

LogiCORE™ IP Aurora 64B/66B v4.1

User Guide

UG237 (v4.1) April 19, 2010



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

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

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

© 2008-2010 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/24/08	1.1	Virtex-5 FPGA Aurora 64B/66B v1.1 release.
03/24/08	1.1.1	Miscellaneous typographical edits.
06/27/08	1.2	<p>Virtex-5 FPGA Aurora 64B/66B v1.2 release.</p> <ul style="list-style-type: none">• Updated the tree content in “Aurora 64B/66B Project Directory Structure,” page 25.• Updated the chapter titles and their respective module names in each chapter’s top-level diagram. Table 2-1, page 26, deleted -example.• Table 2-1, page 26 changed ERROR_COUNT[7:0] to DATA_ERROR_COUNT[7:0] and added UFC_ERROR and USER_K_ERROR.• Added “FRAME_GEN,” page 27 and “FRAME_CHECK,” page 35.• Replaced TX_LOCK with PLL_LOCK, MGT_CLK_LOCKED with V5_CLK_LOCKED, and “Both” simplex with RX/TX simplex.• Updated bus ranges in Figure 5-2, page 53, Figure 5-3, page 53, Table 5-2, page 52, Figure 6-2, page 56, and Table 6-1, page 56.• Miscellaneous typographical edits and edits for clarity.

Date	Version	Revision
09/19/08	1.3	<p>LogiCORE IP Aurora 64B/66B v1.3 release.</p> <p>Added VHDL design support in “Introduction,” page 17.</p> <p>Added “System Requirements,” page 17, “Before You Begin,” page 18, and “Installing the Core,” page 18.</p> <p>Expanded and reorganized “Aurora 64B/66B Project Directory Structure,” page 25 for clarity.</p> <p>Added the following figures and tables:</p> <ul style="list-style-type: none"> • In “Framing TX Data Interface” added Figure 2-5, page 28 and Table 2-2, page 29 • In “Streaming TX Data Interface” added Figure 2-7, page 30 and Table 2-3, page 30 • In “UFC TX Interface” added Figure 2-9, page 31 and Table 2-4, page 31 • In “NFC TX Interface” added Figure 2-11, page 33 and Table 2-5, page 33 • In “User K TX Interface” added Figure 2-13, page 34 and Table 2-6, page 34 • In “Framing RX Data Interface” added Figure 2-15, page 35 and Table 2-7, page 36 • In “Streaming RX Data Interface” added Figure 2-17, page 37 and Table 2-8, page 37 • In “UFC RX Interface” added Figure 2-19, page 38 and Table 2-9, page 38 • In “User K RX Interface” added Figure 2-21, page 39 and Table 2-10, page 40 <p>Reorganized the TX- and UFC-type signals by group in Figure 4-4, page 48 and Figure 4-5, page 49.</p>
04/24/09	2.1	<p>Updated tools and core. Changed DCM to PLL. Added SHIM_CLK. In Chapter 2, “Using the IP Customizer” added “Column Used” and “Use ChipScope Pro Analyzer.” Removed Lane Width. Renamed Clock Source to “Clock Source 1 and Clock Source 2” and Finish to “Generate.” Renamed and added directories in “Aurora 64B/66B Project Directory Structure.”</p> <p>Added the following:</p> <ul style="list-style-type: none"> • “Designing with the Core” section • “Top Level Architecture” section • Appendix A, “Two Aurora 64B/66B Cores in Virtex-5 Family Sharing a RocketIO Transceiver Tile” • Appendix B, “Generating a GTX Wrapper File from the GTX Transceiver Wizard”
09/16/09	3.1	<p>LogiCore IP Aurora 64B/66B v3.1 release. Updated core and tools versions. Added support for Virtex-6 FPGA GTX transceivers. Moved the following sections to the new LogiCORE IP Aurora 64B/66B Getting Started Guide (ug238): Aurora 64B/66B Project Directory Structure, Example Design Overview, FRAME_GEN, and FRAME_CHECK. Deleted Using the Build Script section.</p>
04/19/10	4.1	<p>LogiCore IP Aurora 64B/66B v4.1 release. Updated core and tools versions. Added support for Virtex-6 HXT devices and for Virtex-6 FPGA GTH transceivers. Added status and control ports for simplex-TX cores Table 6-2 and simplex-RX cores (Table 6-3). Restructured document.</p>

Table of Contents

Revision History	2
Schedule of Figures	9
Schedule of Tables	11
Preface: About This Guide	
Contents	13
Additional Resources	13
Conventions	14
Typographical	14
Online Document	15
Chapter 1: Introduction	
About the Core	17
System Requirements	17
Before You Begin	18
Installing the Core	18
Recommended Design Experience	18
Related Documents	18
Additional Core Resources	19
Technical Support	19
Feedback	19
Aurora 64B/66B Core	19
Document	19
Chapter 2: Customizing the Aurora 64B/66B Core	
Introduction	21
Using the IP Customizer	21
IP Customizer	21
Using the Build Script	26
Designing with the Core	26
General Design Guidelines	26
Keep It Registered	26
Recognize Timing Critical Signals	26
Use Supported Design Flows	26
Make Only Allowed Modifications	27
Chapter 3: User Interface	
Introduction	29
Top Level Architecture	29

Framing Interface	32
LocalLink TX Ports	32
LocalLink RX Ports	33
LocalLink Bit Ordering	33
Transmitting Data	34
Receiving Data	38
Framing Efficiency	39
Streaming Interface	40
Streaming TX Ports	40
Streaming RX Ports	41
Transmitting and Receiving Data	41

Chapter 4: Flow Control

Introduction	43
Native Flow Control	44
Example A: Transmitting an NFC Message	45
Example B: Receiving a Message with NFC Idles Inserted	45
User Flow Control	47
Transmitting UFC Messages	48
Receiving User Flow Control Messages	49

Chapter 5: User K-Block Interface

Introduction	51
Transmitting User K-Block	53
Receiving User K-Block	53

Chapter 6: Status, Control, and the GT Interface

Introduction	55
Status and Control Ports	56
Error Signals in Aurora 64B/66B Cores	58
Initialization	59
Reset and Power Down	60
Reset	60
Power Down	60
Timing	60

Chapter 7: Clock Interface and Clocking

Introduction	61
Clock Interface Ports for Virtex-5 and Virtex-6 FPGA Cores	62
Parallel Clocks for Virtex-5/Virtex-6 FPGA Designs	62
Reference Clocks for Virtex-5/Virtex-6 FPGA Designs	63

Chapter 8: Clock Compensation Interface

Introduction	65
Clock Compensation Interface	66

Appendix A: Two Aurora 64B/66B Cores in Virtex-5 Family Sharing a RocketIO Transceiver Tile

Shared Ports and Attributes	69
Steps to Modify Transceiver Specific Ports and Attributes	70
Steps to Bring Out Unused Transceiver Ports to Aurora 64B/66B GTX Wrapper File	70
Steps to Bring Out Aurora 64B/66B Core Specific Logic to GTX Wrapper File.	70
Steps to Instantiate Two Aurora 64B/66B Cores in the Top Level File	71

Appendix B: Generating a GTX Wrapper File from the GTX Transceiver Wizard

Schedule of Figures

Chapter 1: Introduction

Chapter 2: Customizing the Aurora 64B/66B Core

<i>Figure 2-1: Aurora 64B/66B IP Customizer Page 1</i>	22
<i>Figure 2-2: Aurora 64B/66B IP Customizer Page 2</i>	22
<i>Figure 2-3: Aurora 64B/66B IP Customizer Page 2 (Virtex-6 HXT Devices for GTX Transceivers Only)</i>	23
<i>Figure 2-4: Aurora 64B/66B IP Customizer Page 2 (Virtex-6 HXT Devices for GTH Transceivers Only)</i>	23

Chapter 3: User Interface

<i>Figure 3-1: Top-Level Architecture</i>	30
<i>Figure 3-2: Top-Level User Interface</i>	31
<i>Figure 3-3: Aurora 64B/66B Core Framing Interface (LocalLink)</i>	32
<i>Figure 3-4: LocalLink Interface Bit Ordering</i>	33
<i>Figure 3-5: Simple Data Transfer</i>	35
<i>Figure 3-6: Data Transfer with Pause</i>	36
<i>Figure 3-7: Data Transfer Paused by Clock Compensation</i>	36
<i>Figure 3-8: Transmitting Data</i>	37
<i>Figure 3-9: Data Reception with Pause</i>	38
<i>Figure 3-10: Aurora 64B/66B Core Streaming User Interface</i>	40
<i>Figure 3-11: Typical Streaming Data Transfer</i>	42
<i>Figure 3-12: Typical Streaming Data Reception</i>	42

Chapter 4: Flow Control

<i>Figure 4-1: Top-Level Flow Control</i>	43
<i>Figure 4-2: Transmitting an NFC Message</i>	45
<i>Figure 4-3: Transmitting a Message with NFC Idles Inserted</i>	46
<i>Figure 4-4: Transmitting a Single-Cycle UFC Message</i>	48
<i>Figure 4-5: Transmitting a Multi-Cycle UFC Message</i>	49
<i>Figure 4-6: Receiving a Single-Cycle UFC Message</i>	49
<i>Figure 4-7: Receiving a Multi-Cycle UFC Message</i>	50

Chapter 5: User K-Block Interface

<i>Figure 5-1: Top-Level User K-Block Interface</i>	51
<i>Figure 5-2: Transmitting User K Data and User K-Block Number</i>	53
<i>Figure 5-3: Receiving User K Data and User K-Block Number</i>	53

Chapter 6: Status, Control, and the GT Interface

<i>Figure 6-1: Top-Level GTX Interface</i>	55
<i>Figure 6-2: Status and Control Interface for the Aurora 64B/66B Core</i>	56
<i>Figure 6-3: Initialization Overview</i>	59
<i>Figure 6-4: Reset and Power Down Timing</i>	60

Chapter 7: Clock Interface and Clocking

<i>Figure 7-1: Top-Level Clocking</i>	61
---	----

Chapter 8: Clock Compensation Interface

<i>Figure 8-1: Top-Level Clock Compensation Interface</i>	65
<i>Figure 8-2: Streaming Data with Clock Compensation Inserted</i>	66
<i>Figure 8-3: Data Reception Interrupted by Clock Compensation</i>	66

Appendix A: Two Aurora 64B/66B Cores in Virtex-5 Family Sharing a RocketIO Transceiver Tile

<i>Figure A-1: Example Showing Two Single Lane Aurora 64B/66B Cores Sharing a Transceiver Tile in Virtex-5 Family</i>	71
---	----

Appendix B: Generating a GTX Wrapper File from the GTX Transceiver Wizard

Schedule of Tables

Chapter 1: Introduction

Chapter 2: Customizing the Aurora 64B/66B Core

Chapter 3: User Interface

<i>Table 3-1: LocalLink User I/O Ports (TX)</i>	32
<i>Table 3-2: LocalLink User I/O Ports (RX)</i>	33
<i>Table 3-3: TX Data Remainder Values</i>	34
<i>Table 3-4: Typical Channel Frame</i>	35
<i>Table 3-5: Efficiency Example</i>	39
<i>Table 3-6: Typical Overhead for Transmitting 256 Data Bytes</i>	39
<i>Table 3-7: Streaming User I/O Ports (TX)</i>	40
<i>Table 3-8: Streaming User I/O Ports (RX)</i>	41

Chapter 4: Flow Control

<i>Table 4-1: NFC I/O Ports</i>	44
<i>Table 4-2: UFC I/O Ports</i>	47

Chapter 5: User K-Block Interface

<i>Table 5-1: Valid Block Type Field Values for User K-Block</i>	52
<i>Table 5-2: User K-Block I/O Ports</i>	52

Chapter 6: Status, Control, and the GT Interface

<i>Table 6-1: Status and Control Ports for Full-Duplex Cores</i>	56
<i>Table 6-2: Status and Control Ports for Simplex-TX Cores</i>	57
<i>Table 6-3: Status and Control Ports for Simplex-RX Cores</i>	57
<i>Table 6-4: Error Signals in Full-Duplex Cores</i>	58

Chapter 7: Clock Interface and Clocking

<i>Table 7-1: Clock Ports for a Virtex-5 and Virtex-6 FPGA Aurora 64B/66B Core</i>	62
--	----

Chapter 8: Clock Compensation Interface

<i>Table 8-1: Clock Compensation I/O Ports</i>	66
<i>Table 8-2: Standard CC I/O Port</i>	67

Appendix A: Two Aurora 64B/66B Cores in Virtex-5 Family Sharing a RocketIO Transceiver Tile

Appendix B: Generating a GTX Wrapper File from the GTX Transceiver Wizard

About This Guide

This user guide describes the function and operation of the LogiCORE™ IP Aurora 64B/66B v4.1 core and provides information about designing, customizing, and implementing the core.

Contents

This guide contains the following chapters:

- [Preface, “About this Guide”](#) introduces the organization and purpose of the design guide, a list of additional resources, and the conventions used in this document.
- [Chapter 1, “Introduction”](#) describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- [Chapter 2, “Customizing the Aurora 64B/66B Core”](#) describes how to customize an Aurora 64B/66B core with the available parameters.
- [Chapter 3, “User Interface”](#) provides port descriptions for the user interface.
- [Chapter 4, “Flow Control”](#) describes the user flow control and native flow control options for sending and receiving data.
- [Chapter 5, “User K-Block Interface”](#) describes short single block data transmission and reception.
- [Chapter 6, “Status, Control, and the GT Interface”](#) provides diagrams and port descriptions for the Aurora 64B/66B core’s status and control interface, along with the GTX serial I/O interface.
- [Chapter 7, “Clock Interface and Clocking”](#) describes how to connect FPGA clocking resources.
- [Chapter 8, “Clock Compensation Interface”](#) covers Aurora 64B/66B clock compensation, and explains how to customize it for a given system.

Additional Resources

To find additional documentation, see the Xilinx website at:

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

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

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

Conventions

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

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
Courier font	Messages, prompts, and program files that the system displays	speed grade: - 100
Courier bold	Literal commands you enter in a syntactical statement	ngdbuild <i>design_name</i>
Italic font	Variables in a syntax statement for which you must supply values	ngdbuild <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
Square brackets []	An optional entry or parameter. However, in bus specifications, such as bus [7:0] , they are required.	ngdbuild [<i>option_name</i>] <i>design_name</i>
Braces { }	A list of items from which you must choose one or more	lowpwr = { on off }
Vertical bar	Separates items in a list of choices	lowpwr = { on off }
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Omitted repetitive material	allow block <i>block_name loc1 loc2 ... locn</i> ;
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 “ Additional Resources ” for details. Refer to “ Title Formats ” in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the <i>Virtex-5 FPGA User Guide</i> .
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

Introduction

This chapter introduces the LogiCORE™ IP Aurora 64B/66B core and provides related information, including recommended design experience, additional resources, technical support, and how to submit feedback to Xilinx. The Aurora 64B/66B core is based on the *Aurora 64B/66B Protocol Specification* and uses the high-speed serial GTX or GTH transceivers in applicable Virtex®-5 and Virtex-6 FPGAs. The core is delivered as open-source code and supports Verilog and VHDL design environments. Each core comes with an example design and supporting modules.

About the Core

The Aurora 64B/66B core is a Xilinx CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see www.xilinx.com/aurora.

System Requirements

Windows

- Windows XP Professional 32-bit/64-bit
- Windows Vista Business 32-bit/64-bit

Linux

- Red Hat Enterprise Linux WS v4.0 32-bit/64-bit
- Red Hat Enterprise Desktop v5.0 32-bit/64-bit (with Workstation Option)
- SUSE Linux Enterprise (SLE) desktop and server v10.1 32-bit/64-bit

Software

- ISE® v12.1 software
- Mentor Graphics ModelSim v6.5c and above
- Cadence Incisive Enterprise Simulator (IES) v9.2 and above
- Synopsys Synplify Pro D-2009.12
- Synopsys VCS and VCS MX 2009.12 and above

Before You Begin

Before installing the core, you must have a MySupport account and the v12.1 software installed on your system. If you already have an account and have the software installed, go to “[Installing the Core](#),” 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 v12.1 software. For the software installation instructions, see the ISE Design Suite Release Notes and Installation Guide available in the ISE software documentation located on the [ISE Design Suite product page](#).

Installing the Core

The Aurora 64B/66B core is included with the v12.1 software.

See [ISE CORE Generator IP Updates - Installation Instructions](#) for details about installing ISE 12.1.

Recommended Design Experience

Although the Aurora 64B/66B 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.

See [Chapter 6, “Status, Control, and the GT Interface”](#) carefully, and consult the PCB design requirements information in the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*, *Virtex-6 FPGA GTX Transceivers User Guide*, and *Virtex-6 FPGA GTH Transceivers User Guide*. Contact your local Xilinx representative for a closer review and estimation for your specific requirements.

Related Documents

Before generating an Aurora 64B/66B core, users should be familiar with the following:

- SP011 *Aurora 64B/66B Protocol Specification* is located on the [Aurora product page](#)
- SP006 *LocalLink Interface Specification* is located from the [LocalLink product page](#) [Click Download to register and access this document]
- [UG198](#), *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*
- [UG366](#), *Virtex-6 FPGA GTX Transceivers User Guide*
- [UG371](#), *Virtex-6 FPGA GTH Transceivers User Guide*
- ISE software documentation on the [ISE Design Suite product page](#)

Additional Core Resources

For detailed information and updates about the Aurora 64B/66B core, see the following documents, located on the [Aurora product page](#).

- [DS528](#), *Aurora 64B/66B v4.1 Data Sheet*
- [UG238](#), *LogiCORE IP Aurora 64B/66B Getting Started Guide*
- Virtex-5/Virtex-6 FPGA Aurora 64B/66B Release Notes

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the Aurora 64B/66B core.

Xilinx will provide technical support for use of this product as described in this guide. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines, or for modifications to the source code.

Feedback

Xilinx welcomes comments and suggestions about the Aurora 64B/66B core and the accompanying documentation.

Aurora 64B/66B Core

For comments or suggestions about the Aurora 64B/66B core, please submit a WebCase from 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. Be sure to include the following information:

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

Customizing the Aurora 64B/66B Core

Introduction

The Aurora 64B/66B core can be customized to suit a wide variety of requirements using the CORE Generator™ software. This chapter details the customization parameters available to the user and how these parameters are specified within the IP Customizer interface.

Using the IP Customizer

The Aurora 64B/66B IP customizer is presented when the user selects the Aurora 64B/66B core in the CORE Generator software. For help starting and using the CORE Generator software, see the *CORE Generator Guide* in the ISE™ software documentation. [Figure 2-1](#), [Figure 2-2](#), and [Figure 2-3](#) show features that are described in corresponding sections.

Note: The options shown in [Figure 2-3](#) are only available for the Virtex®-6 HXT devices.

IP Customizer

[Figure 2-1](#), [page 22](#) shows the customizer. The left side displays a representative block diagram of the Aurora 64B/66B core as currently configured. The right side consists of user-configurable parameters. For details on the customizing options, see the *LogiCORE IP Aurora 64B/66B Getting Started Guide*.

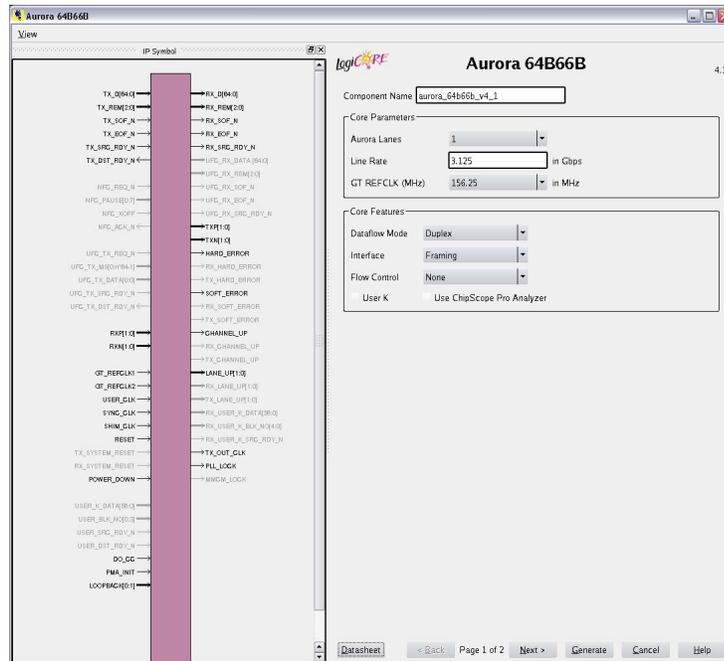


Figure 2-1: Aurora 64B/66B IP Customizer Page 1

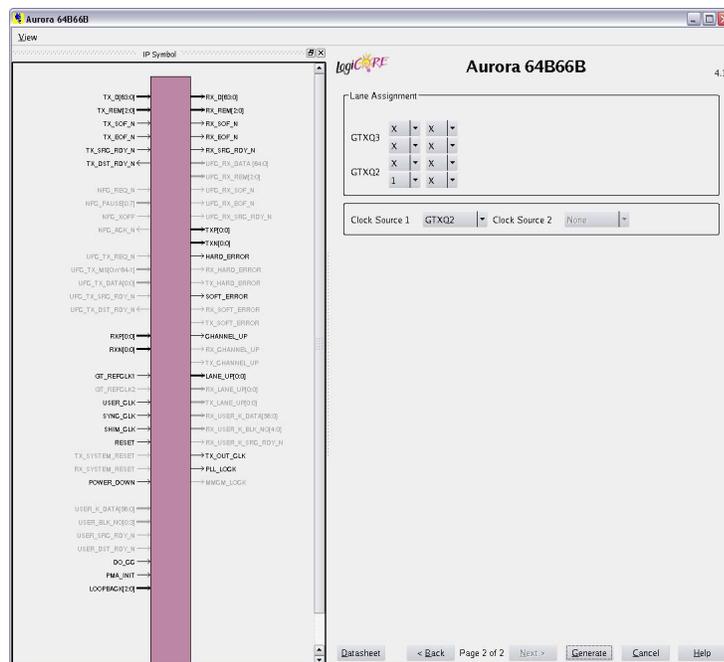


Figure 2-2: Aurora 64B/66B IP Customizer Page 2

The options shown in Figure 2-3 and Figure 2-4 are available only for the Virtex-6 HXT devices.

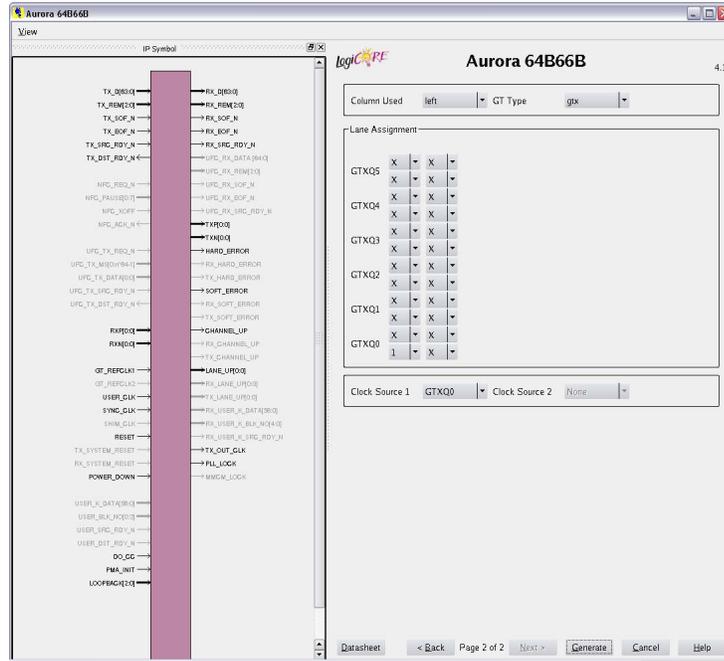


Figure 2-3: Aurora 64B/66B IP Customizer Page 2 (Virtex-6 HXT Devices for GTX Transceivers Only)

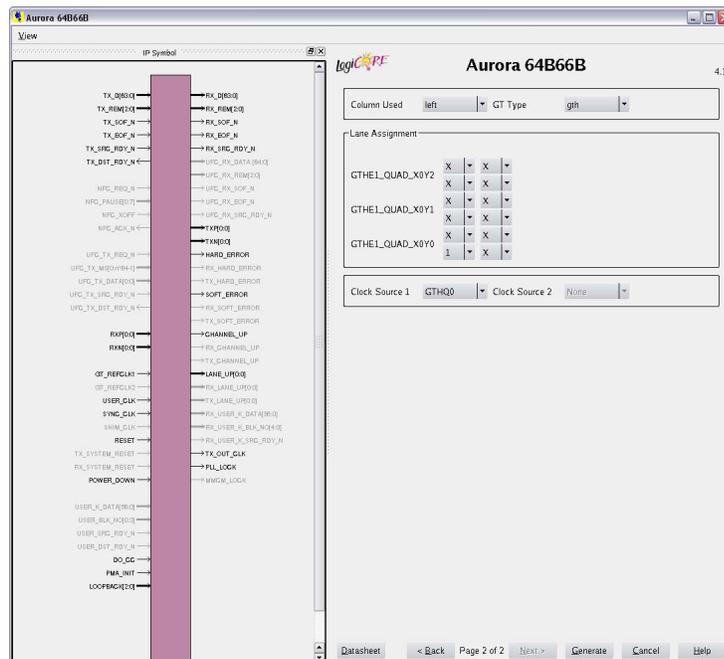


Figure 2-4: Aurora 64B/66B IP Customizer Page 2 (Virtex-6 HXT Devices for GTH Transceivers Only)

Component Name

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

Default: aurora_64b66b_v4_1

Lane Assignment

Refer to the diagram in the information area in [Figure 2-1, page 22](#). Each numbered row represents a GT tile and each active box represents an available GTX/GTH transceiver. For each Aurora lane in the core, starting with Lane 1, select a GTX/GTH transceiver and place the lane by selecting its number in the GTX/GTH placement box.

Aurora Lanes

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

Default: 1

Interface

Select the type of data path interface used for the core. Select Framing to use a LocalLink interface that allows encapsulation of data frames of any length. Select Streaming to use a simple word-based interface with a data valid signal to stream data through the Aurora channel.

Default: Framing

Data Flow Mode

Select the options for direction of the channel the Aurora 64B/66B core will support. Simplex Aurora 64B/66B cores have a single, unidirectional serial port that connects to a complementary simplex Aurora 64B/66B core. Three options are provided as RX-only simplex, TX-only simplex, or RX/TX simplex. These options select the direction of the channel the Aurora 64B/66B core will support. RX/TX simplex creates two cores, one RX and one TX, that share GTX/GTH transceivers.

Duplex - Aurora 64B/66B cores have both TX and the corresponding RX on the other side for communication.

Default: Duplex

Flow Control

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

Available options are:

- UFC only
- Immediate Mode - NFC
- Completion Mode - NFC

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

For the streaming interface, only immediate mode is available. For the framing interface, both immediate and completion modes are available.

Default: None

Line Rate

Enter a floating-point value in gigabits per second. The value entered must be within the valid range shown. This determines the unencoded bit rate at which data is transferred over the serial link.

Default: 3.125 Gbps for GTX transceivers and 10.3125 Gbps for GTH transceivers

GT 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.

Default: 156.25 MHz

Clock Source 1 and Clock Source 2

Select reference clock sources for the GT tiles from the drop-down list in this section.

Default: Clock Source 1: GTXDn/ GTXQn/ GTHQn; Clock Source 2: None

Note: n depends on the GT position.

Column Used

Select appropriate column from the drop down list. This option is applicable only for Virtex-5 TXT and Virtex-6 HXT devices. For other devices, the drop down box is not visible.

Default: left

GT_TYPE

Select the type of GT from the drop down list. This option is applicable only for Virtex-6 HXT devices. For other devices, the drop down box is not visible

Default: gtx

Use ChipScope Pro Analyzer

Select to add ChipScope™ Pro cores (see the *LogiCORE IP Aurora 64B/66B Getting Started Guide*) to the Aurora 64B/66B core. This option provides users a debugging interface that shows the core status signals in the ChipScope Pro analyzer tool.

Default: Unchecked

User K

Select to add User K interface to the core. User K-blocks are special single-block codes passed directly to the user. These blocks are used to implement application-specific control functions.

Default: Unchecked

Generate

Click Generate to generate the core. The modules for the Aurora 64B/66B core are written to the CORE Generator software project directory using the same name as the top level of the core. See the *LogiCORE IP Aurora 64B/66B Getting Started Guide*.

Using the Build Script

A shell script called `implement.sh` and a batch script called `implement.bat` are delivered with the Aurora 64B/66B core in the `implement` subdirectory. These scripts can be used to ease implementation of the Aurora 64B/66B core. Run the script to synthesize the Aurora 64B/66B core using XST. The design runs with the `example_design` module as the top level module, which has built-in frame generator and frame checker modules to generate and verify data integrity. Ensure that the XILINX environment variable is set properly.

Designing with the Core

This section provides a general description of how to use the Aurora 64B/66B core in your designs, and should be used in conjunction with [Chapter 3, “User Interface”](#) that describes core specific interfaces.

General Design Guidelines

All Aurora 64B/66B implementations require careful attention to system performance requirements. Pipelining, logic mappings, placement constraints and logic duplications are all methods that help boost system performance.

Keep It Registered

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

Recognize Timing Critical Signals

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

Use Supported Design Flows

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

Make Only Allowed Modifications

The Aurora 64B/66B core is not user modifiable. Any modifications may have adverse effects on the system timings and protocol compliance. Supported user configurations of the Aurora 64B/66B core can only be made by selecting options from CORE Generator tool.

User Interface

Introduction

An Aurora 64B/66B core can be generated with either a *framing* or *streaming* user data interface. In addition, flow control options are available for designs with framing or streaming interfaces. See [Chapter 4, “Flow Control.”](#)

The framing user interface complies with the *LocalLink Interface Specification* (see [“Related Documents,” page 18](#)). It comprises the signals necessary for transmitting and receiving framed user data. The streaming interface allows users to send data without special frame delimiters. It is simple to operate and uses fewer resources than framing.

Top Level Architecture

Aurora 64B/66B top level (block level) file instantiates Aurora lane module, TX and RX LocalLink modules, global logic module, and wrapper for GTX/GTH transceiver. This top level wrapper file is instantiated in the example design file together with clock, reset circuit and frame generator and checker modules.

[Figure 3-1, page 30](#) shows Aurora 64B/66B top level for a duplex configuration. The top level file is the starting point for a user design.

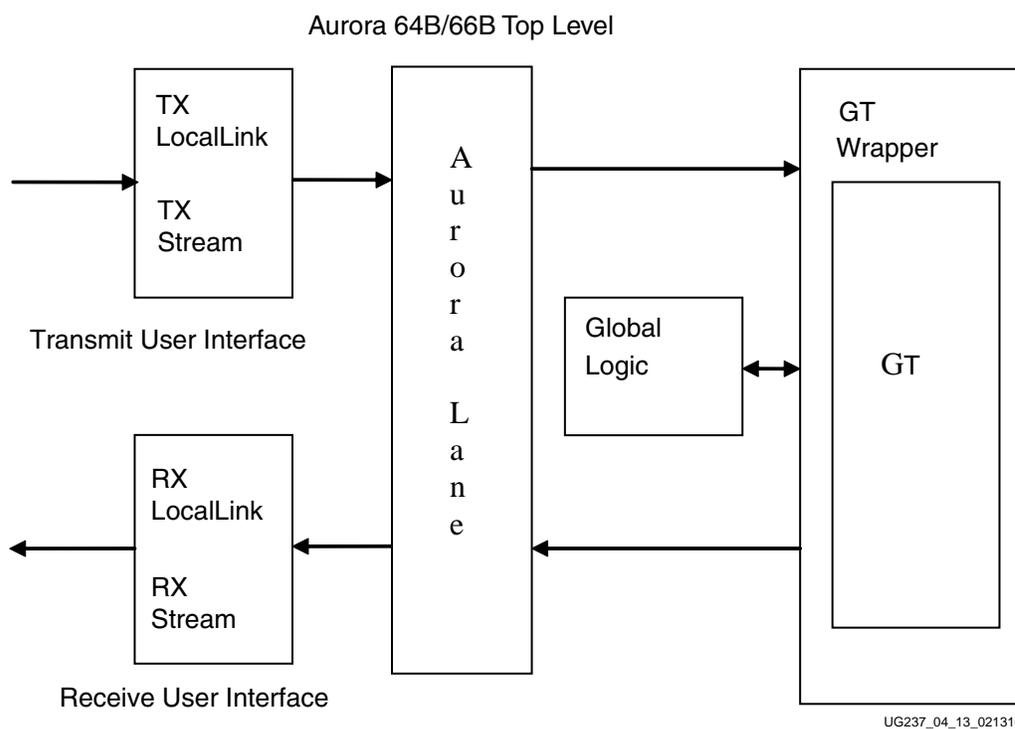
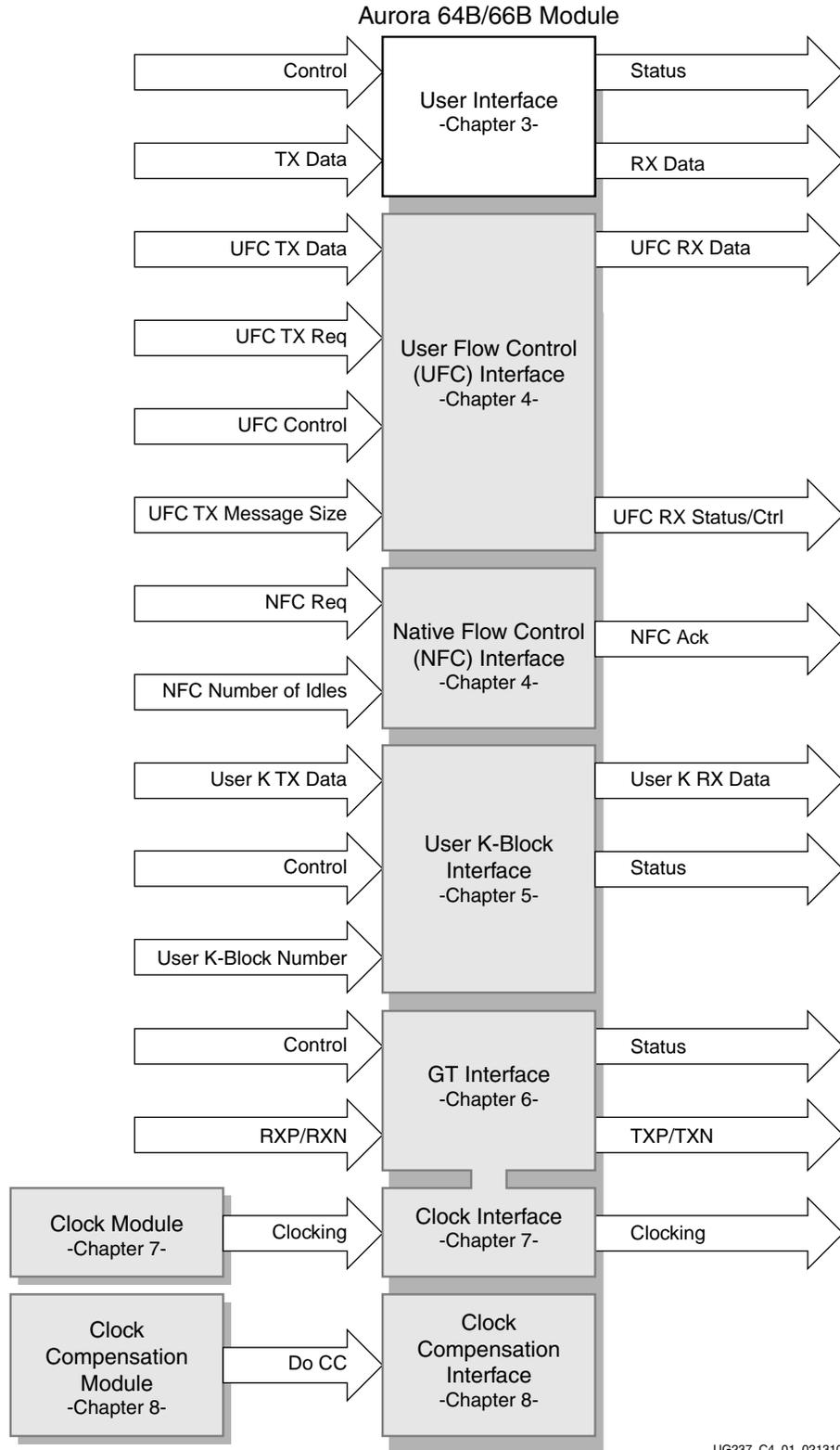


Figure 3-1: Top-Level Architecture

Following sections describe streaming and framing interface in details. User interface logic should be designed such that it complies with timing requirement of the respective interface as explained in the subsequent sections.



UG237_C4_01_021310

Figure 3-2: Top-Level User Interface

Note: The user interface signals vary depending upon the selections made when generating an Aurora 64B/66B core in the CORE Generator™ software.

Framing Interface

Figure 3-3 shows the framing user interface of the Aurora 64B/66B core, with LocalLink-compliant ports for TX and RX data.

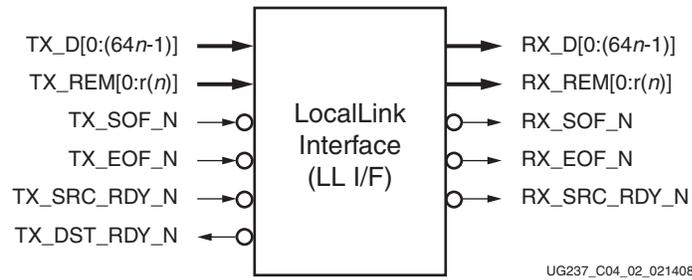


Figure 3-3: Aurora 64B/66B Core Framing Interface (LocalLink)

LocalLink TX Ports

Table 3-1 lists the LocalLink TX data ports and their descriptions.

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

Name	Direction	Description
TX_D[0:(64n-1)]	Input	Outgoing data (Ascending bit order).
TX_DST_RDY_N	Output	Asserted (active-Low) during clock edges when signals from the source will be accepted (if TX_SRC_RDY_N is also asserted). Deasserted (active-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 $r(n) = \text{ceiling} \{ \log_2(64n)/8 \} - 1$.
TX_SOF_N	Input	Signals the start of the outgoing channel frame (active-Low).
TX_SRC_RDY_N	Input	Asserted (active-Low) when LocalLink signals from the source are valid. Deasserted (active-High) when LocalLink control signals and/or data from the source should be ignored (active-Low).

LocalLink RX Ports

Table 3-2 lists the LocalLink RX data ports and their descriptions.

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

Name	Direction	Description
RX_D[0:(64n-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 $r(n) = \text{ceiling} \{ \log_2(64n)/8 \} - 1$.
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 (active-Low) when data and control signals from an Aurora 64B/66B core are valid. Deasserted (active-High) when data and/or control signals from an Aurora 64B/66B 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)
- 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 along with valid no of bytes (RX_REM) during the RX_EOF_N cycle

LocalLink Bit Ordering

The *LocalLink Interface Specification* allows both ascending and descending bit ordering. Aurora 64B/66B cores use ascending ordering. They transmit and receive the most significant bit of the least significant byte first. Figure 3-4 shows the organization of an n -byte example of the LocalLink data interfaces of an Aurora 64B/66B core.

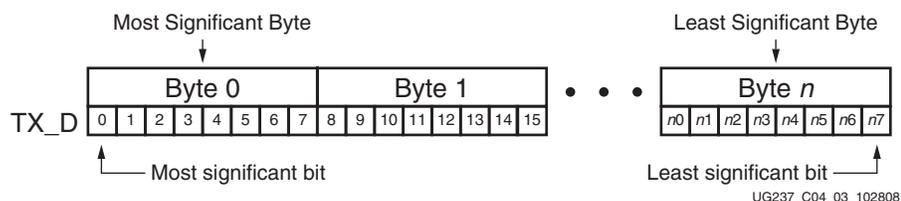


Figure 3-4: LocalLink Interface Bit Ordering

Transmitting Data

LocalLink is a synchronous interface. The Aurora 64B/66B 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: 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. A zero REM value indicates all bytes in the TX_D port are valid. [Table 3-3](#) shows the mapping between TX_REM values and valid byte counts for the TX_D port.

Table 3-3: TX Data Remainder Values

TX_REM Value	Number of Valid Bytes
0	All bytes are valid
1	1
2	2
3	3
.	.
.	.
.	.
<i>n</i>	<i>n</i>

Aurora 64B/66B Frames

The TX_LL submodule translates each user frame that it receives through the TX interface to an Aurora 64B/66B frame. The core starts an Aurora 64B/66B frame by sending a data block with the first word of data, and ends the frame by sending a separator block containing the last bytes of the frame. Idle blocks are inserted whenever data is not available. Blocks are eight bytes of scrambled data or control information with a 2-bit control header (a total of 66 bits). All data in Aurora 64B/66B is sent as part of a data block or a separator block (a separator block consists of a count field, indicating how many bytes are valid in that particular block). [Table 3-4](#) shows a typical Aurora 64B/66B frame with an even number of data bytes.

Length

The user controls the channel frame length by manipulating the TX_SOF_N and TX_EOF_N signals. The Aurora 64B/66B core converts these to data blocks, idle blocks, and separator blocks, as shown in Table 3-4.

Table 3-4: Typical Channel Frame

Data Byte 0	Data Byte 1	Data Byte 2	Data Byte 3	...	Data Byte $n-2$	Data Byte $n-1$	Data Byte n
SEP (1E)	Count (4)	Data Byte 0	Data Byte 1	Data Byte 2	Data Byte 3	x	x

Example A: Simple Data Transfer

Figure 3-5 shows an example of a simple data transfer on a LocalLink interface that is n bytes wide. In this case, the amount of data being sent is $3n$ bytes and so requires three data beats. TX_DST_RDY_N is asserted, indicating that the LocalLink interface is already ready to transmit data. When the Aurora 64B/66B is not sending data, it sends idle blocks.

To begin the data transfer, the user asserts the TX_SOF_N concurrently with TX_SRC_RDY_N and the first n bytes of the user frame. Since TX_DST_RDY_N is already asserted, data transfer begins on the next clock edge. The data bytes are placed in data blocks and transferred through the Aurora 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 0 to indicate that all bytes are valid in the last data beat. The Aurora 64B/66B core sends the final word of data in data blocks, and must send an empty separator block on the next cycle to indicate the end of the frame. 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 separator block to indicate 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 64B/66B core sends idles.

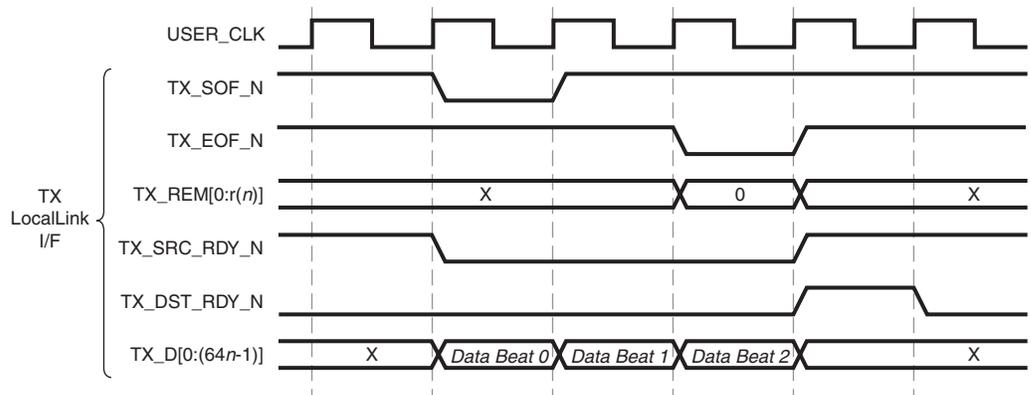


Figure 3-5: Simple Data Transfer

Example B: Data Transfer with Pause

Figure 3-6 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 64B/66B core to ignore all data on the bus and transmit idle blocks instead. The pause continues until TX_SRC_RDY_N is deasserted.

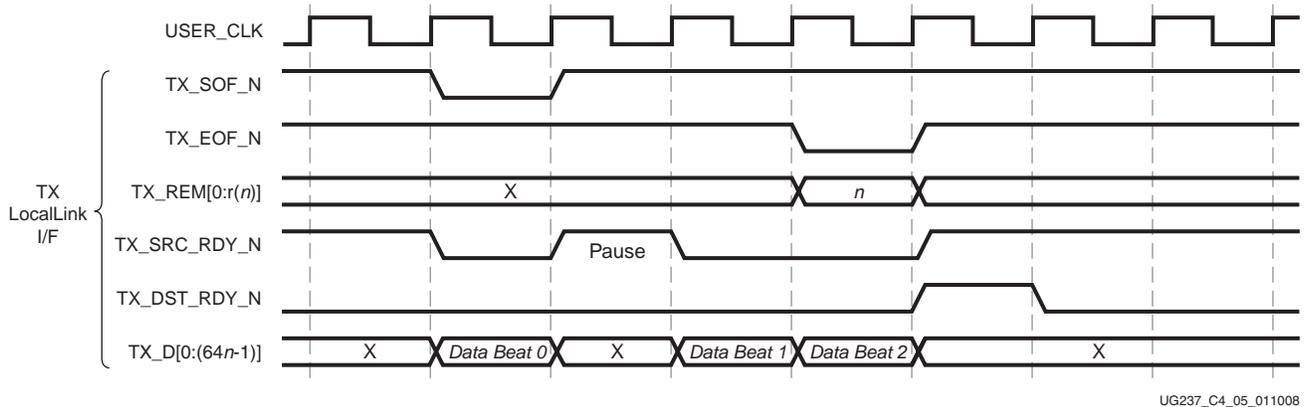
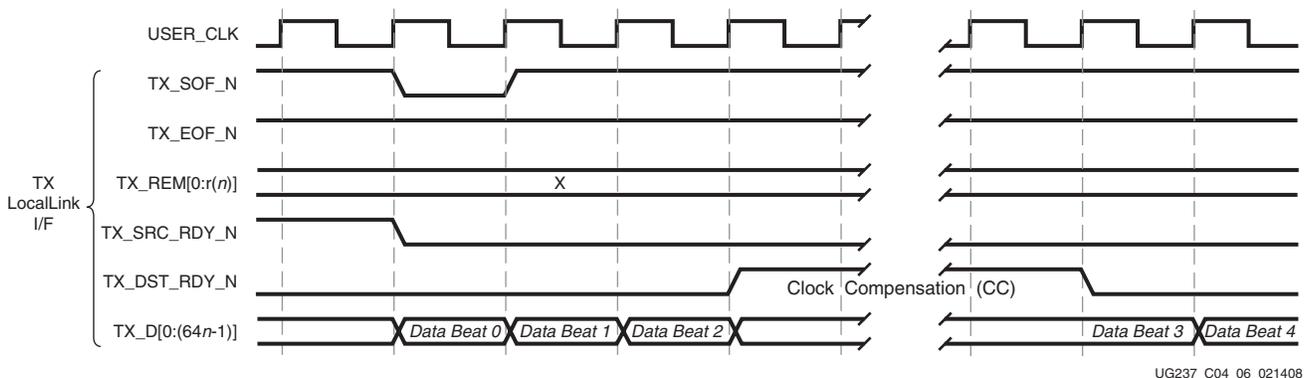


Figure 3-6: Data Transfer with Pause

Example C: Data Transfer with Clock Compensation

The Aurora 64B/66B core automatically interrupts data transmission when it sends clock compensation sequences. The clock compensation sequence imposes three cycles of PAUSE every 10,000 cycles.

Figure 3-7 shows how the Aurora 64B/66B core pauses data transmission during the clock compensation sequence.



Notes:

1. When clock compensation is used, uninterrupted data transmission is not possible. See Chapter 8, “Clock Compensation Interface” for more information about when clock compensation is required.

Figure 3-7: Data Transfer Paused by Clock Compensation

TX Interface Example

This section illustrates a simple example of an interface between a transmit FIFO and the LocalLink interface of an Aurora 64B/66B core.

To review, in order to transmit data, the user asserts TX_SOF_N and TX_SRC_RDY_N. TX_DST_RDY_N indicates that the data on the TX_D bus will be transmitted on the next rising edge of the clock, assuming TX_SRC_RDY_N remains asserted.

Figure 3-8 is a diagram of a typical connection between an Aurora 64B/66B core and the 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, then generates the TX_SOF_N and TX_SRC_RDY_N signals. These two signals cause the Aurora 64B/66B core to start reading the FIFO by asserting the TX_DST_RDY_N signal.

The Aurora 64B/66B core encapsulates the FIFO data and transmits it until the FIFO is empty. At this point, the example application tells the Aurora 64B/66B core to end the transmission using the TX_EOF_N signal.

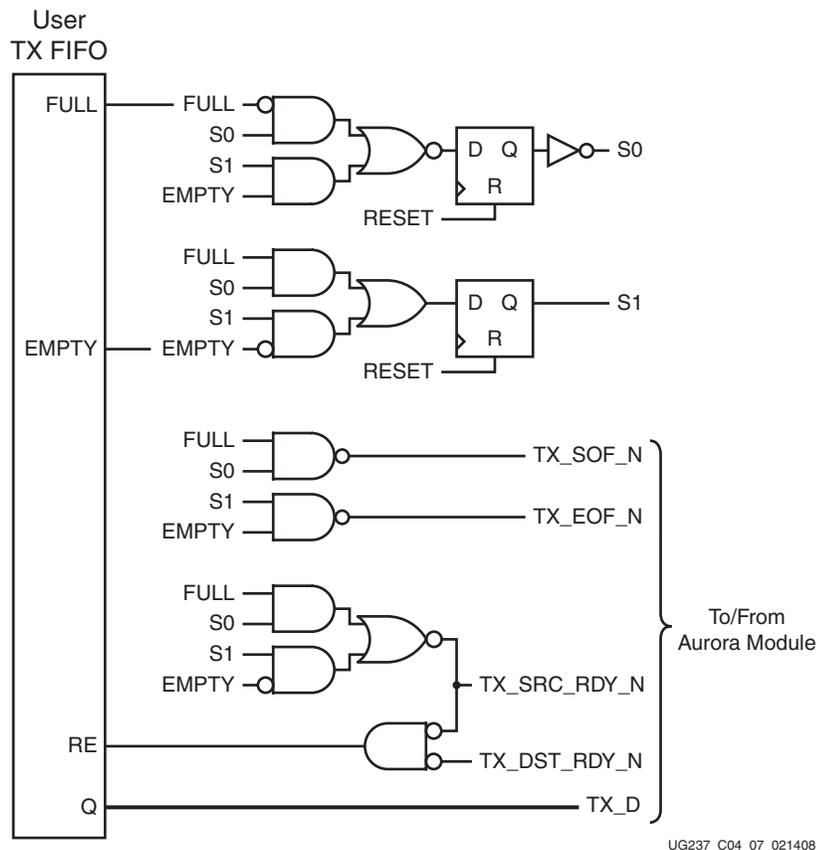


Figure 3-8: Transmitting Data

Receiving Data

When the Aurora 64B/66B core receives an Aurora 64B/66B frame, it presents it to the user through the RX LocalLink interface after discarding the control information, idle blocks, and clock compensation blocks.

The Aurora 64B/66B core has no built-in 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 64B/66B 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 (active-High).

RX_SOF_N is asserted concurrently with the first word of each frame from the Aurora 64B/66B 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 64B/66B core can deassert RX_SRC_RDY_N anytime, even during a frame.

“[Example A: Data Reception with Pause](#)” shows the reception of a typical Aurora 64B/66B frame.

Example A: Data Reception with Pause

[Figure 3-9](#) 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.

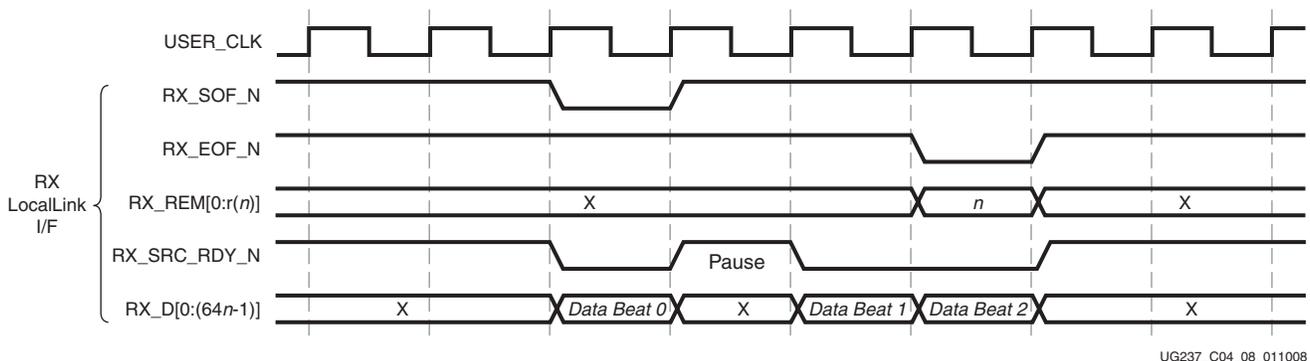


Figure 3-9: Data Reception with Pause

RX Interface Example

The RX LocalLink interface of an Aurora 64B/66B core can be implemented with a simple FIFO. To receive data, the FIFO 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 WE port, the data and framing signals and REM value are written to the FIFO.

Framing Efficiency

There are two factors that affect framing efficiency in the Aurora 64B/66B core:

- Size of the frame
- Data invalid request from gear box that occurs after every 32 USER_CLK cycles

The clock compensation (CC) sequence, which uses three USER_CLK cycles on every lane every 10,000 USER_CLK cycles, consumes about 0.03% of the total channel bandwidth.

The gear box in GTX/GTH transceivers requires periodic pause to account for the clock divider ratio and 64B/66B encoding. This appears as a back pressure in the LocalLink interface and user data needs to be stopped after every 32 cycles. For more information on gear box pause in GTX/GTH, see the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*, the *Virtex-6 FPGA GTX Transceivers User Guide*, and the *Virtex-6 FPGA GTH Transceivers User Guide*.

The Aurora 64B/66B core implements the Strict Aligned option of the Aurora 64B/66B protocol. No data blocks are placed after Idle blocks or SEP blocks on a given cycle. The restriction of not placing data blocks after SEP blocks reduces framing efficiency in a multilane Aurora 64B/66B core.

Example

[Table 3-5](#) is an example calculated after including overhead for clock compensation. It shows the efficiency for a single-lane channel and illustrates that the efficiency increases as frame length increases.

Table 3-5: Efficiency Example

User Data Bytes	Framing Efficiency %
100	96.12
1,000	99.18
10,000	99.89

[Table 3-6](#) shows the overhead in single-lane channel when transmitting 256 bytes of frame data. The resulting data unit is 264 bytes long due to the SEP block used to end the frame. This results in 3.03% overhead in the transmitter. In addition, clock compensation blocks must be transmitted for three cycles every 10,000 cycles, resulting in an additional 0.03% overhead in the transmitter.

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

Lane	Clock	Function
[D0:D7]	1	Channel frame data
[D8:D15]	2	Channel frame data

Table 3-6: Typical Overhead for Transmitting 256 Data Bytes (Cont'd)

Lane	Clock	Function
.		
.		
.		
[D248:D255]	32	Channel frame data
Control block	33	SEP0 block

Streaming Interface

Figure 3-10 shows an example of an Aurora 64B/66B core configured with a streaming user interface.

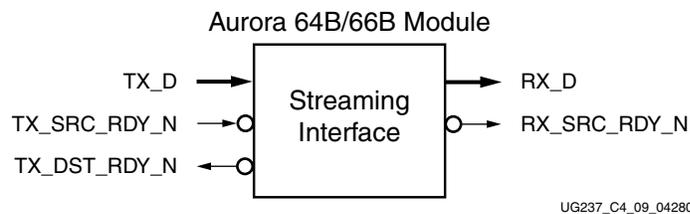


Figure 3-10: Aurora 64B/66B Core Streaming User Interface

Streaming TX Ports

Table 3-7 lists the streaming TX data ports.

Table 3-7: Streaming User I/O Ports (TX)

Name	Direction	Description
TX_D[0:(64n-1)]	Input	Outgoing data (Ascending bit order).
TX_DST_RDY_N	Output	Asserted (active-Low) during clock edges when signals from the source are accepted (if TX_SRC_RDY_N is also asserted). Deasserted (active-High) on clock edges when signals from the source will be ignored.
TX_SRC_RDY_N	Input	Asserted (active-Low) when LocalLink signals from the source are valid. Deasserted (active-High) when LocalLink control signals and/or data from the source should be ignored (active-Low).

Streaming RX Ports

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

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

Name	Direction	Description
RX_D[0:(64n-1)]	Output	Incoming data from channel partner (Ascending bit order).
RX_SRC_RDY_N	Output	Asserted (active-Low) when data and control signals from an Aurora 64B/66B core are valid. Deasserted (active-High) when data and/or control signals from an Aurora 64B/66B 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 active-Low). The streaming Aurora interface expects data to be filled for the entire TX_D port width (integral multiple of eight bytes). The Aurora 64B/66B core deasserts TX_DST_RDY_N (active-High) when the channel is not ready to receive data. Otherwise, TX_DST_RDY_N remains 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 CC logic 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 Interface.”

When data arrives at the RX side of the Aurora channel it is presented on the RX_D bus and RX_SRC_RDY_N 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 3-11, page 42 shows a typical example of streaming data transfer. The example begins with neither of the ready signals asserted, indicating that both the user logic and the Aurora 64B/66B core are not ready to transfer data. During the next clock cycle, the Aurora 64B/66B 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_SRC_RDY_N signal and placing data on the TX_D bus. Because both ready signals are now asserted, data D0 is transferred from the user logic to the Aurora 64B/66B core. Data D1 is transferred on the following clock cycle. In this example, the Aurora 64B/66B core deasserts its ready signal, TX_DST_RDY_N, and no data is transferred until the next clock cycle when, once again, the TX_DST_RDY_N signal is asserted. Then the user deasserts TX_SRC_RDY_N on the next clock cycle, and no data is transferred until both ready signals are asserted.

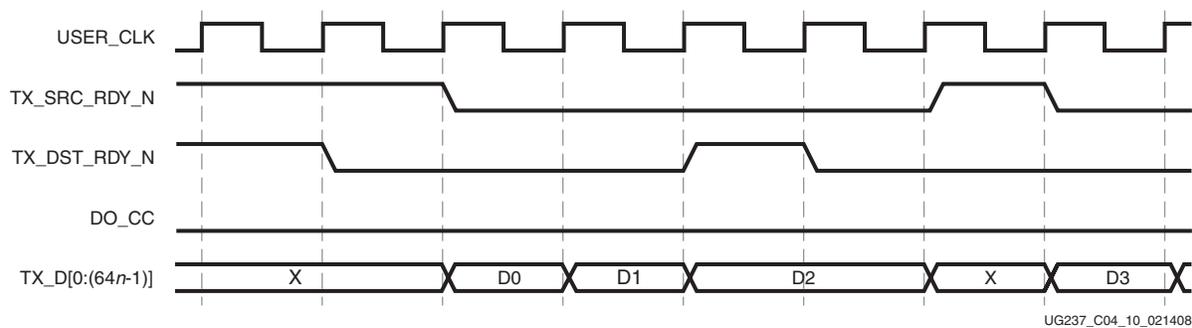


Figure 3-11: Typical Streaming Data Transfer

Figure 3-12, page 42 shows a typical example of streaming data reception.

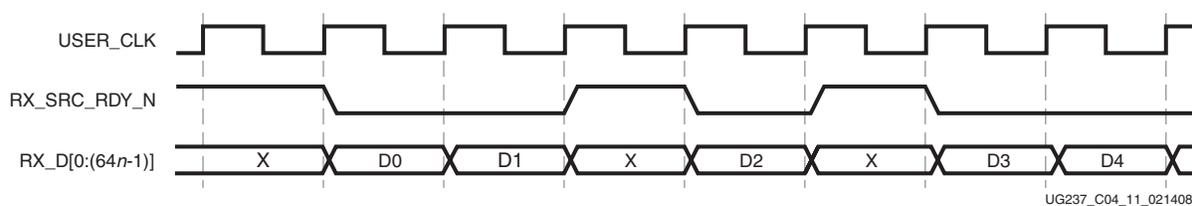


Figure 3-12: Typical Streaming Data Reception

Flow Control

Introduction

This chapter explains how to use Aurora flow control. Two optional flow control interfaces are available. *Native flow control* (NFC) is used for regulating the data transmission rate at the receiving end of a full-duplex channel. *User flow control* (UFC) is used to accommodate high-priority messages for control operations.

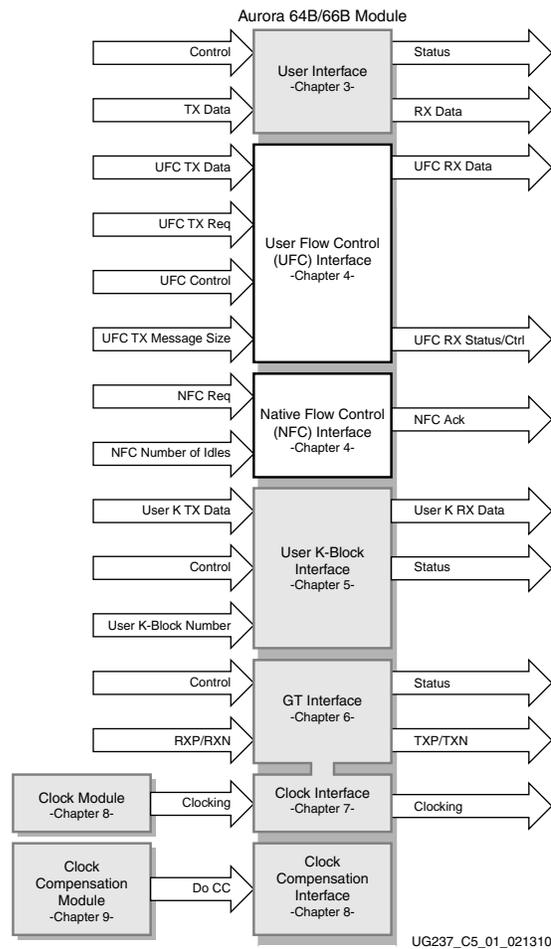


Figure 4-1: Top-Level Flow Control

Native Flow Control

Table 4-1 lists the ports for the native flow control (NFC) interface.

Table 4-1: NFC I/O Ports

Name	Direction	Description
NFC_ACK_N	Output	Asserted when an Aurora 64B/66B core accepts an NFC request (active-Low).
NFC_PAUSE[0:7]	Input	Indicates how many USER_CLK cycles the channel partner must wait before it can send data when it receives the NFC message. Must be held until NFC_ACK_N is asserted. The number of USER_CLK cycles without data is equal to NFC_PAUSE + 1.
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.
NFC_XOFF	Input	Assert to send an XOFF message, requesting that the channel partner stop sending data until it receives a non-XOFF NFC message or reset.

The Aurora 64B/66B protocol includes NFC to allow receivers to control the rate at which data is sent to them by specifying a number of cycles that the channel partner cannot send data. 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, see the *Aurora 64B/66B Protocol Specification*.

To send an NFC message to a channel partner, the user application asserts NFC_REQ_N and writes an 8-bit count to NFC_PAUSE. The NFC_PAUSE code indicates the minimum number of cycles the channel partner must wait after receiving an NFC message before it can resume sending data. The user application must hold NFC_REQ_N, NFC_PAUSE, and NFC_XOFF (if used) until NFC_ACK_N is asserted on a positive USER_CLK edge, indicating the Aurora 64B/66B core will transmit the NFC message. Aurora 64B/66B cores cannot transmit data while sending NFC messages. TX_DST_RDY_N is always deasserted on the cycle following an NFC_ACK_N assertion. NFC Completion mode is available only for the framing Aurora 64B/66B interface.

Example A: Transmitting an NFC Message

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

Note: Signal TX_DST_RDY_N is deasserted for one cycle to create the gap in the data flow in which the NFC message is placed.

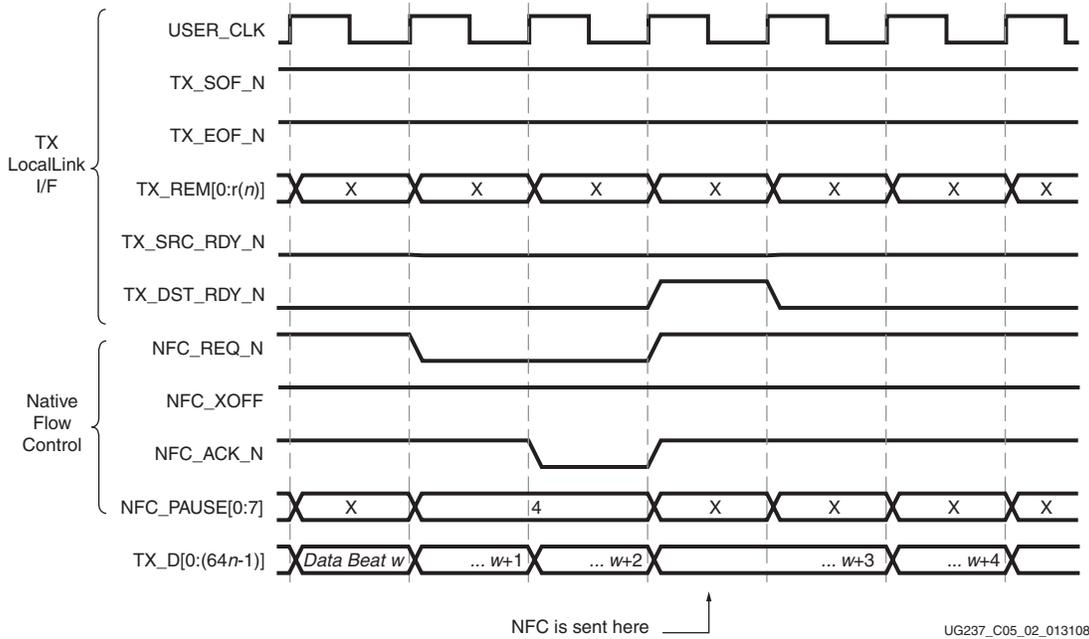


Figure 4-2: Transmitting an NFC Message

Example B: Receiving a Message with NFC Idles Inserted

Figure 4-3 shows an example of the signals on the TX user interface when an NFC message is received. In this case, the NFC message sends the number 8'b01, requesting two cycles without data transmission. The core deasserts TX_DST_RDY_N on the user interface to prevent data transmission for two cycles. In this example, the core is operating in Immediate NFC mode. Aurora 64B/66B cores can also operate in completion mode, where NFC Idles are only inserted before the first data bytes of a new frame. 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.

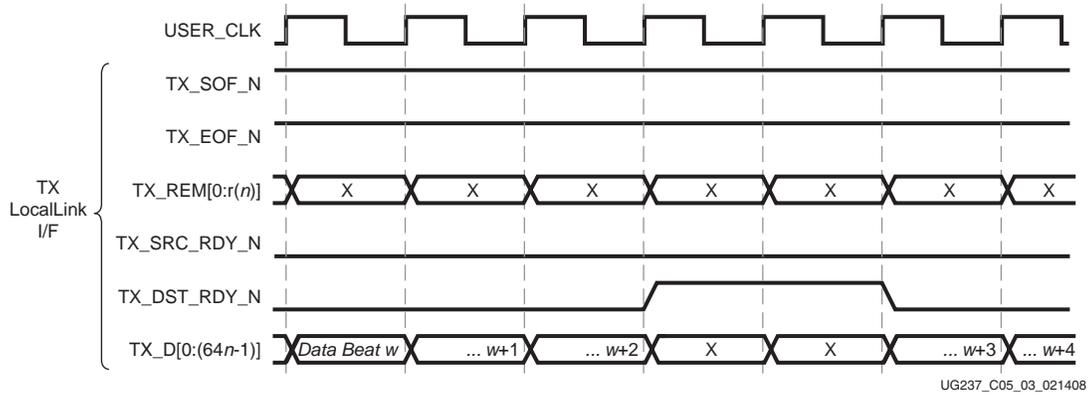


Figure 4-3: Transmitting a Message with NFC Idles Inserted

User Flow Control

The Aurora 64B/66B 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 than frame data. UFC messages are interruptible by high-priority control blocks such as CC/NR/CB/NFC blocks.

Table 4-2 describes the ports for the UFC interface.

Table 4-2: 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). Requests are processed after a single cycle, unless another UFC message is in progress and not on its last cycle. After a request, the UFC_TX_DATA bus will be ready to send data within two cycles unless interrupted by a higher priority event.
UFC_TX_MS[0:7]	Input	Specifies the number of bytes in the UFC message (the message size). The max UFC message size is 256.
UFC_TX_DST_RDY_N	Output	Asserted (active-Low) when an Aurora 64B/66B core is ready to read data from the UFC_TX_DATA interface. This signal is asserted one clock cycle after UFC_TX_REQ_N is asserted and no high priority requests in progress. UFC_TX_DST_RDY_N continues to be asserted while the core waits for data for the most recently requested UFC message. The signal is deasserted for CC, CB, and NFC requests, which are higher priority. While UFC_TX_DST_RDY_N is asserted, TX_DST_RDY_N is deasserted.
UFC_TX_DATA[0:(64n-1)]	Input	Input bus for UFC message data to the Aurora channel. Data is read from the bus into the channel only when both UFC_TX_SRC_RDY_N and UFC_TX_DST_RDY_N are asserted on a positive USER_CLK edge. If the number of bytes in the message is not an integer multiple of the bytes in the bus, on the last cycle, only the bytes needed to finish the message starting from the left of the bus are used.
UFC_TX_SRC_RDY_N	Input	Assert (active-Low) when data on UFC_TX_DATA is valid. If deasserted while UFC_TX_DST_RDY_N is asserted, Idle blocks are inserted in the UFC message.
UFC_RX_DATA[0:(64n-1)]	Output	Incoming UFC message data from the channel partner.
UFC_RX_SRC_RDY_N	Output	Asserted when the values on the UFC_RX_DATA port is valid. When this signal is not asserted, all values on the UFC_RX_DATA port should be ignored (active-Low).
UFC_RX_SOF_N	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. $n = 256$ bytes max. REM bus widths are given $\{\log_2(64n)/8\}-1$.

Transmitting UFC Messages

UFC messages can carry from 1 to 256 data bytes. The user application specifies the length of the message by driving the number of bytes required minus one on the UFC_TX_MS port.

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 for a single cycle. After a request, a new request cannot be made until UFC_TX_DST_RDY_N is asserted for the final cycle of the previous request. The data for the UFC message must be placed on the UFC_TX_DATA port and the UFC_TX_SRC_RDY_N signal must be asserted whenever the bus contains valid message data. The core deasserts TX_DST_RDY_N while sending UFC data, and keeps UFC_TX_DST_RDY_N asserted until it has enough data to complete the message that was requested. If UFC_TX_SRC_RDY_N is deasserted during a UFC message, Idles are sent in the channel, TX_DST_RDY_N remains deasserted, and UFC_TX_DST_RDY_N remains asserted. If a CC request, CB request, or NFC request is made to the core, UFC_TX_DST_RDY_N is deasserted while the requested operation is performed, since CC, CB, and NFC have higher priority.

Example A: Transmitting a Single-Cycle UFC Message

The procedure for transmitting a single cycle UFC message is shown in [Figure 4-4](#). In this case a 4-byte message is being sent on an 8-byte interface.

Note: Signals TX_DST_RDY_N and UFC_TX_DST_RDY_N are deasserted for a cycle before the core accepts message data: this cycle is used to send the UFC header.

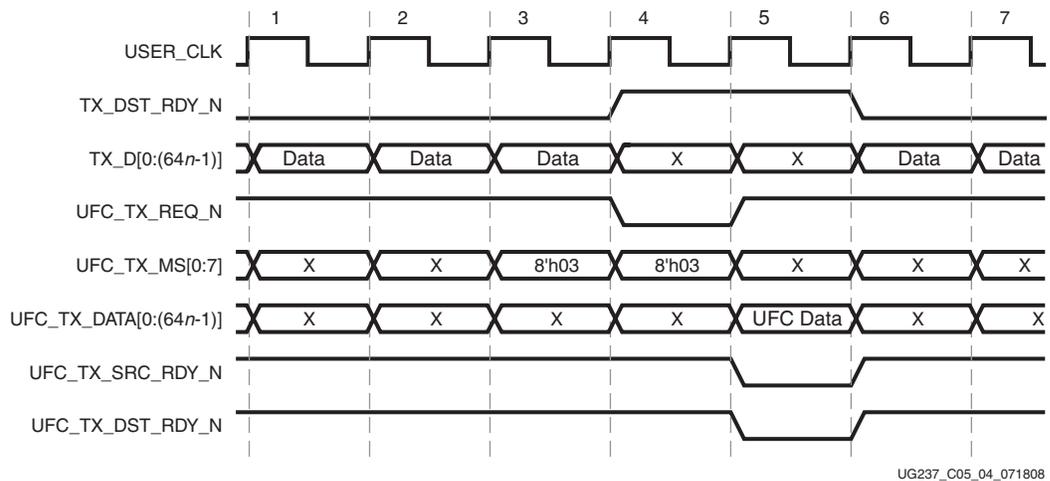


Figure 4-4: Transmitting a Single-Cycle UFC Message

Example B: Transmitting a Multi-Cycle UFC Message

The procedure for transmitting a two-cycle UFC message is shown in [Figure 4-5, page 49](#). In this case the user application is sending a 16-byte message using an 8-byte interface.

UFC_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 UFC data.

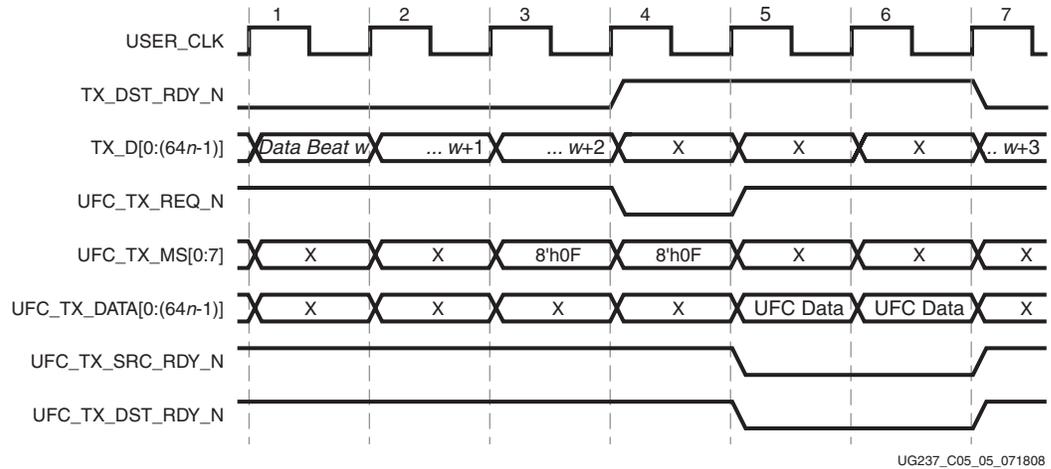


Figure 4-5: Transmitting a Multi-Cycle UFC Message

Receiving User Flow Control Messages

When the Aurora 64B/66B core receives a UFC message, it passes the data from the message to the user application through a dedicated UFC LocalLink interface. The data is presented on the UFC_RX_DATA port; UFC_RX_SOF_N indicates the start of the message data and UFC_RX_EOF_N indicates the end. UFC_RX_REM is used to show the number of valid bytes on UFC_RX_DATA during the last cycle of the message (for example, while UFC_RX_EOF_N is asserted). Signals on the UFC_RX LocalLink interface are only valid when UFC_RX_SRC_RDY_N is asserted.

Example C: Receiving a Single-Cycle UFC Message

Figure 4-6 shows an Aurora 64B/66B 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 4, indicating only the four most significant bytes of the interface are valid.

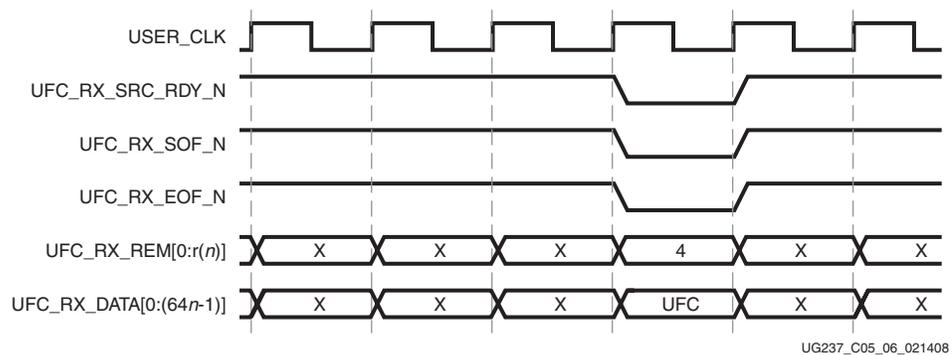


Figure 4-6: Receiving a Single-Cycle UFC Message

Example D: Receiving a Multi-Cycle UFC Message

Figure 4-7 shows an Aurora 64B/66B core with an 8-byte interface receiving a 15-byte message.

Note: The resulting frame is two cycles long, with `UFC_RX_REM` set to 7 on the second cycle indicating that all seven bytes of the data are valid.

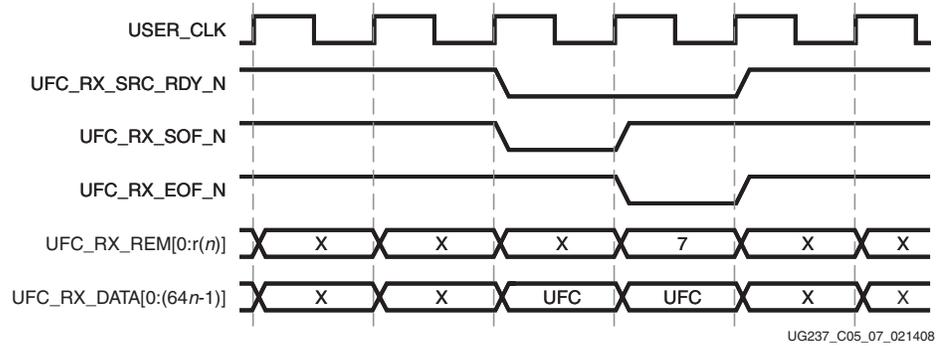


Figure 4-7: Receiving a Multi-Cycle UFC Message

User K-Block Interface

Introduction

This chapter describes short single block data transmission and reception.

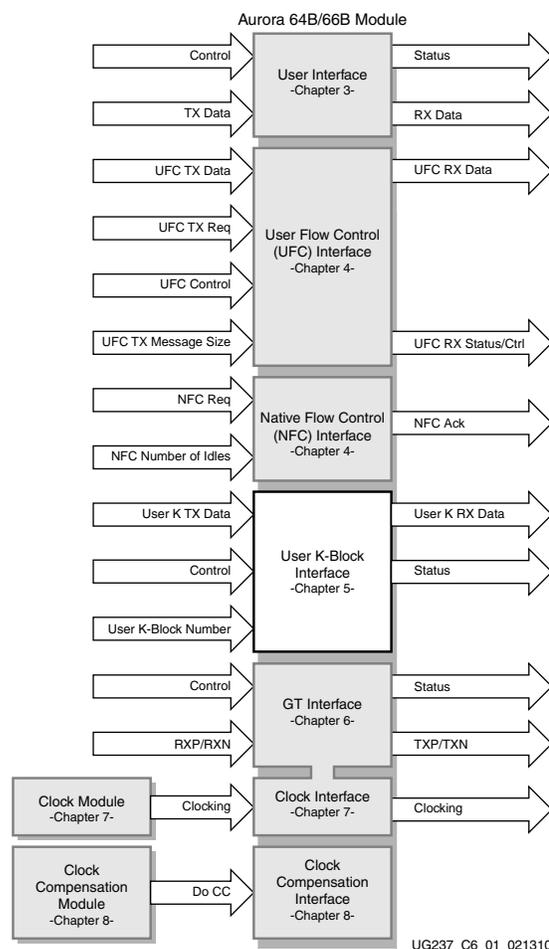


Figure 5-1: Top-Level User K-Block Interface

User K-blocks are special single block codes which includes control blocks that are not decoded by the Aurora interface, but are instead passed directly to the user. These blocks can be used to implement application specific control functions. There are nine available User K-blocks (Table 5-1). They have priority next to UFC but higher than user data.

Table 5-1: Valid Block Type Field Values for User K-Block

User K-Block Name	User K-Block BTF
User K-Block 0	0xD2
User K-Block 1	0x99
User K-Block 2	0x55
User K-Block 3	0xB4
User K-Block 4	0xCC
User K-Block 5	0x66
User K-Block 6	0x33
User K-Block 7	0x4B
User K-Block 8	0x87

The User K-block is not differentiated for streaming or framing designs. Each block code of User K is eight bytes wide and is encoded with a User K BTF, which is indicated by the user as USER_K_BLK_NO. The User K-block is a single block code and is always delineated by USER_K_BLK_NO. It can have only seven bytes of USER_K_DATA.

Table 5-2 lists the ports for the User K-block interface.

Table 5-2: User K-Block I/O Ports

Name	Direction	Description
USER_K_TX_DATA [0:(56n-1)]	Input	User K-block data.
USER_K_TX_BLK_NO [0:3]	Input	User K-block number to be transmitted.
USER_K_TX_SRC_RDY_N	Input	Source ready for User K-block. Asserted when User K data on USER_K_TX_DATA port is valid.
USER_K_TX_DST_RDY_N	Output	Destination ready for User K-block.
USER_K_RX_SRC_RDY_N	Output	Receive source ready for User K-block.
USER_K_RX_DATA [0:(56n-1)]	Output	Receive User K-blocks from the Aurora lane.
USER_K_RX_BLK_NO[0:(4n-1)]	Output	Receive User K-block numbers from Aurora lane.

Transmitting User K-Block

The USER_K_TX_DST_RDY_N is asserted by Aurora and is prioritized by CC, CB, NFC and UFC. After placing the USER_K_TX_DATA and USER_K_TX_BLK_NO along with the USER_K_TX_SRC_RDY_N, the user can change the USER_K_TX_DATA or USER_K_TX_BLK_NO if required when USER_K_TX_DST_RDY_N is asserted (Figure 5-2). This enables Aurora to select appropriate User K BTF among the nine User K-blocks. The data available during assertion of the USER_K_TX_DST_RDY_N is always serviced.

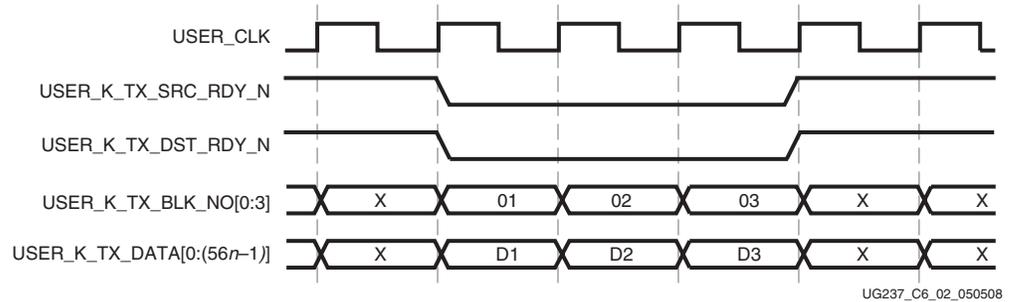


Figure 5-2: Transmitting User K Data and User K-Block Number

Receiving User K-Block

The receive BTF is decoded and the block number for the corresponding BTF is passed on to the user as such (Figure 5-3). The user can validate the USER_K_RX_DATA available on the bus when USER_K_RX_SRC_RDY_N is asserted.

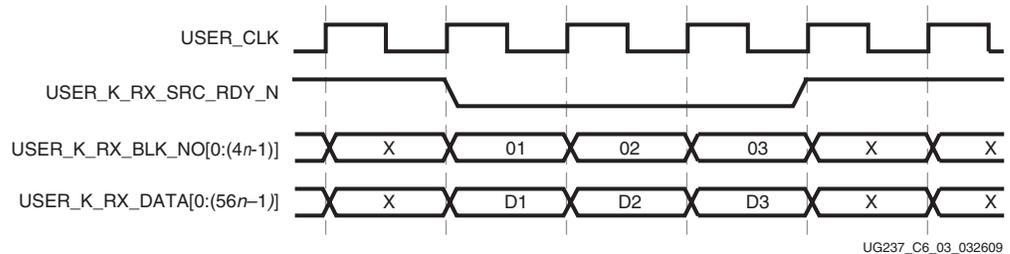


Figure 5-3: Receiving User K Data and User K-Block Number

Status, Control, and the GT Interface

Introduction

The status and control ports of the Aurora 64B/66B core allow user applications to monitor the Aurora channel and use built-in features of the GT interface.

This chapter provides diagrams and port descriptions for the Aurora 64B/66B core’s status and control interface, along with the GTX/GTH serial I/O interface.

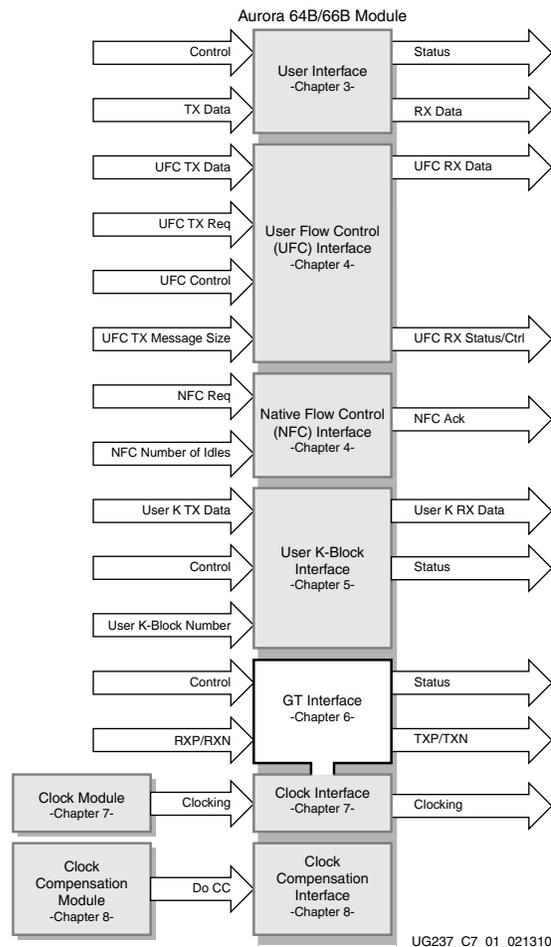


Figure 6-1: Top-Level GTX Interface

Status and Control Ports

Aurora 64B/66B cores are full-duplex/simplex, and provide a TX and an RX Aurora channel connection. The Aurora 64B/66B core does not require any side band signals for simplex mode of operation. Figure 6-2 shows the status and control interface for an Aurora 64B/66B core. Table 6-1 describes the function of each of the ports in the interface.

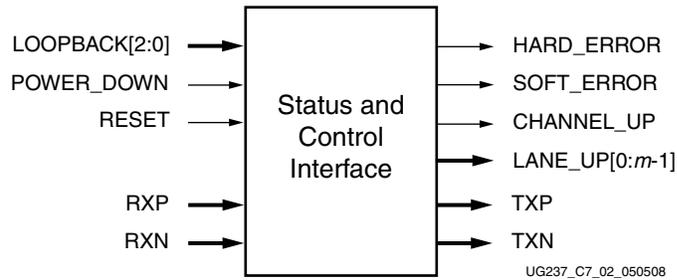


Figure 6-2: Status and Control Interface for the Aurora 64B/66B Core

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.
LANE_UP[0:m-1] ⁽¹⁾	Output	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High). The Aurora 64B/66B core can only receive data after all LANE_UP signals are active-High.
HARD_ERROR	Output	Hard error detected (active-High, asserted until Aurora 64B/66B core resets). See "Error Signals in Aurora 64B/66B Cores," page 58 for more details.
LOOPBACK[2:0]	Input	See the <i>Virtex-5 FPGA RocketIO GTX Transceiver User Guide</i> , the <i>Virtex-6 FPGA GTX Transceivers User Guide</i> , and the <i>Virtex-6 FPGA GTH Transceivers User Guide</i> for details about loopback. See "Related Documents" in Chapter 1.
POWER_DOWN	Input	Drives the power-down input to the GTX/GTH transceiver (active-High).
RESET	Input	Resets the Aurora 64B/66B core (active-High).
SOFT_ERROR	Output	Soft error detected in the incoming serial stream. See "Error Signals in Aurora 64B/66B Cores," page 58 for more details. (active-High, asserted for a single clock).
RXP[0:m-1]	Input	Positive differential serial data input pin.
RXN[0:m-1]	Input	Negative differential serial data input pin.
TXP[0:m-1]	Output	Positive differential serial data output pin.
TXN[0:m-1]	Output	Negative differential serial data output pin.

Notes:

1. m is the number of GTX/GTH transceivers

Table 6-2: Status and Control Ports for Simplex-TX Cores

Name	Direction	Description
TX_CHANNEL_UP	Output	Asserted when Aurora channel initialization is complete and channel is ready to send data.
TX_LANE_UP[0:m-1] ⁽¹⁾	Output	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High). The Aurora 64B/66B core can only transmit data after all TX_LANE_UP signals are active-High.
TX_HARD_ERROR	Output	Hard error detected (active-High, asserted until Aurora 64B/66B core resets). See “Error Signals in Aurora 64B/66B Cores,” page 58 for more details.
POWER_DOWN	Input	Drives the power-down input to the GTX/GTH transceiver (active-High).
TX_SYSTEM_RESET	Input	Resets the Aurora 64B/66B core (active-High).
TX_SOFT_ERROR	Output	Soft error detected in the transmit logic. See “Error Signals in Aurora 64B/66B Cores,” page 58 for more details. (active-High, asserted for a single clock).
TXP[0:m-1]	Output	Positive differential serial data output pin.
TXN[0:m-1]	Output	Negative differential serial data output pin.

Notes:

1. *m* is the number of GTX and GTH transceivers.

Table 6-3: Status and Control Ports for Simplex-RX Cores

Name	Direction	Direction
RX_CHANNEL_UP	Output	Asserted when Aurora channel initialization is complete and channel is ready to receive data.
RX_LANE_UP[0:m-1] ⁽¹⁾	Output	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High). The Aurora 64B/66B core can only receive data after all RX_LANE_UP signals are active-High.
RX_HARD_ERROR	Output	Hard error detected (active-High, asserted until Aurora 64B/66B core resets). See “Error Signals in Aurora 64B/66B Cores,” page 58 for more details.
POWER_DOWN	Input	Drives the power-down input to the GTX/GTH transceiver (active-High).
RX_SYSTEM_RESET	Input	Resets the Aurora 64B/66B core (active-High).
RX_SOFT_ERROR	Output	Soft error detected in the receive logic. See “Error Signals in Aurora 64B/66B Cores,” page 58 for more details. (active-High, asserted for a single clock).
RXP[0:m-1]	Input	Positive differential serial data input pin.
RXN[0:m-1]	Input	Negative differential serial data input pin.

Notes:

1. *m* is the number of GTX and GTH transceivers.

Error Signals in Aurora 64B/66B Cores

Equipment problems and channel noise can cause errors during Aurora channel operation. The 64B/66B encoding allows the Aurora 64B/66B core to detect some 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 high-speed serial GTX/GTH 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 Block Sync algorithm described in the *Aurora 64B/66B Protocol Specification* to determine whether to treat a burst of soft errors as a hard error.

Whenever a hard error is detected, the Aurora 64B/66B core automatically resets itself and attempts to reinitialize. In most cases, this allows 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 block sync state machine.

Table 6-4 summarizes the error conditions that the Aurora 64B/66B core can detect and the error signals used to alert the user application.

Table 6-4: Error Signals in Full-Duplex Cores

Signal	Description
HARD_ERROR/ TX_HARD_ERROR/ RX_HARD_ERROR	<p>TX Overflow/Underflow: The elastic buffer for TX data overflows or underflows. This can occur when the user clock and the reference clock sources are not running at the same frequency.</p> <p>RX Overflow/Underflow: The clock correction and channel bonding FIFO for RX data overflows or underflows. This can occur when the clock source frequencies for the two channel partners are not within ± 100 ppm.</p> <p>Soft Errors: There are too many soft errors within a short period of time. The block sync state machine used for alignment automatically attempts to realign if too many invalid sync headers are detected.</p>
SOFT_ERROR/ TX_SOFT_ERROR/ RX_SOFT_ERROR	<p>Invalid SYNC Header: The 2-bit header on the 64-bit block was not a valid control or data header.</p> <p>Invalid BTF: A control block was received with an unrecognized value in the block type field (BTF). This is usually the result of a bit error.</p>

Initialization

Aurora 64B/66B cores initialize automatically after power up, reset, or hard error. Aurora 64B/66B core 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.

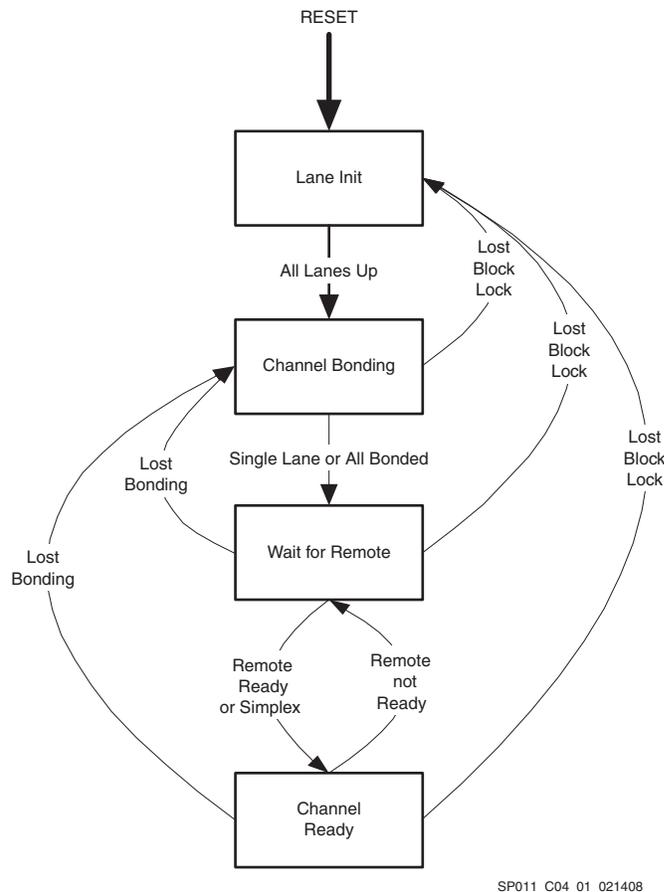


Figure 6-3: Initialization Overview

Aurora 64B/66B 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 Aurora 64B/66B cores do not have any sideband connection and use timers to declare that the partner is out of initialization and is ready for data transfer. These simplex cores transmit periodic channel bonding characters to ensure the channel partner is bonded. If at any time during the data transfer, the link is broken or reinitialized, the

channel will auto recover after the periodic channel bond character is sent to the partner and does not require any reset.

After the link is broken, the Aurora 64B66B initialization state machine (Figure 6-3, page 59) will move from Channel Ready state to Wait for Remote state. It waits in this state until the periodic Channel Bond character is received by the partner and then moves to the Channel Ready state. The data transferred during the reinitialization process will be lost.

The user can modify the timer value based on their channel requirement. For simulation, the timer value (-GSIMPLEX_TIMER) can be modified on the simulate_mti.do to test for different timer values. For implementation, the SIMPLEX_TIMER_VALUE parameter must be modified on <component name>/example_design/<component name>_block.v[hd].

Reset and Power Down

Reset

The RESET/ TX_SYSTEM_RESET/ RX_SYSTEM_RESET signals on the control and status interface are used to set the Aurora 64B/66B 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.

Power Down

When POWER_DOWN is asserted, the GTX/GTH transceivers in the Aurora 64B/66B 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 Chapter 7, “Clock Interface and Clocking”) TX_OUT_CLK stops when the GTX/GTH transceivers are powered down. See the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*, *Virtex-6 FPGA GTX Transceivers User Guide*, and the *Virtex-6 FPGA GTH Transceivers User Guide* for details about powering down GTX/GTH transceivers.

Timing

Figure 6-4 shows the timing for the RESET signal. In a quiet environment, t_{CU} is generally less than 500 clocks; In a noisy environment, t_{CU} can be much longer.

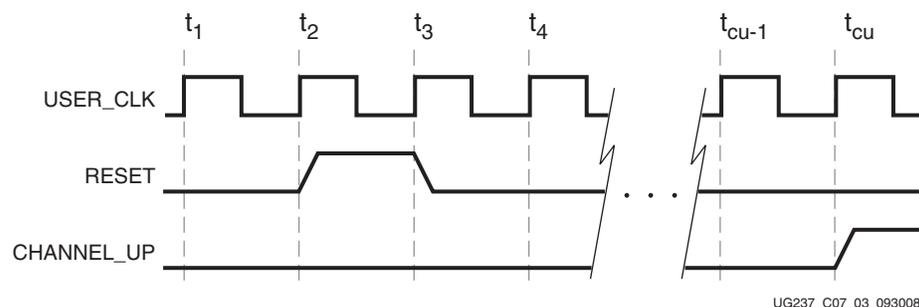


Figure 6-4: Reset and Power Down Timing

Clock Interface and Clocking

Introduction

Good clocking is critical for the correct operation of the Virtex®-5 and Virtex-6 FPGA Aurora 64B/66B core. The core requires a low-jitter reference clock to drive the high-speed TX clock and clock recovery circuits in the GTX/GTH transceiver. It also requires at least one frequency locked parallel clock for synchronous operation with the user application.

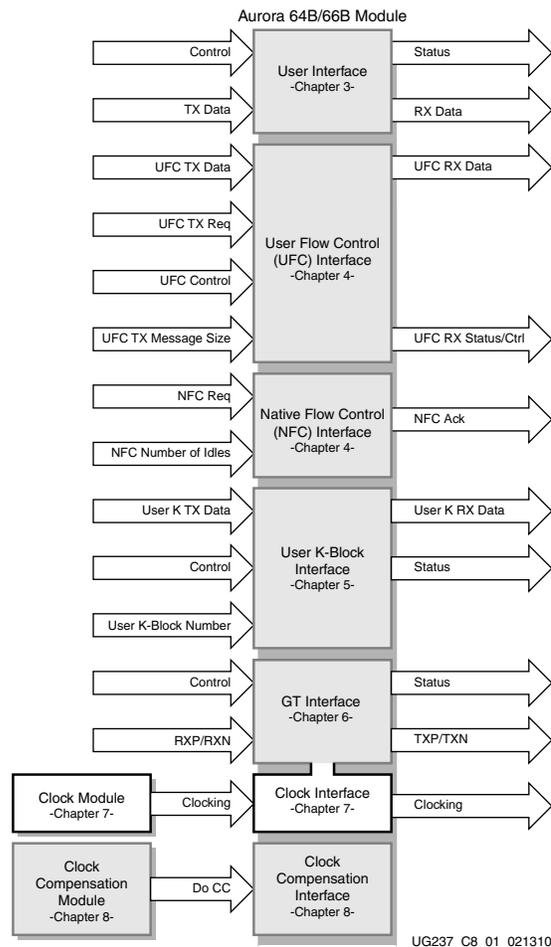


Figure 7-1: Top-Level Clocking

Each Aurora 64B/66B core is generated with an example directory that includes a design called `example_design`. This design instantiates the Aurora 64B/66B core that was generated and demonstrates a working clock configuration for the core. First-time users should examine the `aurora` example design and use it as a template when connecting the clock interface.

Clock Interface Ports for Virtex-5 and Virtex-6 FPGA Cores

[Table 7-1](#) describes the Virtex-5 and Virtex-6 FPGA Aurora 64B/66B core clock ports. In GTX/GTH transceiver designs, the reference clock can be from GTXD/GTXQ/GTHQ, which is a differential input clock for each GT tile. The reference clock for GT tile is provided through the CLKIN port.

Table 7-1: Clock Ports for a Virtex-5 and Virtex-6 FPGA Aurora 64B/66B Core

Name	Direction	Description
DCM_NOT_LOCKED/ MMCM_NOT_LOCKED	Input	If a PLL is used to generate clocks for the Aurora 64B/66B core, the DCM_NOT_LOCKED/MMCM_NOT_LOCKED signal should be connected to the inverse of the PLL LOCKED signal. The clock modules provided with the Aurora 64B/66B core use the PLL for clock division. The DCM_NOT_LOCKED/MMCM_NOT_LOCKED signal from the clock module should be connected to the DCM_NOT_LOCKED/MMCM_NOT_LOCKED signal on the Aurora 64B/66B core.
USER_CLK	Input	Parallel clock shared by the Aurora 64B/66B core and the user application. The USER_CLK is the output of a BUFG whose input is derived from TX_OUT_CLK. The rate is one fourth the rate of TX_OUT_CLK, which is determined by the clock rate settings of the RocketIO transceivers in the core.
TX_OUT_CLK	Output	Clock signal from Virtex-5/Virtex-6 FPGA GTX/GTH transceivers. The GTX/GTH transceiver generates TX_OUT_CLK from its reference clock based on its PLL speed setting. This clock, when buffered, should be used as the user clock for logic connected to the Aurora 64B/66B core.
SHIM_CLK	Input	Parallel clock used by the internal synchronization logic of the RocketIO transceivers in the Aurora 64B/66B core. SHIM_CLK is double the rate of USER_CLK. Note: This clock is not available for Virtex-6 FPGA GTH transceivers
SYNC_CLK	Input	Parallel clock used by internal synchronization logic of the RocketIO transceivers in the Aurora 64B/66B core. SYNC_CLK is double the rate of SHIM_CLK.
PLL_LOCK	Output	Active-High, asserted when TX_OUT_CLK is stable. When this signal is deasserted (Low), circuits using TX_OUT_CLK should be held in reset.

Parallel Clocks for Virtex-5/Virtex-6 FPGA Designs

Connecting USER_CLK, SYNC_CLK, SHIM_CLK, and TX_OUT_CLK

The Virtex-5/Virtex-6 FPGA Aurora 64B/66B cores use three phase-locked parallel clocks. The first is USER_CLK, which synchronizes all signals between the core and the user application. All logic touching the core must be driven by USER_CLK, which in turn must be the output of a global clock buffer (BUFG). USER_CLK runs at one-fourth the rate of TX_OUT_CLK, which is determined by the Clock Settings for the design. The

TX_OUT_CLK is selected so that the data rate of the parallel side of the module matches the data rate of the serial side of the module, taking into account 64B/66B encoding and decoding.

The second phase-locked parallel clock is SHIM_CLK. This clock must also come from a BUFG and is half the rate of the TX_OUT_CLK. It is connected directly to the Aurora 64B/66B core to drive the internal synchronization logic of the high-speed serial GTX/GTH transceivers.

The third phase-locked parallel clock is SYNC_CLK. This clock must also come from a BUFG and is equal to the rate of TX_OUT_CLK. It is also connected to the Aurora 64B/66B core to drive internal synchronization logic of RocketIO transceiver.

To make it easier to use the two parallel clocks, a clock module is provided in a subdirectory called clock_module under example_design. The ports for this module are described in [Table 7-1, page 62](#); If the clock module is used, the DCM_NOT_LOCKED/MMCM_NOT_LOCKED signal should be connected to the DCM_NOT_LOCKED/MMCM_NOT_LOCKED output of the clock module, TX_OUT_CLK should connect to the clock module's CLK port, and PLL_LOCK should connect to the clock module's PLL_NOT_LOCKED port. If the clock module is not used, connect the DCM_NOT_LOCKED/MMCM_NOT_LOCKED signal to the inverse of the LOCKED signal from any PLL used to generate either of the parallel clocks, and use the PLL_LOCK signal to hold the PLLs in reset during stabilization if TX_OUT_CLK is used as the PLL's source clock.

Reference Clocks for Virtex-5/Virtex-6 FPGA Designs

Aurora 64B/66B cores require low-jitter reference clocks for generating and recovering high-speed serial clocks in the GTX/GTH transceivers. Each reference clock can be set to Virtex-5/Virtex-6 FPGA reference clock input ports: GTXD/GTXQ/GTHQ. Reference clocks should be driven with high-quality clock sources whenever possible to decrease jitter and prevent bit errors. DCMs should never be used to drive reference clocks, because they introduce too much jitter.

For multi-lane designs, the Aurora 64B/66B wizard allows selecting clocks three tiles above and three tiles below the selected tile per north-south clocking criteria. A second reference clock source can be selected if the tile selection exceeds the 7-tile boundary. For details on north-south clocking, see the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*, the *Virtex-6 FPGA GTX Transceivers User Guide*, and the *Virtex-6 FPGA GTH Transceivers User Guide*.

Clock Compensation Interface

Introduction

Clock compensation is a feature that allows up to ± 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.

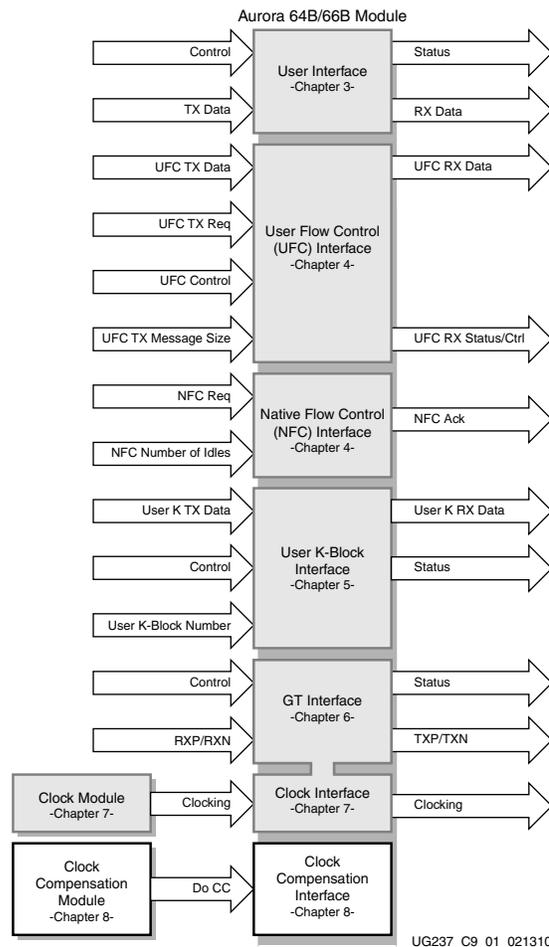


Figure 8-1: Top-Level Clock Compensation Interface

The Aurora 64B/66B 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 64B/66B 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.

Clock Compensation Interface

All Aurora 64B/66B 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 64B/66B 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.

[Figure 8-2](#) and [Figure 8-3](#) are waveform diagrams showing how the DO_CC signal works.

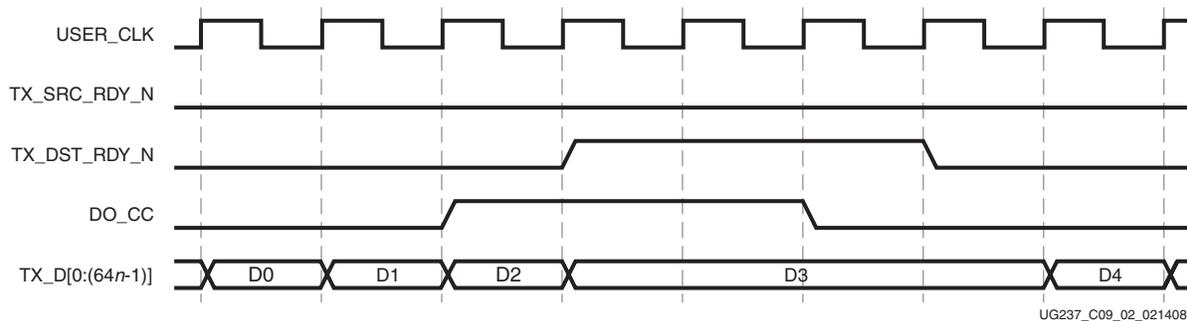


Figure 8-2: Streaming Data with Clock Compensation Inserted

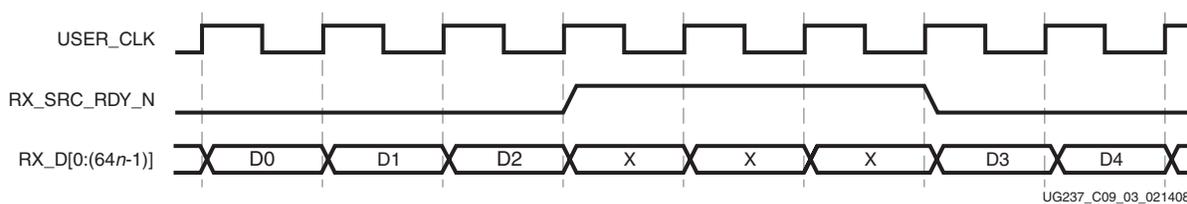


Figure 8-3: Data Reception Interrupted by Clock Compensation

The Aurora protocol specifies a clock compensation mechanism that allows up to ± 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 three cycles every 10,000 cycles. While DO_CC is asserted, TX_DST_RDY_N is deasserted on the TX user interface while the channel is being used to transmit clock compensation sequences.

A standard clock compensation module is generated along with each Aurora 64B/66B core from the CORE Generator™ software, in the cc_manager subdirectory under example_design. It automatically generates pulses to create Aurora compliant clock compensation sequences on the DO_CC port. This module always be connected to the

clock compensation port on the Aurora module, except in special cases. [Table 8-2](#) shows the port description for the standard CC module.

Table 8-2: Standard CC I/O Port

Name	Direction	Description
DO_CC	Output	Connect this port to the DO_CC input of the Aurora 64B/66B 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 TX-only simplex or an RX/TX simplex 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 66](#)) because the reference clock frequencies on both sides of the module are locked. In this case, DO_CC should both be tied to ground.

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 three 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 compensation sequences within 8 cycles of one another.

Two Aurora 64B/66B Cores in Virtex-5 Family Sharing a RocketIO Transceiver Tile

The RocketIO™ transceivers in the Virtex®-5 family have dual architecture wherein two transceivers in each tile share a common PLL (see the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*). Aurora 64B/66B core requires one transceiver per lane and the transceiver can be mapped to GTX0 or GTX1 in a tile. The unused transceiver is powered down during core generation. In order to use the powered down transceiver for another Aurora 64B/66B core sharing a common reference clock source, the user has to bring out the unused transceiver ports to the Aurora 64B/66B GTX wrapper file.

Aurora 64B/66B core instantiates GTX_DUAL primitive in <component name>_tile.v[hd] file which is located in <component name>/example_design/gtx/. This file contains GTX0 and GTX1 specific ports together with shared tile ports. This file is hierarchically called in <component name>_gtx_wrapper.v[hd]. User has to bring out unused transceiver ports to <component name>_gtx_wrapper.v[hd] file which is instantiated in the Aurora 64B/66B block level file. See [Figure A-1, page 71](#) for file level hierarchy of Aurora 64B/66B core.

Shared Ports and Attributes

Aurora 64B/66B automatically sets shared PLL ports and attributes based on the CORE Generator™ software options. Reference clock input and GTX reset to a tile is driven from example design. Aurora 64B/66B core derives clock from TXOUTCLK0/1 and based on GTX0 or GTX1 selection, TXOUTCLK0 or TXOUTCLK1 is fed to the PLL input for generation of core clock. The user can use the same TXOUTCLK source used in core 1 (see [Figure A-1, page 71](#)) or optionally can use other TXOUCLK port for core 2. Since reference clock and GTX reset ports are common for GTX0 and GTX1, application of GTX reset will reset both the cores.

Steps to Modify Transceiver Specific Ports and Attributes

When user shares a tile across two Aurora 64B/66B core, shared PLL settings are not user modifiable. Follow the following steps in order to modify GTX0 or GTX1 specific ports and attributes

1. Generate an Aurora 64B/66B v3.1 core (core 1) using CORE Generator software. Select transceiver locations such that GTX0 or GTX1 in a tile is used for Aurora 64B/66B core. Generate another Aurora 64B/66B core (core 2) such that unused transceiver (GTX1 or GTX0) in core 1 is used for core 2.
2. Go to directory <component name>/example_design/gtx/ of core 1.
3. Open <component name>_tile.v[hd] file.
4. If line rate selection for Aurora 64B/66B core 2 is different from that of core 1, modify PLL_TXDIVSEL_OUT_1 and PLL_RXDIVSEL_OUT_1 with appropriate value in case GTX0 is selected for core 1. Modify PLL_TXDIVSEL_OUT_0 and PLL_RXDIVSEL_OUT_0 in case GTX1 is selected for core 1. See the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* for valid values of attributes PLL_TXDIVSEL_OUT_0/1 and PLL_RXDIVSEL_OUT_0/1. User does not have to modify any other attribute.

Steps to Bring Out Unused Transceiver Ports to Aurora 64B/66B GTX Wrapper File

Follow the following steps to bring out unused transceiver ports to Aurora 64B/66B <component name>_gtx_wrapper.v[hd] file:

1. Go to directory <component name>/example_design/gtx/ of core 1.
2. Open <component name>_gtx_wrapper.v[hd] file.
3. Search for GTX_TILE_INST in <component name>_gtx_wrapper.v[hd].
For a multi-lane Aurora 64B/66B core, search for <component name>_TILE_INST_LANE*n* (*n*=1,2,3etc). Select the instance in which one transceiver is used for Aurora 64B/66B core and the other is powered down.
4. If GTX0 is used for Aurora 64B/66B core, attributes and ports which have "1" appended are hard coded to default values. If GTX1 is used for Aurora 64B/66B core, attributes and ports that have "0" appended are hard coded to default values. Port map all such hard coded ports and attributes to new ports and attributes and declare them in module list (Verilog) or entity list (VHDL).

Steps to Bring Out Aurora 64B/66B Core Specific Logic to GTX Wrapper File

Follow the following steps to copy Aurora 64B/66B core specific logic:

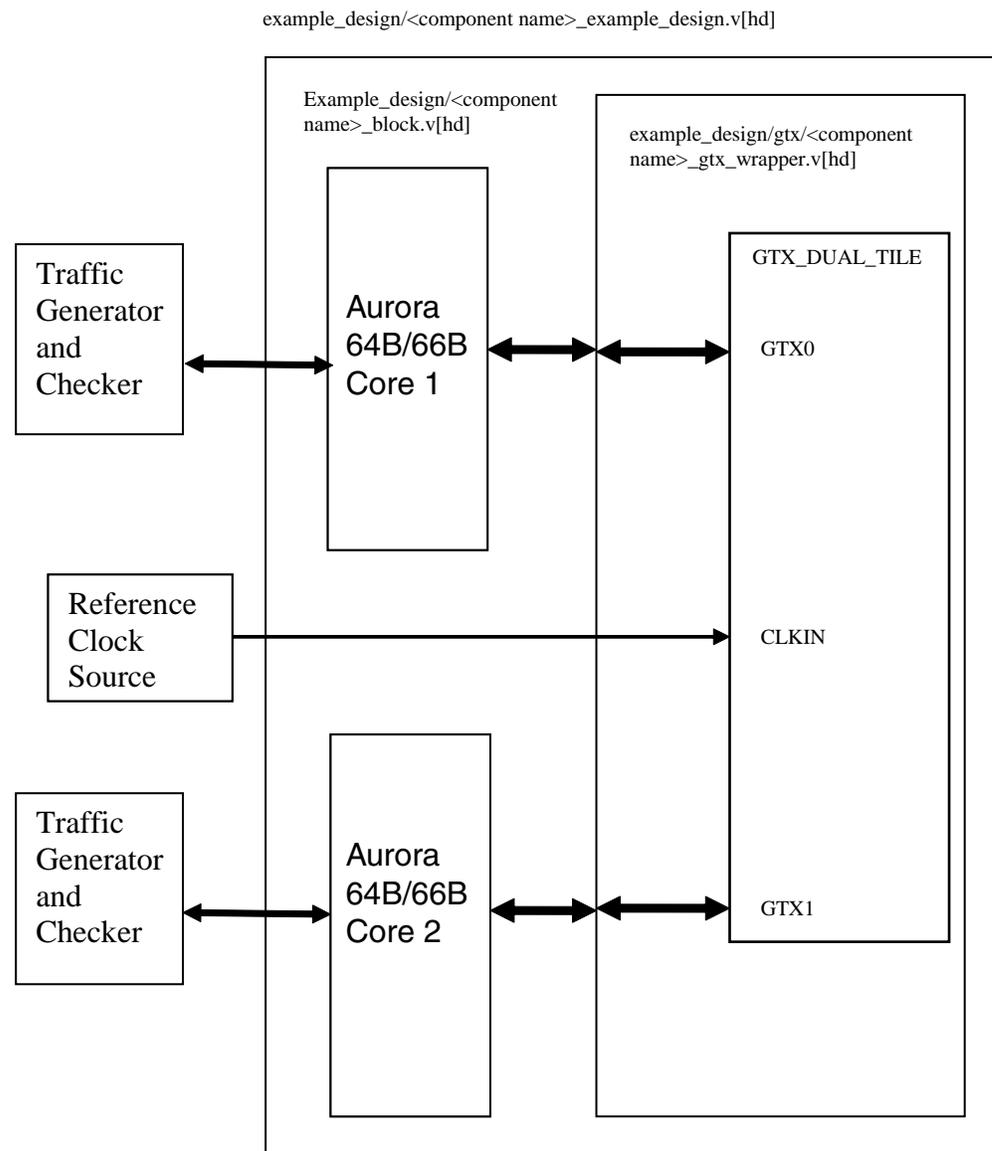
1. Go to directory <component name>/example_design/gtx/ of core 2.
2. Open <component name>_gtx_wrapper.v[hd] file.
3. Copy instances GTX_WIDTH_AND_CLK_CONV_TX, SCRAMBLER_64B66B, DESCRAMBLER_64B66B, BLOCK_SYNC_SM, GTX_WIDTH_AND_CLK_CONV_RX and GTX_WIDTH_AND_CLK_CONV_RX along with associated logic to <component name>_gtx_wrapper.v[hd] of core 1. Map ports appropriately as described in step 4.

Steps to Instantiate Two Aurora 64B/66B Cores in the Top Level File

Follow the following steps to instantiate two Aurora 64B/66B cores in the top level file:

1. Go to directory <component name>/example_design of core 1.
2. Open < component name>_block.v(hd).
3. Search gtx_wrapper_i instance. Add all new ports and attributes in step 5 in the port map list. Finally map these new ports to another Aurora 64B/66B core (core 2 in Figure A-1).

Figure A-1 shows two single lane Aurora 64B/66B cores sharing a tile in Virtex-5 family.



UG237_A_01_071109

Figure A-1: Example Showing Two Single Lane Aurora 64B/66B Cores Sharing a Transceiver Tile in Virtex-5 Family

Generating a GTX Wrapper File from the GTX Transceiver Wizard

Follow the following steps to generate transceiver wrapper file using RocketIO™ transceiver wizard:

1. Using CORE Generator™ tool, run the Virtex-5 FPGA RocketIO and Virtex-6 FPGA GTX transceiver wizard.
Make the component name match the Aurora 64B/66B core.
2. Select transceivers and clock source based on application requirement.
3. Select internal data width as 16 bits for Aurora 64B/66B protocol.
4. Select TX encoding as 64B/66B with Ext Seq Ctr and data path width as 32. Same settings should be selected for RX.
5. Select RXRESET, RXRECCLK, RXBUFSTATUS, RXBUFRESET, TXOUTCLK, TXRESET, TXPOLARITY, TXENPRBSTST, TXBUFSTATUS and TXINHIBIT optional ports.
6. Select TXPREEMPHASIS, TXDIFFCTRL, RXEQMIX, RXCDRRESET and RXPOLARITY ports.
7. Keep all other settings as default. Generate the core.
8. Replace the <component name>_tile.v[hd] file in the example_design/gtx directory with the newly created <component name>_tile.v[hd] file.
9. Add TX and RX power down ports in replaced <component name>_tile.v[hd] file in the example_design/gtx/ directory.
10. Resynthesize and re-implement.