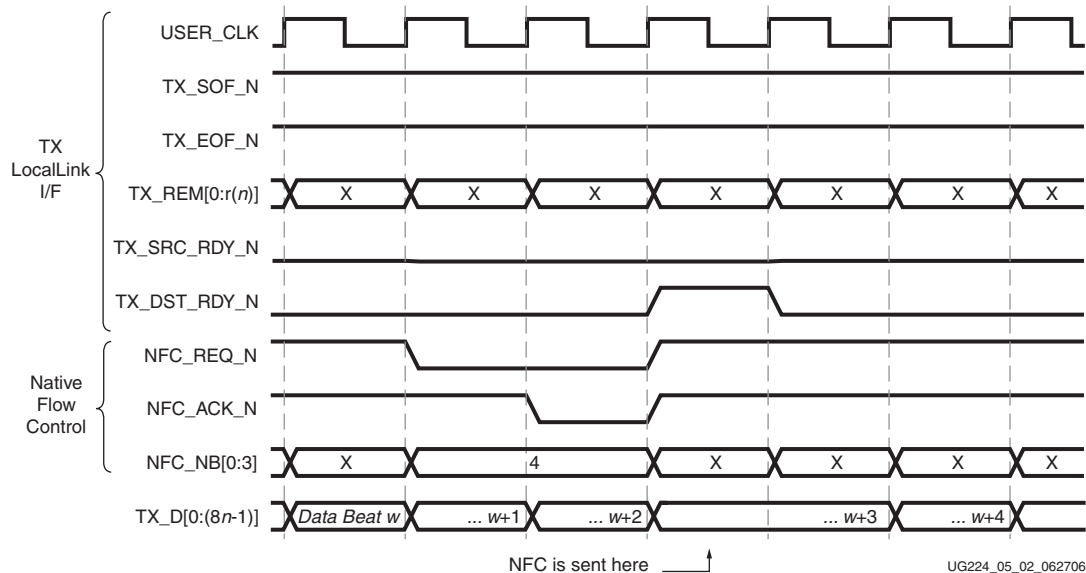


Figure 5-2: Transmitting an NFC Message

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

Figure 5-3 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 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.

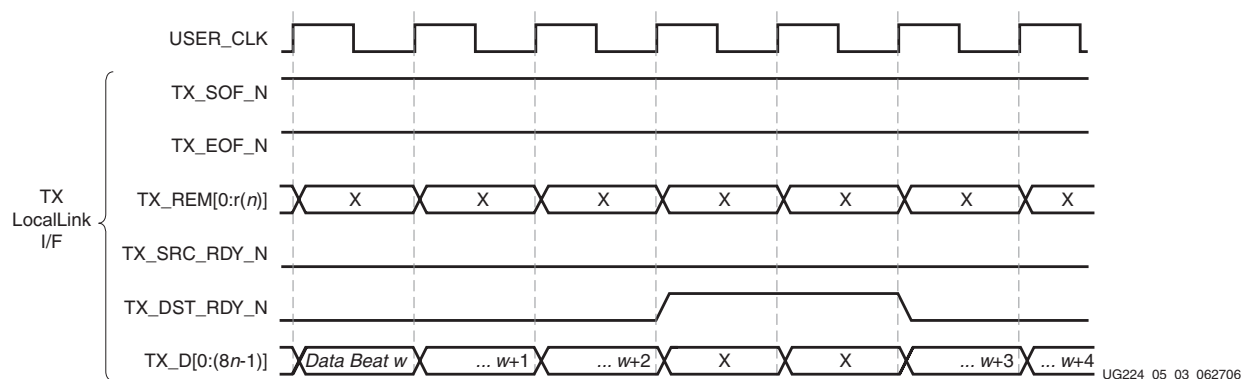


Figure 5-3: Transmitting a Message with NFC Idles Inserted

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

## User Flow Control

The Aurora 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 5-3](#) describes the ports for the UFC interface.

**Table 5-3: 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 <a href="#">Table 5-4, page 51</a> .
UFC_TX_ACK_N	Output	Asserted when an Aurora 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 5-4 shows the legal SIZE code values for UFC.

Table 5-4: 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 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 5-4 shows a useful circuit for switching TX\_D from sending regular data to UFC data.

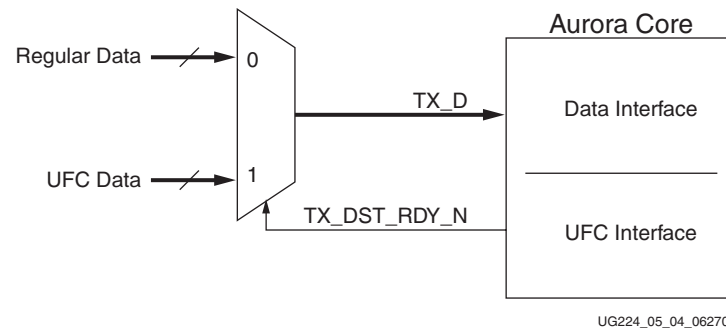


Figure 5-4: Data Switching Circuit

Table 5-5, page 52 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 5-5: 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				
8 Bytes	3		3		
10 Bytes	4		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				
12 Bytes	5		3		2
14 Bytes	6				
16 Bytes	7				
2 Bytes	0	8 Bytes	1	16 Bytes or more	1
4 Bytes	1		2		
6 Bytes	2				
8 Bytes	3				
10 Bytes	4				
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 5-5. In this case a 4-byte message is being sent on an 8-byte interface. Note that TX\_DST\_RDY\_N is deasserted for two cycles. Aurora cores use this gap in the data flow to transmit the UFC header and message data.

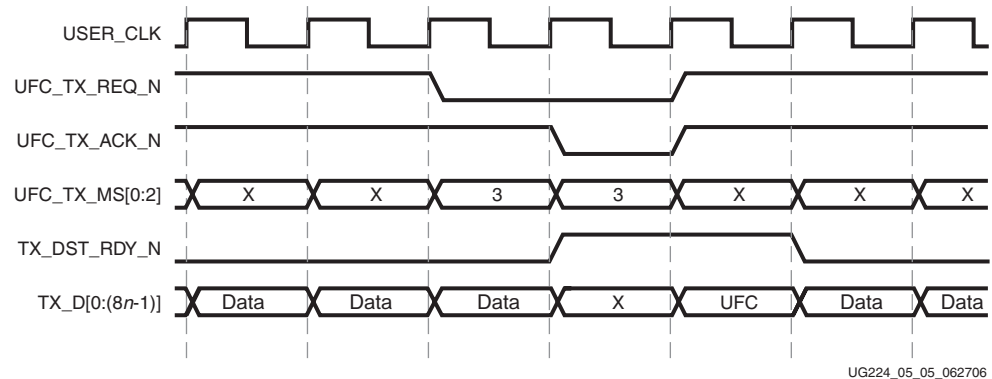


Figure 5-5: Transmitting a Single-Cycle UFC Message

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

The procedure for transmitting a two-cycle UFC message is shown in Figure 5-6. In this case the user application is sending a 16-byte message using an 8-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.

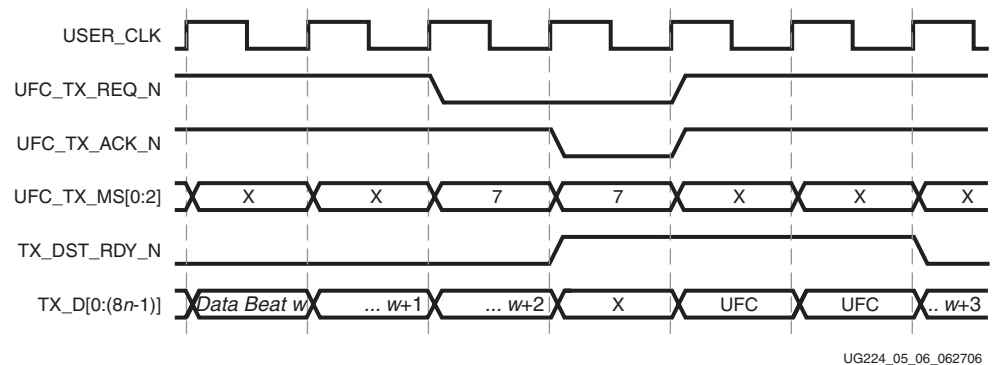


Figure 5-6: Transmitting a Multi-Cycle UFC Message

## Receiving User Flow Control Messages

When the Aurora 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 5-7 shows an Aurora core with an 8-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.

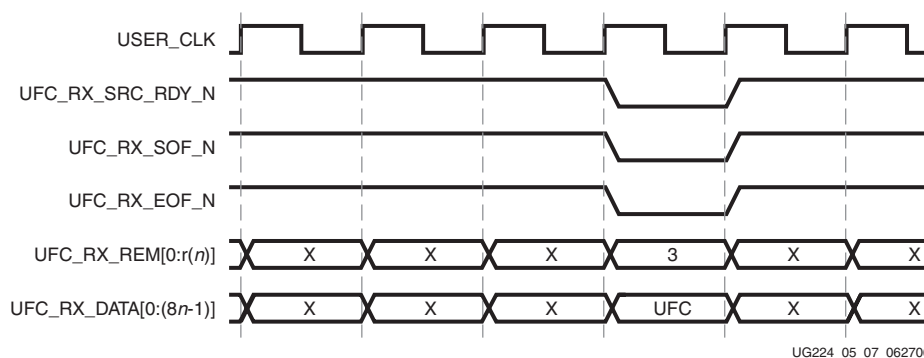


Figure 5-7: Receiving a Single-Cycle UFC Message

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

Figure 5-8 shows an Aurora core with an 8-byte interface receiving a 16-byte message. Note that the resulting frame is two cycles long, with UFC\_RX\_REM set to 7 on the second cycle indicating that all 8 bytes of the data are valid.

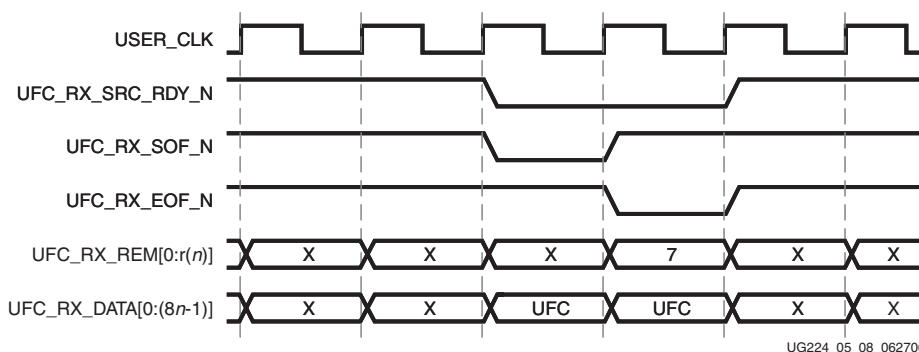


Figure 5-8: Receiving a Multi-Cycle UFC Message

## Status, Control, and the GTP Block Interface

### Introduction

The status and control ports of the Aurora core allow user applications to monitor the Aurora channel and use built-in features of the GTP transceivers.

Aurora 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 core's status and control interface, along with the GTP transceiver serial I/O interface and the sideband initialization ports that are used exclusively for simplex modules.

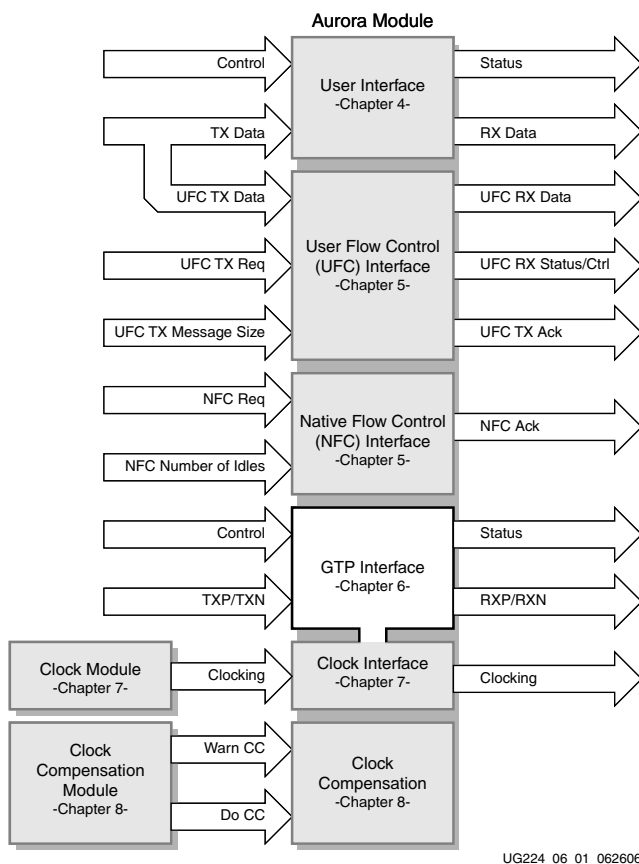


Figure 6-1: Top-Level GTP Block Interface

## Full-Duplex Cores

### Full-Duplex Status and Control Ports

Full-duplex cores provide a TX and an RX Aurora channel connection. Figure 6-2 shows the status and control interface for a full-duplex Aurora core. Table 6-1 describes the function of each of the ports in the interface.

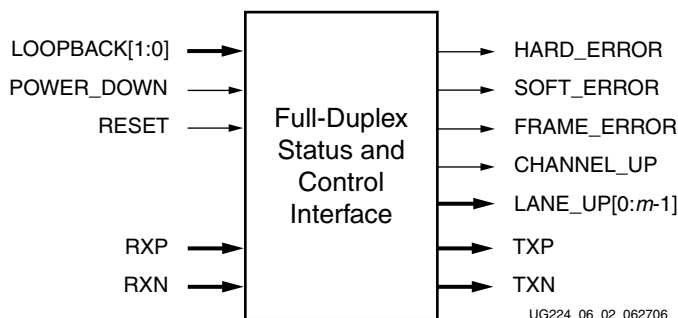


Figure 6-2: Status and Control Interface for Full-Duplex Cores

Table 6-1: Status and Control Ports for Full-Duplex Cores

Name	Direction	Description
CHANNEL_UP	Output	Asserted when Aurora channel initialization is complete and channel is ready to send data. The Aurora core can 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 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 core resets). See “Error Signals in Full-Duplex Cores,” page 57 for more details.
LOOPBACK[0] LOOPBACK[1]	Input	Refer to the <i>Virtex-5 RocketIO GTP Transceiver User Guide</i> for details about loopback. See “Related Xilinx Documents” in Chapter 1.
POWER_DOWN	Input	Drives the powerdown input of the GTP transceiver (active-High).
RESET	Input	Resets the Aurora 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 “Error Signals in Full-Duplex Cores,” page 57 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.



Table 6-1: Status and Control Ports for Full-Duplex Cores (Continued)

Name	Direction	Description
TXP[0:m-1]	Output	Positive differential serial data output pin.
TXN[0:m-1]	Output	Negative differential serial data output pin.

**Notes:**

1.  $n$  is the number of bytes in the interface; maximum value of  $n$  is 16 for the UFC interface
2. Data width is given by  $[0:(8n-1)]$
3. REM bus widths are given by  $[0:r(n)]$ , where  $r(n) = \text{ceiling} [\{\log_2(n)\}-1]$
4.  $m$  is the number of GTP transceivers
5. LANE\_UP width is given by  $[0:m-1]$

## Error Signals in Full-Duplex Cores

Equipment problems and channel noise can cause errors during Aurora channel operation. 8B/10B encoding allows the Aurora 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 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 burst of soft errors. The core uses the leaky bucket algorithm described in the *Aurora 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 core automatically resets itself and attempts to reinitialize. In most cases, this will allow the Aurora 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 leaky bucket algorithm.

Aurora cores with a LocalLink data interface can also detect errors in Aurora 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 6-2 summarizes the error conditions the Aurora core can detect and the error signals used to alert the user application.

Table 6-2: Error Signals in Full-Duplex Cores

Signal	Description
HARD_ERROR	<p><b>TX Overflow/Underflow:</b> 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><b>RX Overflow/Underflow:</b> 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><b>Bad Control Character:</b> The protocol engine attempts to send a bad control character. This is an indication of design corruption or catastrophic failure.</p> <p><b>Soft Errors:</b> There are too many soft errors within a short period of time. The Aurora 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><b>Invalid Code:</b> 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><b>Disparity Error:</b> 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> <p><b>No Data in Frame:</b> A channel frame is received with no data.</p>
FRAME_ERROR	<p><b>Truncated Frame:</b> A channel frame is started without ending the previous channel frame, or a channel frame is ended without being started.</p> <p><b>Invalid Control Character:</b> The protocol engine receives a control character that it does not recognize.</p> <p><b>Invalid UFC Message Length:</b> A UFC message is received with an invalid length.</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 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 cores can 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 6-3](#) shows the status and control interface for a simplex TX core. [Table 6-3](#) describes the function of each of the ports in the interface.

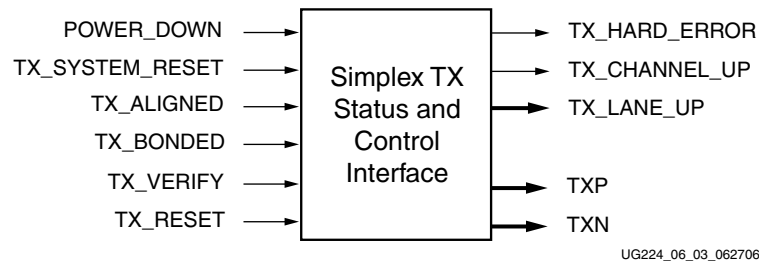


Figure 6-3: Status and Control Interface for Simplex TX Core

Table 6-3: 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.
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 channel initialization is complete and channel is ready to send data. The Aurora core can 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 core resets). See <a href="#">“Error Signals in Simplex Cores,” page 64</a> for more details.
POWER_DOWN	Input	Drives the powerdown input of the GTP transceiver (active-High).
TX_SYSTEM_RESET	Input	Resets the Aurora 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.  $n$  is the number of bytes in the interface; maximum value of  $n$  is 16 for the UFC interface
2. Data width is given by  $[0:(8n-1)]$
3. REM bus widths are given by  $[0:r(n)]$ , where  $r(n) = \text{ceiling}[\{\log_2(n)\}-1]$
4.  $m$  is the number of GTP transceivers
5. TX\_LANE\_UP width is given by  $[0:m-1]$

## Simplex RX Status and Control Ports

Simplex RX cores allow user applications to receive data from a simplex TX core. Figure 6-4 shows the status and control interface for a simplex RX core. Table 6-4, page 61 describes the function of each of the ports in the interface.

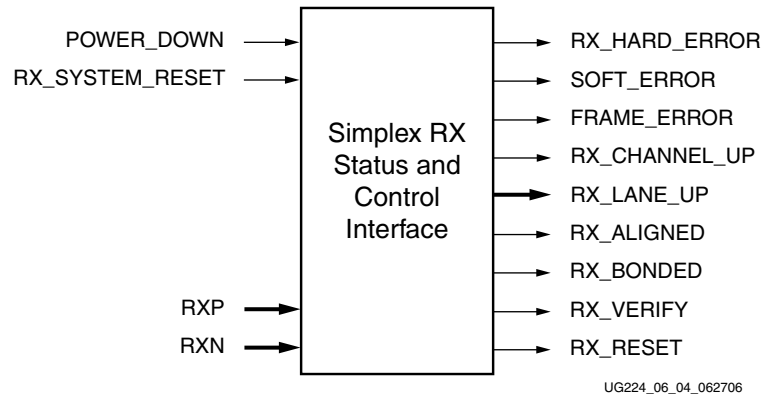


Figure 6-4: Status and Control Interface for Simplex RX Core

Table 6-4: 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 channel initialization is complete and channel is ready to send data. The Aurora core can 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 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 core resets). See “Error Signals in Simplex Cores,” page 64 for more details.
POWER_DOWN	Input	Drives the powerdown input of the GTP transceiver (active-High).

Table 6-4: Status and Control Ports for Simplex RX Cores (Continued)

Name	Direction	Description
RX_SYSTEM_RESET	Input	Resets the Aurora core (active-High).
SOFT_ERROR	Output	Soft error detected in the incoming serial stream. See “Error Signals in Simplex Cores,” page 64 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:**

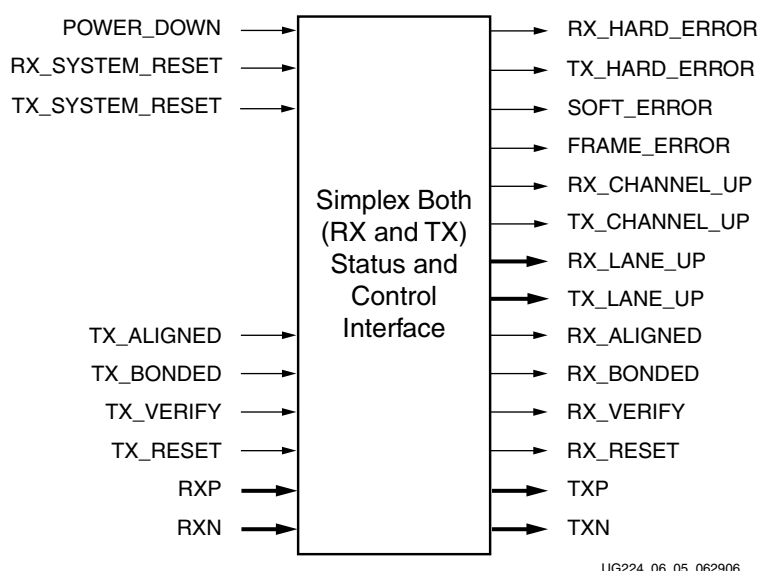
1.  $n$  is the number of bytes in the interface; maximum value of  $n$  is 16 for the UFC interface
2. Data width is given by  $[0:(8n-1)]$
3. REM bus widths are given by  $[0:r(n)]$ , where  $r(n) = \text{ceiling}[\{\log_2(n)\}-1]$
4.  $m$  is the number of GTP transceivers
5. RX\_LANE\_UP width is given by  $[0:m-1]$

## 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 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 6-5 shows the status and control interface for a simplex Both core. Table 6-5 describes the function of each of the ports in the interface.



UG224\_06\_05\_062906

Figure 6-5: Status and Control Interface for Simplex Both Cores

Table 6-5: 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 channel initialization is complete and channel is ready to send data. The Aurora core can receive data before CHANNEL_UP.
TX_CHANNEL_UP	Input	Asserted when Aurora channel initialization is complete and channel is ready to send data. The Aurora core can 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 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 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.
RX_HARD_ERROR TX_HARD_ERROR	Output	Hard error detected. (Active-High, asserted until Aurora core resets). See “Error Signals in Simplex Cores,” page 64 for more details.
POWER_DOWN	Input	Drives the powerdown input of the GTP transceiver (active-High).

Table 6-5: Status and Control Ports for Simplex “Both” Cores (Continued)

Name	Direction	Description
RX_SYSTEM_RESET TX_SYSTEM_RESET	Input	Resets the Aurora core (active-High).
SOFT_ERROR	Output	Soft error detected in the incoming serial stream. See “Error Signals in Simplex Cores,” page 64 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.  $n$  is the number of bytes in the interface; maximum value of  $n$  is 16 for the UFC interface
2. Data width is given by  $[0:(8n-1)]$
3. REM bus widths are given by  $[0:r(n)]$ , where  $r(n) = \text{ceiling}[\{\log_2(n)\}-1]$
4.  $m$  is the number of GTP transceivers
5. LANE\_UP width is given by  $[0:m-1]$

## Error Signals in Simplex Cores

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 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 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 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 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 leaky bucket algorithm.

Simplex RX and simplex Both cores with a LocalLink data interface can also detect errors in Aurora 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 6-6 summarizes the error conditions simplex Aurora cores can detect and the error signals uses to alert the user application.

Table 6-6: Error Signals in Simplex Cores

Signal	Description	TX	RX	Both
HARD_ERROR	<b>TX Overflow/Underflow:</b> 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
	<b>RX Overflow/Underflow:</b> 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
	<b>Bad Control Character:</b> The protocol engine attempts to send a bad control character. This is an indication of design corruption or catastrophic failure.	x		x
	<b>Soft Errors:</b> There are too many soft errors within a short period of time. The Aurora 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
SOFT_ERROR	<b>Invalid Code:</b> 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
	<b>Disparity Error:</b> 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
	<b>No Data in Frame:</b> A channel frame is received with no data.		x	x
FRAME_ERROR	<b>Truncated Frame:</b> A channel frame is started without ending the previous channel frame, or a channel frame is ended without being started.	x		x
	<b>Invalid Control Character:</b> The protocol engine receives a control character that it does not recognize.		x	x
	<b>Invalid UFC Message Length:</b> A UFC message is received with an invalid length.		x	x

## Simplex Initialization

Simplex cores do not depend on signals from an Aurora 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 a 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. Examples of suitable back channels include:

The `aurora_example` design included in the examples directory with simplex Aurora cores shows a simple side channel that uses 3-4 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 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 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 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-Low signal. When POWER\_DOWN is asserted, the GTP transceivers in the Aurora 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 7, “Clock Interface and Clocking”](#)). TX\_OUT\_CLK will stop when the GTP transceivers are powered down. See the *Virtex-5 RocketIO GTP Transceiver User Guide* for the device you are using for details about powering down GTP transceivers.

## Timing

[Figure 6-6](#) 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.

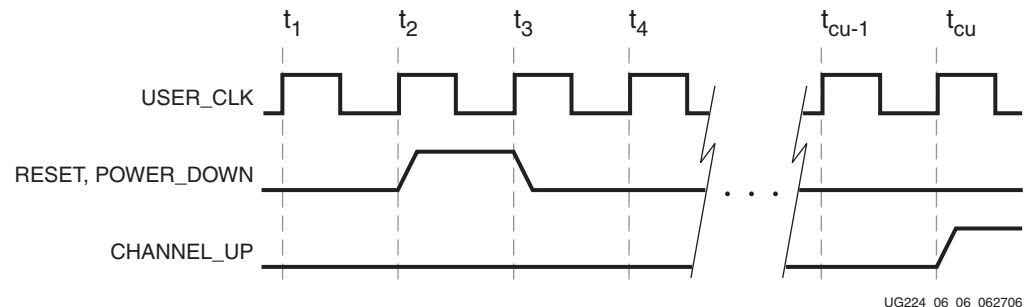


Figure 6-6: Reset and Power Down Timing



## Clock Interface and Clocking

### Introduction

Good clocking is critical for the correct operation of the Aurora 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 transceiver. It also requires at least one frequency locked parallel clock for synchronous operation with the user application.

The Virtex -5 GTP architecture has a pair of transceivers in each GTP\_DUAL tile. Because of the shared PMA PLL architecture inside the GTP\_DUAL tile, each reference clock sources both channels. 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 transceiver.

Each Aurora core is generated with an examples directory that includes a design called `aurora_example`. This design by instantiating the generated Aurora 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.

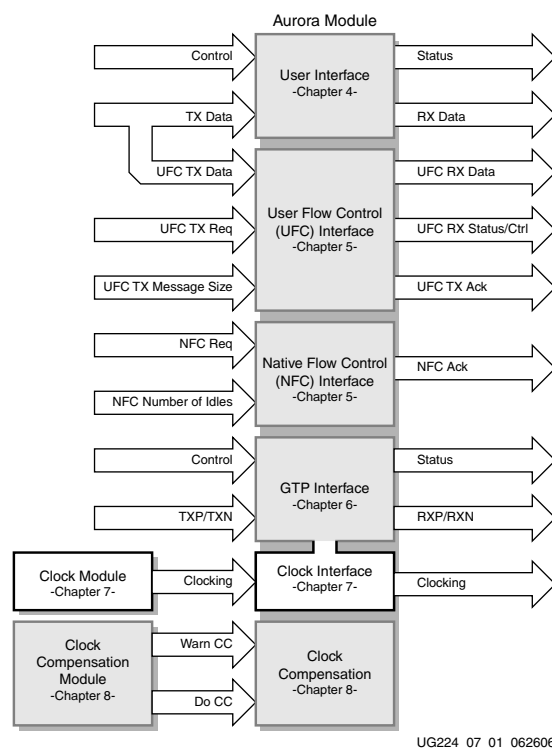


Figure 7-1: Top-Level Clocking

## Clock Interface Ports for GTP Transceiver Cores

Table 7-1 describes the GTP Aurora core clock ports.

Table 7-1: Clock Ports for a GTP Aurora Core

Clock Ports	Direction	Description
DCM_NOT_LOCKED	Input	If a DCM is used to generate clocks for the Aurora core, the DCM_NOT_LOCKED signal should be connected to the inverse of the DCM's LOCKED signal. The clock module provided with the Aurora core uses the DCM for clock division. The DCM_NOT_LOCKED signal from the clock module should be connected to the DCM_NOT_LOCKED signal on the Aurora core. If DCM is not used to generate clock signals for the Aurora core, tie DCM_NOT_LOCKED to ground.
USER_CLK	Input	Parallel clock shared by the Aurora core and the user application. On GTP Aurora cores the USER_CLK is the output of a DCM whose input is derived from TX_OUT_CLK. The rate is the half the rate of the TX_OUT_CLK, which is determined by the clock rate settings of the GTP transceivers in the core.
SYNC_CLK	Input	Parallel clock used by the internal synchronization logic of the GTP transceivers in the Aurora core. SYNC_CLK is twice the rate of USER_CLK and it is phase aligned to the USER_CLK.
REF_CLK	Input	REF_CLK (CLKP/CLKN) is a dedicated external clock generated from an oscillator. This clock is fed through BUFDS. To minimize the number of oscillators, the GTP transceiver architecture has a NORTH/SOUTH clock routing matrix using CLKP/CLKN. A GTP_DUAL tile shares its clock with its neighbors using dedicated clock routing resources.
GREF_CLK	Input	The internal clock nets of the FPGA can provide the reference clock for the GTP_DUAL by connecting the output of a global clock buffer (BUFG) or a regional clock buffer (BUFR) to the CLKIN port. This hidden input port is an alternative reference clock source from the global clock tree.

## Reference Clocks for GTP Transceiver Designs

In GTP transceiver designs, the reference clock can be either GREFCLK or REFCLK, which is a differential input clock for each GTP\_DUAL. The reference clock for GTP\_DUAL is provided through the CLKIN port.

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

- “Clocking from Inside the FPGA: GREFCLK”
- “Clocking from an External Source”
- “Clocking from a Neighboring GTP\_DUAL Tile”

### Clocking from Inside the FPGA: GREFCLK

GTP transceivers are arranged in pairs called GTP\_DUAL tiles. The internal clock nets of the FPGA can provide the reference clock for the GTP\_DUAL by connecting the output of a global clock buffer (BUFG) or a regional clock buffer (BUFR) to the CLKIN port. This type of clock is called GREFCLK and should be avoided, if possible. FPGA clocking resources introduce a large amount of jitter for operations at high speeds. Thus GREF has the lowest performance among the clocking options. See *Virtex-5 Family Overview* for the jitter margins at different speeds.

Figure 7-2 shows how a GTP\_DUAL tile connects to a BUFR or a BUFG. If a BUFR is used, it must be located in the same region as the GTP\_DUAL tile.

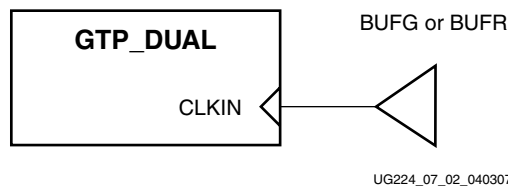


Figure 7-2: Single GTP\_DUAL Tile Clocked from the FPGA

### Clocking from an External Source

Each GTP\_DUAL tile has a pair of dedicated pins that can be connected to an external clock source. To use these pins, a BUFDS primitive is instantiated. In the User Constraints file, the BUFDS input pins are set to the dedicated clock pins for the tile. In the design, the output of the BUFDS is connected to the CLKIN port.

Each GTP\_DUAL takes differential REFCLK inputs, which are directly bonded to the FPGA pins. REFCLK should be used for high data rate applications (1 Gb/s or higher). For multilane Aurora designs, REFCLK of any GTP\_DUAL can be used as the reference clock for the Aurora design. Using a low-jitter oscillator delivers a high-quality clock suitable for top-speed operation. Figure 7-3 shows a differential GTP\_DUAL clock pin pair sourced by an external oscillator on the board.

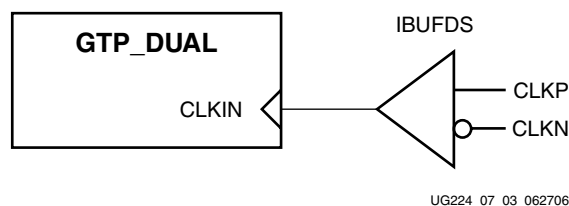


Figure 7-3: Single GTP\_DUAL Tile Clocked Externally

## Clocking from a Neighboring GTP\_DUAL Tile

The external clock from one tile can be used to drive the CLKIN ports of neighboring tiles. The example in Figure 7-4 uses the clock from one GTP\_DUAL tile to clock six neighboring tiles. A GTP\_DUAL tile shares its clock with its neighbors using dedicated clock routing resources. GREFCLK cannot be shared using these resources.

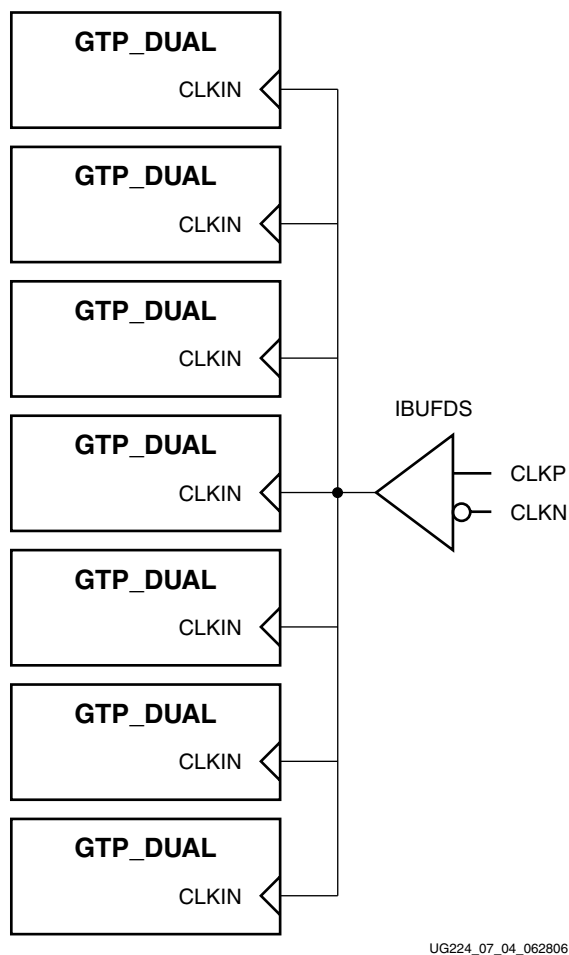


Figure 7-4: Multiple Tiles with a Shared Reference Clock

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

The maximum number of GTP transceivers that can be sourced by a single clock pin pair translates to a maximum of 14 pairs. Designs with more than 14 GTP 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 transceivers are combined to form a single channel.

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

## Clock Rates for GTP Transceiver Designs

GTP transceivers support a wide range of serial rates. The attributes used to configure the GTP transceivers in the Aurora core for a specific line rate are kept in the MGT\_WRAPPER module for simulation. These attributes are set automatically by the CORE Generator tool 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 RocketIO GTP Transceiver 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 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 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 core to provide Aurora-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.

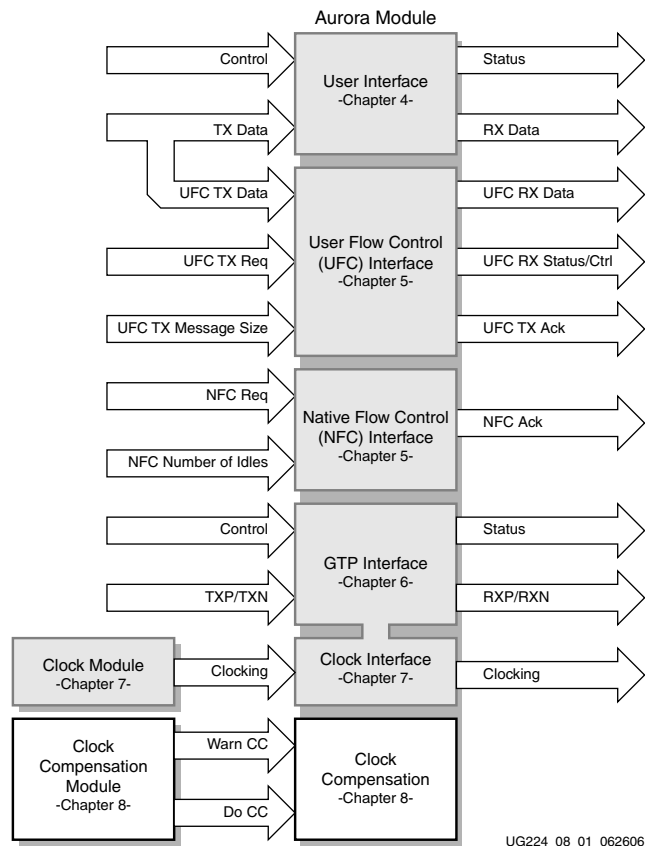


Figure 8-1: Top-Level Clock Compensation

## Clock Compensation Interface

All Aurora cores include a clock compensation interface for controlling the transmission of clock compensation sequences. Table 8-1 describes the function of the clock compensation interface ports.

Table 8-1: Clock Compensation I/O Ports

Name	Direction	Description
DO_CC	Input	The Aurora 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 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 8-2 and Figure 8-3 are waveform diagrams showing how the DO\_CC signal works.

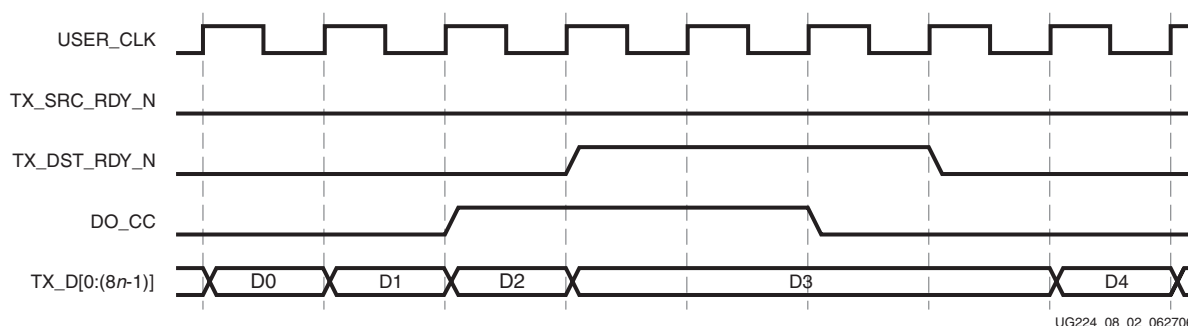


Figure 8-2: Streaming Data with Clock Compensation Inserted

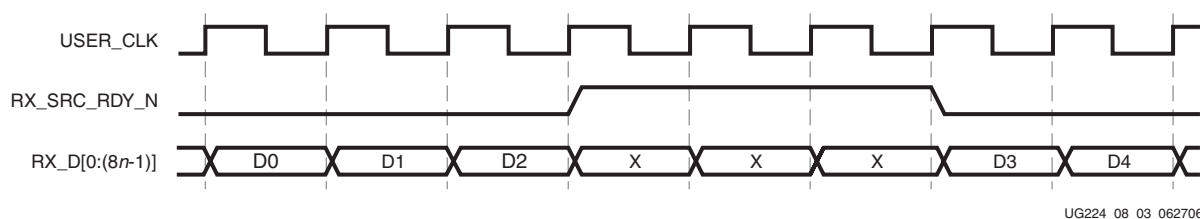


Figure 8-3: Data Reception Interrupted by Clock Compensation

The Aurora protocol specifies a clock compensation mechanism that allows up to  $\pm 100$  ppm difference between reference clocks on each side of an Aurora channel. To perform Aurora-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 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 8-2 shows the required durations and periods for 2-byte and 4-byte wide lanes.

Table 8-2: 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). 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 8-3 shows the number of lookahead cycles required for each combination of lane width, channel width, and maximum UFC message size.

Table 8-3: 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

Table 8-3: Lookahead Cycles (Continued)

Data Interface Width	Max UFC size	WARN_CC Lookahead
14	2-14	3
14	16	4
≥16	2-16	3

To make Aurora compliance easy, a standard clock compensation module is generated along with each Aurora core from the CORE Generator tool, in the `cc_manager` subdirectory. It automatically generates pulses to create Aurora 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 module, except in special cases. Table 8-4 shows the port description for the Standard CC module.

Table 8-4: Standard CC I/O Port

Name	Direction	Description
WARN_CC	Output	Connect this port to the WARN_CC input of the Aurora core when using UFC.
DO_CC	Output	Connect this port to the DO_CC input of the Aurora 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 channel are being driven by the same clock (see Figure 8-3, page 76) 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 user constraints file 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 2 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 8 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