LogiCORE IP Aurora 8B/10B v6.2

User Guide

UG766 March 1, 2011

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/21/10	1.0	First release of the core with AXI interface support. The previous release of this document was UG353.
03/01/11	1.1	ISE 13.1 software release with v6.2 core. Removed references to Virtex-5 devices, including Appendix. Changed <component_name> to <component name> throughout. Updated signal names in waveforms to reflect AXI nomenclature. Updated GUI screen captures. Moved Supported Tools and System Requirements into Chapter 1 and removed Chapter 2, Installing and Licensing the Core, because a license is no longer required. Added `<component name>.xise` to Table 10-1, `<component name>.v[hd]` to Table 10-4, and `implement_planahead.tcl` to Table 10-9, `simulate_isim.bat`, `simulate_isim.sh`, and `wave_isim.tcl` to Table 10-11. In Table 10-11, changed the name of `mti_wave.do` to `wave_mti.do`. Added note on page 95 to Appendix C. Added step 9 to Case 1: Virtex-6 FPGA GTX Wrapper. Added step 11 to Case 2: Spartan-6 FPGA GTP Wrapper.

Table of Contents

Appendix D: Aurora AXI4-Stream Migration Guide

Schedule of Figures

Schedule of Tables

About This Guide

The LogiCORE™ IP Aurora 8B/10B core supports the AMBA® protocol AXI4-Stream user interface. The *LogiCORE IP Aurora 8B/10B v6.2 User Guide* provides information for generating a LogiCORE IP Aurora 8B/10B core using Virtex®-6 FPGA GTX transceivers and Spartan®-6 FPGA GTP transceivers.

The core implements the Aurora 8B/10 protocol using the high-speed serial transceivers on the Virtex-6 LXT, SXT, CXT, HXT, and lower-power family and the Spartan-6 LXT family.

This user guide describes the function and operation of the LogiCORE IP Aurora 8B/10B v6.2 core and provides information about designing, customizing, and implementing the core.

Guide Contents

This guide contains the following chapters and appendices:

- Chapter 1, Introduction describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, Customizing the Aurora 8B/10B Core provides port descriptions for the user interface.
- Chapter 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, Status, Control, and the GTP/GTX Block Interface provides details on using the Aurora 8B/10B core's status and control ports to initialize and monitor the Aurora 8B/10B channel.
- Chapter 6, Clock Interface and Clocking describes how to connect FPGA clocking resources.
- Chapter 7, Clock Compensation covers Aurora 8B/10B clock compensation, and explains how to customize it for a given system.
- Chapter 8, Quick Start Example Design provides an overview of the Aurora 8B/10B protocol and core, and gives a step-by-step tutorial on how to generate Aurora 8B/10B designs with the CORE Generator™ software.
- Chapter 9, Example Design Overview defines the main components of the example design.
- Chapter 10, Project Directory Structure provides detailed information about the example design, including a description of files and the directory structure generated by the Xilinx® CORE Generator tool, the purpose and contents of the provided

scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

- Appendix A, Handling Timing Errors Due To Far Apart Transceiver Selection
- Appendix B, Performance and Core Latency
- Appendix C, Generating a Wrapper File from Its Respective GTP/GTX Transceiver Wizard
- Appendix D, Aurora AXI4-Stream Migration Guide explains about migration of legacy (LocalLink based) Aurora cores to the AXI4-Stream based Aurora core.

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 that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported

Convention	Meaning or Use	Example
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **bus[7:0]**, they are required.	**ngdbuild** [*option_name*] *design_name*
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}**
Angle brackets < >	User-defined variable or in code samples	<directory name>

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis . . .	Repetitive material that has been omitted	**allow block** *block_name loc1 loc2 ... locn;*
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active Low	**usr_teof_n** is active low.

Online Document

The following 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 Guide Contents for details. Refer to Title Formats in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Chapter 1

Introduction

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

About the Core

The Aurora 8B/10B core is a CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see www.xilinx.com/aurora.

A license is not required to use the Aurora 8B/10B core.

Supported Tools and System Requirements

Operating Systems

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

Tools

- ISE® software 13.1
- Mentor Graphics ModelSim 6.6d

Recommended Design Experience

Although the Aurora core is a fully verified solution, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application. For best results, previous experience building high-performance, pipelined FPGA designs using Xilinx implementation software and user constraints files (`.ucf`) is recommended.

Read Chapter 5, Status, Control, and the GTP/GTX Block Interface, carefully.

Consult the PCB design requirements information in:

- *Virtex-6 FPGA GTX Transceivers User Guide*
- *Spartan-6 FPGA GTP Transceivers User Guide*

Contact your local Xilinx representative for a closer review and estimation for your specific requirements.

References

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

- SP002, *Aurora 8B/10B Protocol Specification*
- *AMBA AXI4-Stream Protocol Specification*
- UG366, *Virtex-6 FPGA GTX Transceivers User Guide*
- UG386, *Spartan-6 FPGA GTP Transceivers User Guide*
- ISE® software documentation: www.xilinx.com/ise

Additional Core Resources

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

- DS797, *LogiCORE IP Aurora 8B/10B v6.2 Data Sheet*
- UG058: *Aurora 8B/10B Bus Functional Model User Guide* (Contact: auroramkt@xilinx.com)
- Aurora 8B/10B Release Notes
- Aurora Solution list – AR#21263

Technical Support

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

Xilinx provides technical support for use of this product as described in the *LogiCORE IP Aurora 8B/10B v6.2 User Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow the guidelines.

Feedback

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

Core

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

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

Documentation

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

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

Chapter 2

Customizing the Aurora 8B/10B Core

Introduction

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

Using the IP Customizer

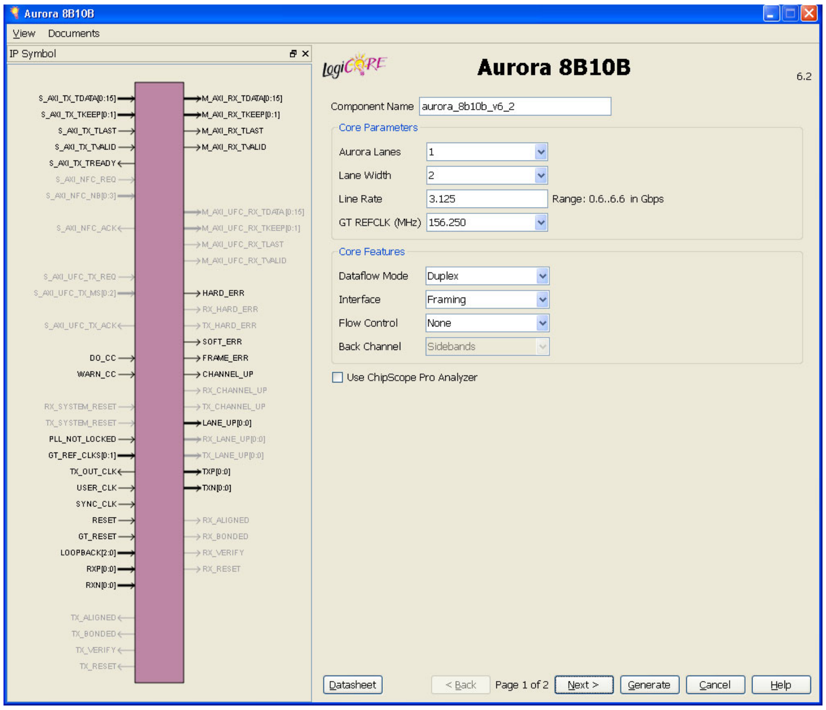
The Aurora 8B/10B IP Customizer is presented when the user selects the Aurora 8B/10B core in the CORE Generator software. For help starting and using the CORE Generator software, see the *CORE Generator Help* in the ISE® software documentation. Each numbered item in Figure 2-1, page 20 corresponds to its respective section that describes the purpose of the feature.

IP Customizer

Figure 2-1, page 20 shows the customizer. The left side displays a representative block diagram of the Aurora 8B/10B core as currently configured. The right side consists of user-configurable parameters.

The second pages of the GUI are shown in:

- Figure 2-2, page 20 for Virtex®-6 FPGA GTX transceivers
- Figure 2-3, page 21 for Spartan®-6 FPGA GTP transceivers



Aurora 8B10B 6.2

Component Name: aurora_8b10b_v6_2

Core Parameters

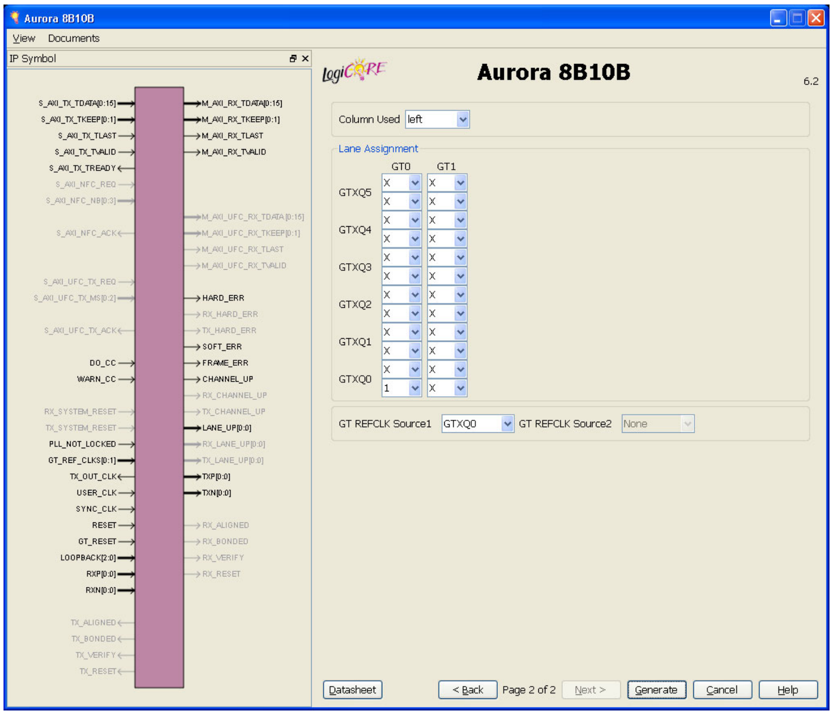
Aurora Lanes	1
Lane Width	2
Line Rate	3.125 Range: 0.6..6.6 in Gbps
GT REFCLK (MHz)	156.250

Core Features

Dataflow Mode	Duplex
Interface	Framing
Flow Control	None
Back Channel	Sidebands

☐ Use ChipScope Pro Analyzer

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



Aurora 8B10B 6.2

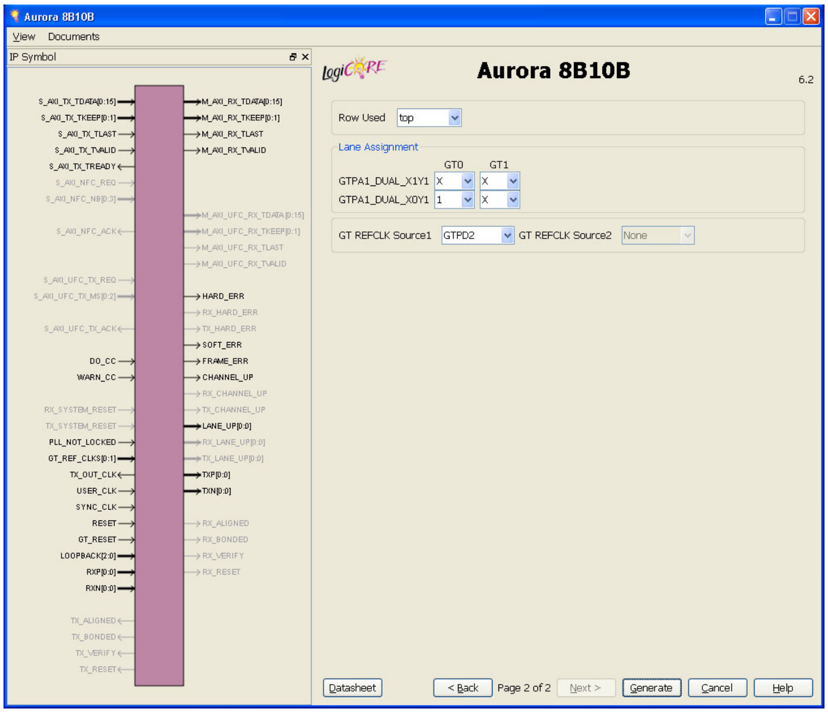
Column Used: left

Lane Assignment

	GT0	GT1
GTXQ5	X	X
	X	X
GTXQ4	X	X
	X	X
GTXQ3	X	X
	X	X
GTXQ2	X	X
	X	X
GTXQ1	X	X
	X	X
GTXQ0	X	X
	1	X

GT REFCLK Source1: GTXQ0 GT REFCLK Source2: None

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



Row Used: top

Lane Assignment

	GT0	GT1
GTPA1_DUAL_X1Y1	X	X
GTPA1_DUAL_X0Y1	1	X

GT REFCLK Source1: GTPD2 GT REFCLK Source2: None

Figure 2-3: **Second GUI Page for Spartan-6 FPGA GTP Transceivers**

Component Name

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

Default: aurora_8b10b_v6_2

Lane Assignment

Refer to the diagram in the information area in Figure 2-2, page 20 and Figure 2-3, page 21. Each row represents a GTPA1_DUAL tile in Spartan-6 FPGAs or two columns represent a GTX Quad in Virtex-6 FPGAs. Each active box represents an available GTP/GTX transceiver.

Aurora Lanes

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

Default: 1

Lane Width

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

Default: 2

Interface

Select the type of datapath interface used for the core. Select *Framing* to use an AXI4-Stream interface that allows encapsulation of data frames of any length. Select *Streaming* to use a simple word-based interface with a data valid signal to stream data through the Aurora 8B/10B channel. See Chapter 3, User Interface for more information.

Default: Framing

Dataflow Mode

Select the options for direction of the channel the Aurora 8B/10B core supports. Simplex Aurora 8B/10B cores have a single, unidirectional serial port that connects to a complementary simplex Aurora 8B/10B core. Available options are *RX-only Simplex, TX-only Simplex, RX/TX Simplex, and Duplex*. RX/TX Simplex creates two cores, one RX and one TX, that share a single GTP/GTX transceiver. It is provided for simulation purposes only and is not recommended to be used for FPGA implementations. See Chapter 5, Status, Control, and the GTP/GTX Block Interface for more information.

Default: Duplex

Back Channel

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

- Sidebands
- Timer

Default: Sidebands

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

Flow Control

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

Available options are listed below (See Chapter 4, Flow Control for more information):

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

Default: None

Line Rate

Enter a floating-point value in gigabits per second within the valid range. This determines the unencoded bit rate at which data is transferred over the serial link. The aggregate data rate of the core is (0.8 × line rate) × Aurora 8B/10B lanes.

Default: 3.125 Gbps

GT REFCLK (MHz)

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

Default: 156.250 MHz

GT REFCLK Source1 and GT REFCLK Source2

Select reference clock sources for the GTPA1_DUAL tiles or GTX Quad from the drop-down list in this section.

- Default: GT REFCLK Source1 - GTPD0; GT REFCLK Source2 - None for Spartan-6 FPGA GTP transceivers
- Default: GT REFCLK Source1 - GTXQ0; GT REFCLK Source2 - None for Virtex-6 FPGA GTX transceivers
- GTPD0 and GTXQ0 change based on the selected device and package.

Column Used

Select the appropriate column of transceivers used from the drop-down list. The column used is enabled only for Virtex-6 HXT devices and is disabled for all other devices.

Default: left

Row Used

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

Default: top

Use ChipScope Pro Analyzer

Select to add ChipScope™ Pro cores to the Aurora 8B/10B core (see Using ChipScope Pro Cores with the Aurora 8B/10B Core in Chapter 8). This option provides users a debugging interface that shows the core status signals in the ChipScope Pro analyzer tool.

Default: Unchecked

Generate

Click Generate to generate the core. The modules for the Aurora 8B/10B core are written to the CORE Generator software project directory using the same name as the top level of the core. See Chapter 10, Project Directory Structure for details about the example_design directory and files.

Using the Build Script

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

```
./implement.sh
```

Designing with the Core

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

General Design Guidelines

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

Use the Example Design as a Starting Point

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

Know the Degree of Difficulty

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

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

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

Keep It Registered

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

Recognize Timing Critical Signals

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

Use Supported Design Flows

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

Make Only Allowed Modifications

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

Chapter 3

User Interface

Introduction

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

The framing user interface complies with the *AXI4-Stream Protocol Specification*. It comprises the signals necessary for transmitting and receiving framed user data. The streaming interface allows users to send data without special frame delimiters. It is simple to operate and uses fewer resources than framing.

Top Level Architecture

Aurora 8B/10B top level (block level) file instantiates Aurora 8B/10B lane module, TX and RX AXI4-Stream modules, global logic module, and wrapper for GTP/GTX transceiver. This top level wrapper file is instantiated in the example design file together with clock, reset circuit and frame generator and checker modules.

Figure 3-1, page 28 shows Aurora 8B/10B top level for a duplex configuration. The top level file is the starting point for a user design.

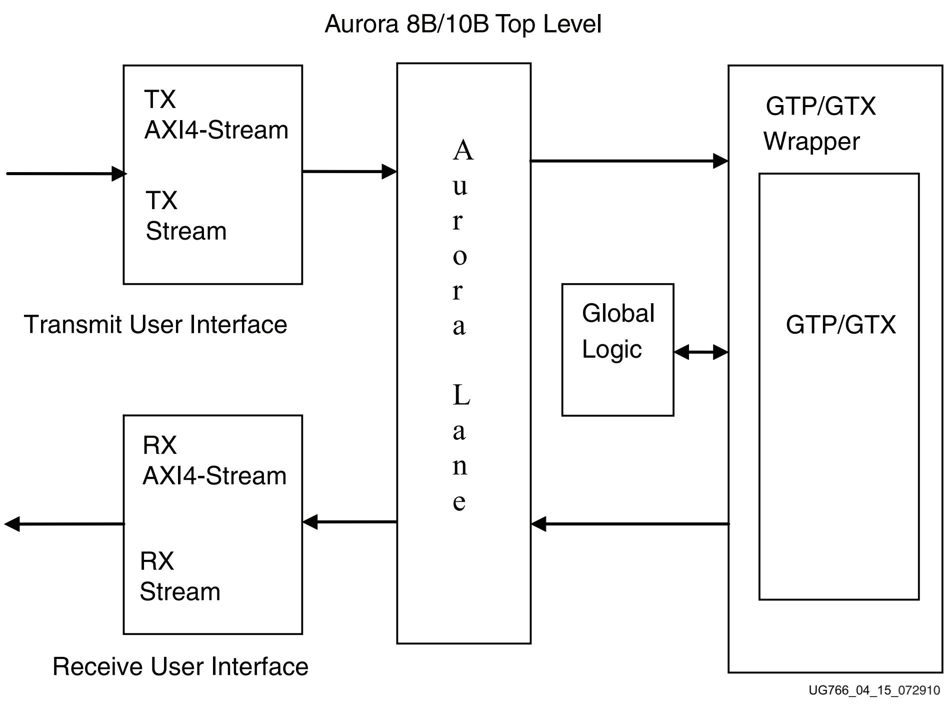


Figure 3-1: **Top-Level Architecture**

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

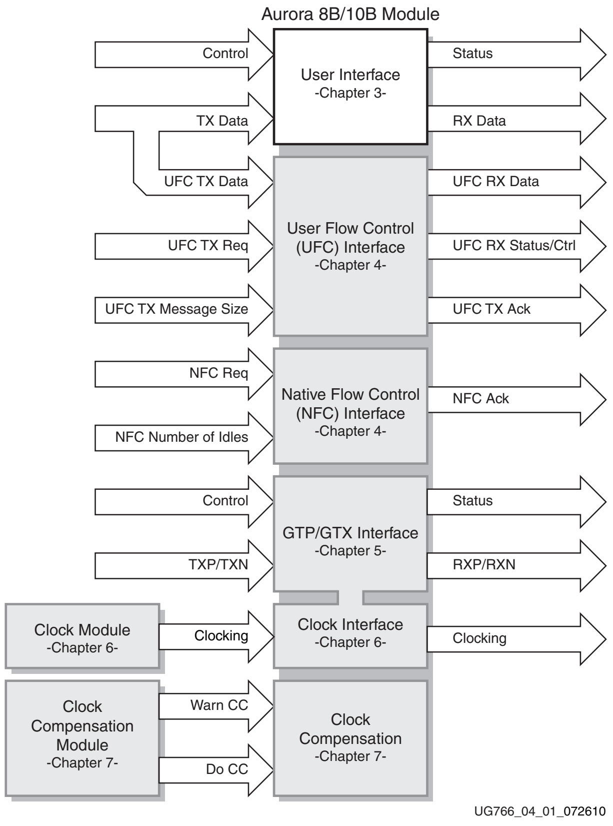


Figure 3-2: **Top-Level User Interface**

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

Framing Interface

Figure 3-3 shows the framing user interface of the Aurora 8B/10B core, with AXI4-Stream-compliant ports for TX and RX data.

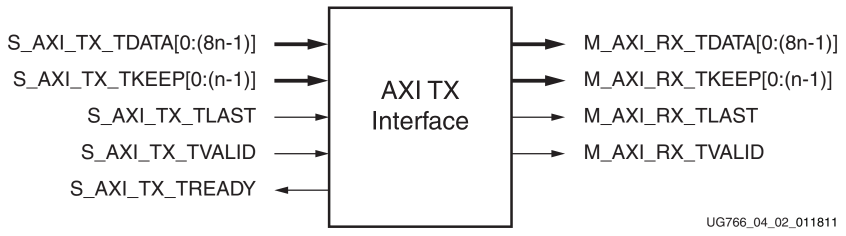


Figure 3-3: **Aurora 8B/10B Core Framing Interface (AXI4-Stream)**

Framing TX Ports

Table 3-1 lists port descriptions for AXI4-Stream TX data ports. These ports are included on full-duplex, simplex TX, and RX/TX Simplex framing cores.

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

Name	Direction	Description
S_AXI_TX_TDATA[0:(8n-1)	Input	Outgoing data (Ascending bit order). • *n* is the number of bytes
S_AXI_TX_TREADY	Output	Asserted (High) during clock edges when signals from the source are accepted (if S_AXI_TX_TVALID is also asserted). Deasserted (Low) on clock edges when signals from the source are ignored.
S_AXI_TX_TLAST	Input	Signals the end of the frame (active-High).
S_AXI_TX_TKEEP[0:(n-1)]	Input	Specifies the number of valid bytes in the last data beat; valid only while S_AXI_TX_TLAST is asserted. S_AXI_TX_TKEEP is the byte qualifier that indicates whether the content of the associated byte of S_AXI_TX_TDATA is valid or not. The Aurora core expects the data to be filled continuously from LSB to MSB. There cannot be invalid bytes interleaved with the valid S_AXI_TX_TDATA bus.
S_AXI_TX_TVALID	Input	Asserted (High) when AXI4-Stream signals from the source are valid. Deasserted (Low) when AXI4-Stream control signals and/or data from the source should be ignored.

Framing RX Ports

Table 3-2 lists port descriptions for Framing RX data ports. These ports are included on full-duplex, simplex RX, and RX/TX Simplex framing cores.

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

Name	Direction	Description
M_AXI_RX_TDATA[0:8(n-1)]]	Output	Incoming data from channel partner (Ascending bit order).
M_AXI_RX_TLAST	Output	Signals the end of the incoming frame (active-High, asserted for a single user clock cycle). Ignored when M_AXI_RX_TVALID is deasserted (Low).
M_AXI_RX_TKEEP[0:(n-1)]	Output	Specifies the number of valid bytes in the last data beat; valid only when M_AXI_RX_TLAST is asserted.
M_AXI_RX_TVALID	Output	Asserted (High) when data and control signals from an Aurora 8B/10B core are valid. Deasserted (Low) when data and/or control signals from an Aurora 8B/10B core should be ignored.

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

- Take data from the user on the S_AXI_TX_TDATA bus
- Encapsulate and stripe the data across lanes in the Aurora 8B/10B channel (S_AXI_TX_TVALID, S_AXI_TX_TLAST)
- Pause data (that is, insert idles) (S_AXI_TX_TVALID)

When the core receives data, it does the following:

- Detects and discards control bytes (idles, clock compensation, SCP, ECP)
- Asserts framing signal (M_AXI_RX_TLAST)
- Recovers data from the lanes
- Assembles data for presentation to the user on the M_AXI_RX_TDATA bus

AXI4-Stream Bit Ordering

Aurora 8B/10B cores use ascending ordering. They transmit and receive the most significant bit of the most significant byte first. Figure 3-4 shows the organization of an *n*-byte example of the AXI4-Stream data interfaces of an Aurora 8B/10B core.

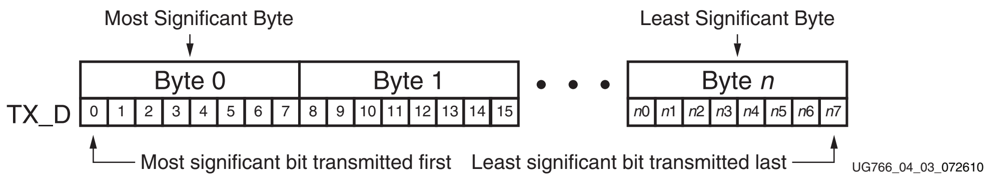


Figure 3-4: **AXI4-Stream Interface Bit Ordering**

Transmitting Data

AXI4-Stream is a synchronous interface. The Aurora 8B/10B core samples the data on the interface only on the positive edge of USER_CLK, and only on the cycles when both S_AXI_TX_TREADY and S_AXI_TX_TVALID are asserted (High).

When AXI4-Stream signals are sampled, they are only considered valid if S_AXI_TX_TVALID is asserted. The user application can deassert S_AXI_TX_TVALID on any clock cycle; this causes the Aurora 8B/10B core to ignore the AXI4-Stream input for that cycle. If this occurs in the middle of a frame, idle symbols are sent through the Aurora 8B/10B channel, which eventually result in a idle cycles during the frame when it is received at the RX user interface.

AXI4-Stream data is only valid when it is framed. Data outside of a frame is ignored. To start a frame, assert S_AXI_TX_TVALID while the first word of data is on the S_AXI_TX_TDATA port. To end a frame, assert S_AXI_TX_TLAST while the last word (or partial word) of data is on the S_AXI_TX_TDATA port.

Note: In the case of frames that are a single word long or less, S_AXI_TX_TVALID and S_AXI_TX_TLAST are asserted simultaneously.

Data Remainder

AXI4-Stream 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 S_AXI_TX_TKEEP bus is used to indicate the number of valid bytes in the final word of the frame. The bus is only used when S_AXI_TX_TLAST is asserted.

Aurora 8B/10B Frames

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

Length

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

Table 3-3: **Typical Channel Frame**

$/SCP/_1$	$/SCP/_2$	Data Byte 0	Data Byte 1	Data Byte 2	...	Data Byte n -1	Data Byte n	$/ECP/_1$	$/ECP/_2$

Example A: Simple Data Transfer

Figure 3-5 shows an example of a simple data transfer on a AXI4-Stream interface that is *n*-bytes wide. In this case, the amount of data being sent is 3*n* bytes and so requires three data beats. S_AXI_TX_TREADY is asserted, indicating that the AXI4-Stream interface is ready to transmit data. When the Aurora 8B/10B core is not sending data, it sends idle sequences.

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

To end the data transfer, the user asserts S_AXI_TX_TLAST, S_AXI_TX_TVALID, the last data bytes, and the appropriate value on the S_AXI_TX_TKEEP bus. In this example, S_AXI_TX_TKEEP is set to N (in the waveform for demonstration) to indicate that all bytes are valid in the last data beat. One clock cycle after S_AXI_TX_TLAST is asserted, the AXI4-Stream interface deasserts S_AXI_TX_TREADY and uses the gap in the data flow to send the final offset data bytes and the /ECP/ ordered set, indicating the end of the frame. S_AXI_TX_TREADY is reasserted on the next cycle so that more data transfers can continue. As long as there is no new data, the Aurora 8B/10B core sends idles.

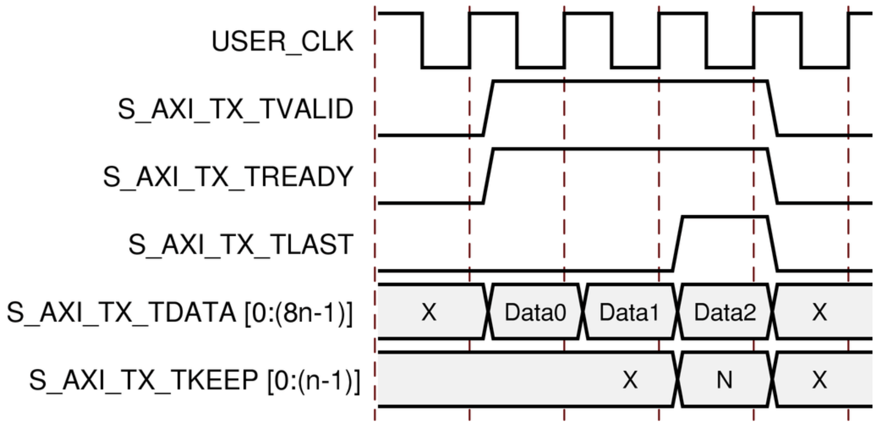


Figure 3-5: **Simple Data Transfer**

Example B: Data Transfer with Pad

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

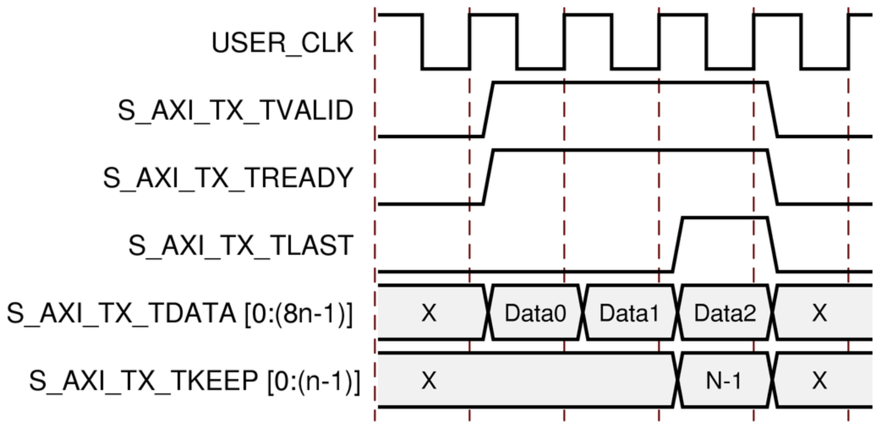


Figure 3-6: **Data Transfer with Pad**

Example C: Data Transfer with Pause

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

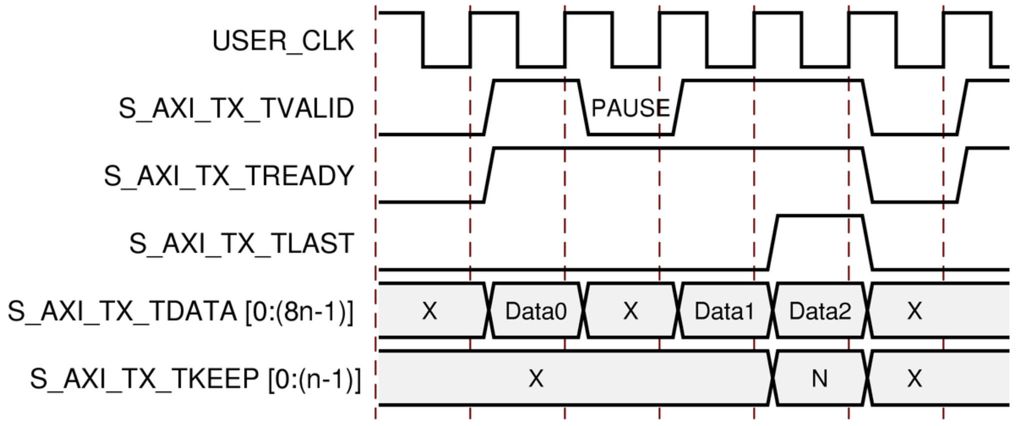


Figure 3-7: **Data Transfer with Pause**

Example D: Data Transfer with Clock Compensation

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

Figure 3-8 shows how the Aurora 8B/10B core pauses data transmission during the clock compensation[1] sequence.

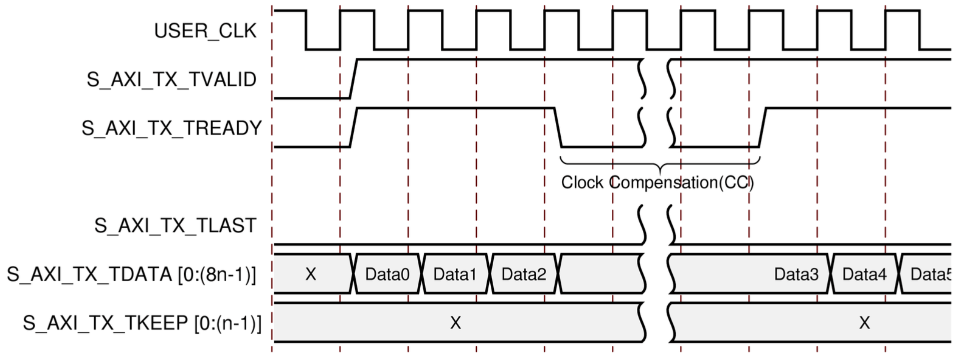


Figure 3-8: **Data Transfer Paused by Clock Compensation**

Receiving Data

When the Aurora 8B/10B core receives an Aurora 8B/10B frame, it presents it to the user through the RX AXI4-Stream interface after discarding the framing characters, idles, and clock compensation sequences.

The RX submodules has no built in elastic buffer for user data. As a result, there is no M_AXI_RX_TREADY signal on the RX AXI4-Stream interface. The only way for the user application to control the flow of data from an Aurora 8B/10B channel is to use one of the core's optional flow control features. In most cases, a FIFO should be added to the RX data path to ensure no data is lost while flow control messages are in transit.

The Aurora 8B/10B core asserts the M_AXI_RX_TVALID signal when the signals on its RX AXI4-Stream interface are valid. Applications should ignore any values on the RX AXI4-Stream ports sampled while M_AXI_RX_TVALID is deasserted (Low).

M_AXI_RX_TVALID is asserted concurrently with the first word of each frame from the Aurora 8B/10B core. M_AXI_RX_TLAST is asserted concurrently with the last word or partial word of each frame. The M_AXI_RX_TKEEP port indicates the number of valid bytes in the final word of each frame.M_AXI_RX_TKEEP is only valid when M_AXI_RX_TLAST is asserted.

The Aurora 8B/10B core can deassert M_AXI_RX_TVALID anytime, even during a frame. The timing of the M_AXI_RX_TVALID deassertions is independent of the way the data was transmitted. The core can occasionally deassert M_AXI_RX_TVALID even if the frame was originally transmitted without pauses. These pauses are a result of the framing character stripping and left alignment process, as the core attempts to process each frame with as little latency as possible.

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

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

Example A: Data Reception with Pause

Figure 3-9 shows an example of 3*n* bytes of received data interrupted by a pause. Data is presented on the M_AXI_RX_TDATA bus. When the first *n* bytes are placed on the bus, M_AXI_RX_TVALID is asserted to indicate that data is ready for the user. On the clock cycle following the first data beat, the core deasserts M_AXI_RX_TVALID, indicating to the user that there is a pause in the data flow.

After the pause, the core asserts M_AXI_RX_TVALID and continues to assemble the remaining data on the M_AXI_RX_TDATA bus. At the end of the frame, the core asserts M_AXI_RX_TLAST. The core also computes the value of M_AXI_RX_TKEEP bus and presents it to the user based on the total number of valid bytes in the final word of the frame.

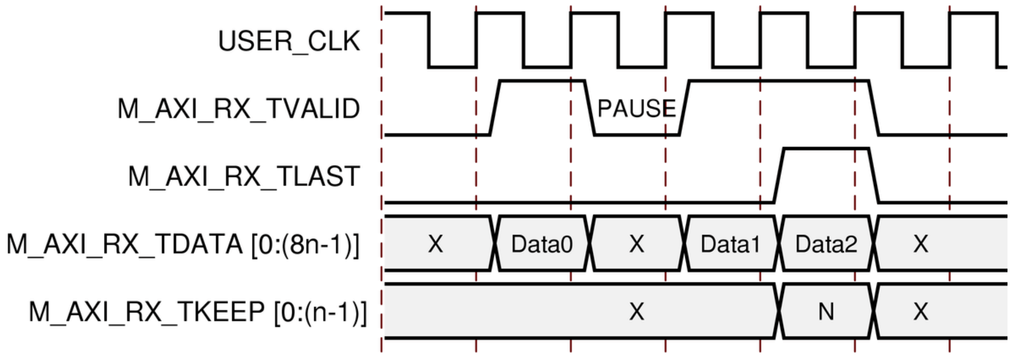


Figure 3-9: **Data Reception with Pause**

Framing Efficiency

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

- Size of the frame
- Width of the data path

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

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

Like many parallel interfaces, AXI4-Stream processes data from only one frame at a time. The core must drop data when it arrives on the GTP/GTX transceiver interface at the same time as data from a previous cycle.

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

Use the formula shown in Figure 3-10 to calculate the efficiency for a design of any number of lanes, any width of interface, and frames of any number of bytes.

Note: This formula includes the overhead for clock compensation.

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

Where:

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

UG766_04_11_072610

Figure 3-10: **Formula for Calculating Overhead**

Example

Table 3-4 is an example calculated from the formula given in Figure 3-10. It shows the efficiency for an 8-byte, 4-lane channel and illustrates that the efficiency increases as the length of channel frames increases.

Table 3-4: **Efficiency Example**

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

Table 3-5 shows the overhead in an 8-byte, 4-lane channel when transmitting 256 bytes of frame data across the four lanes. The resulting data unit is 264 bytes long due to start and end characters, and due to the idles necessary to fill out the lanes. This amounts to 3.03% of overhead in the transmitter. In addition, a 12-byte clock compensation sequence occurs on each lane every 10,000 bytes, which adds a small amount more to the overhead. The receiver can handle a slightly more efficient data stream because it does not require any idle pattern.

Table 3-5: **Typical Overhead for Transmitting 256 Data Bytes**

Lane	Clock	Function	Character or Data Byte	
			Byte 1	Byte 2
0	1	Start of channel frame	$/SCP/_1$	$/SCP/_2$
1	1	Channel frame data	D0	D1
2	1	Channel frame data	D2	D3
3	1	Channel frame data	D4	D5
		. . .		
0	33	Channel frame data	D254	D255
1	33	Transmit idles	/I/	/I/
2	33	Transmit idles	/I/	/I/
3	33	End of channel frame	$/ECP/_1$	$/ECP/_2$

Table 3-6 shows the overhead that occurs with each value of S_AXI_TX_TKEEP.

Table 3-6: **S_AXI_TX_TKEEP Value and Corresponding Bytes of Overhead**

S_AXI_TX_TKEEP Bus Value (in Binary)	SCP	Pad	ECP	Idles	Total
1000_0000	2	1	2	6	11
1100_0000		0			10
1110_0000		1		4	9
1111_0000		0			8
1111_1000		1		2	7
1111_1100		0			6
1111_1110		1		0	5
1111_1111		0			4

Streaming Interface

Figure 3-11 shows an example of an Aurora 8B/10B core configured with a streaming user interface.

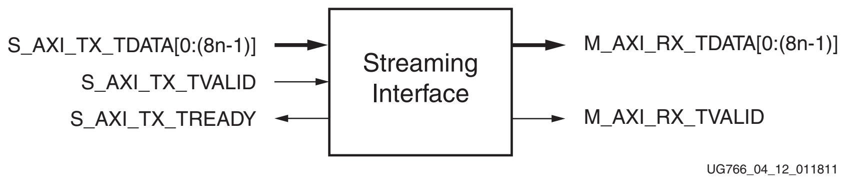


Figure 3-11: **Aurora 8B/10B Core Streaming User Interface**

Streaming TX Ports

Table 3-7 lists the streaming TX data ports. These ports are included on full-duplex, simplex TX, and RX/TX Simplex framing cores.

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

Name	Direction	Description
S_AXI_TX_TDATA[0:(8n-1)]]	Input	Outgoing data (ascending bit order).
S_AXI_TX_TREADY	Output	Asserted (High) during clock edges when signals from the source are accepted (if S_AXI_TX_TVALID is also asserted). Deasserted (Low) on clock edges when signals from the source are ignored.
S_AXI_TX_TVALID	Input	Asserted (High) when AXI4-Stream signals from the source are valid. Deasserted (Low) when AXI4-Stream control signals and/or data from the source should be ignored.

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
M_AXI_RX_TDATA[0:(8n-1)]	Output	Incoming data from channel partner (Ascending bit order).
M_AXI_RX_TVALID	Output	Asserted (High) when data and control signals from an Aurora 8B/10B core are valid. Deasserted (Low) when data from an Aurora 8B/10B core should be ignored.

Transmitting and Receiving Data

The streaming interface allows the Aurora 8B/10B channel to be used as a pipe. Words written into the TX side of the channel are delivered, in order after some latency, to the RX side. After initialization, the channel is always available for writing, except when the DO_CC signal is asserted to send clock compensation sequences. Applications transmit data through the S_AXI_TX_TDATA port, and use the S_AXI_TX_TVALID port to indicate when the data is valid (asserted High). The Aurora 8B/10B core deasserts S_AXI_TX_TREADY (Low) when the channel is not ready to receive data. Otherwise, S_AXI_TX_TREADY remains asserted.

When S_AXI_TX_TVALID is deasserted, gaps are created between words. These gaps are preserved, except when clock compensation sequences are being transmitted. Clock compensation sequences are replicated or deleted by the GTP/GTX transceiver to make up for frequency differences between the two sides of the Aurora 8B/10B channel. As a result, gaps created when DO_CC is asserted can shrink and grow. For details on the DO_CC signal, see Chapter 7, Clock Compensation.

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

Figure 3-12, page 41 shows a typical example of streaming data. The example begins with neither of the ready signals asserted, indicating that both the user logic and the Aurora 8B/10B core are not ready to transfer data. During the next clock cycle, the Aurora 8B/10B core indicates that it is ready to transfer data by asserting S_AXI_TX_TREADY. One cycle later, the user logic indicates that it is ready to transfer data by asserting the S_AXI_TX_TDATA bus and the S_AXI_TX_TVALID signal. Because both ready signals are now asserted, data D0 is transferred from the user logic to the Aurora 8B/10B core. Data D1 is transferred on the following clock cycle. In this example, the Aurora 8B/10B core deasserts its ready signal, S_AXI_TX_TREADY, and no data is transferred until the next clock cycle when, once again, the S_AXI_TX_TREADY signal is asserted. Then the user deasserts S_AXI_TX_TVALID on the next clock cycle, and no data is transferred until both ready signals are asserted.

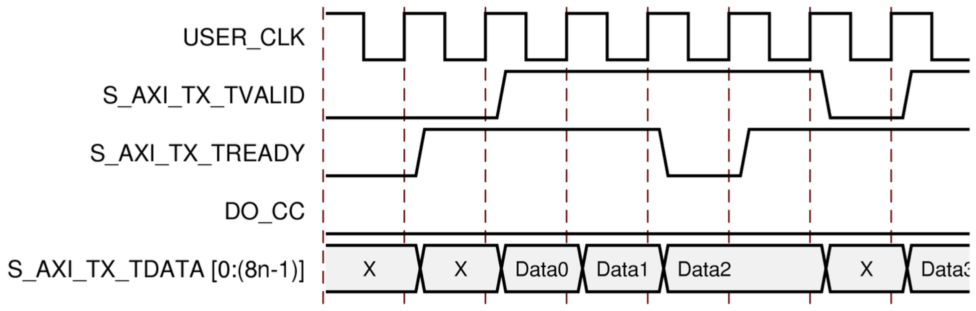


Figure 3-12: **Typical Streaming Data Transfer**

Figure 3-13 shows the receiving end of the data transfer that is shown in Figure 3-12.

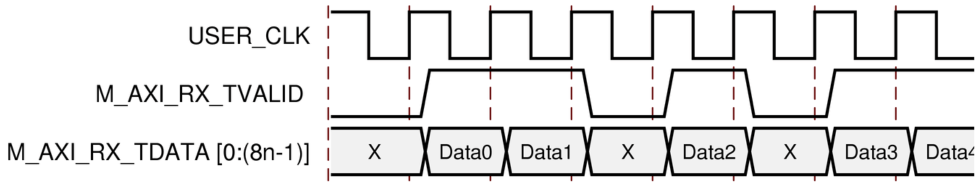


Figure 3-13: **Typical Data Reception**

Chapter 4

Flow Control

Introduction

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

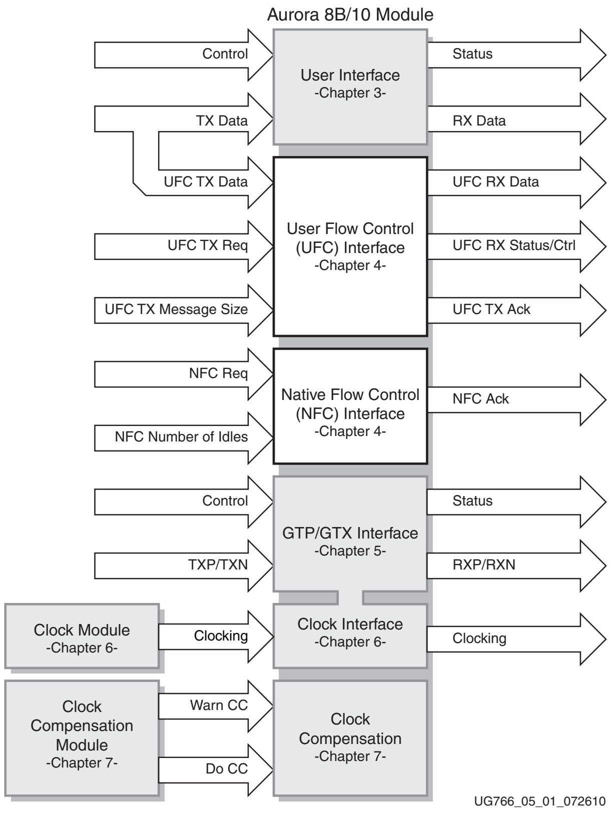


Figure 4-1: **Top-Level Flow Control**

Native Flow Control

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

Table 4-1: **NFC Codes**

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

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

Table 4-2: **NFC I/O Ports**

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

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

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

Example A: Transmitting an NFC Message

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

Note: S_AXI_TX_TREADY is deasserted for one cycle (assumes that *n* is at least 2) to create the gap in the data flow in which the NFC message is placed.

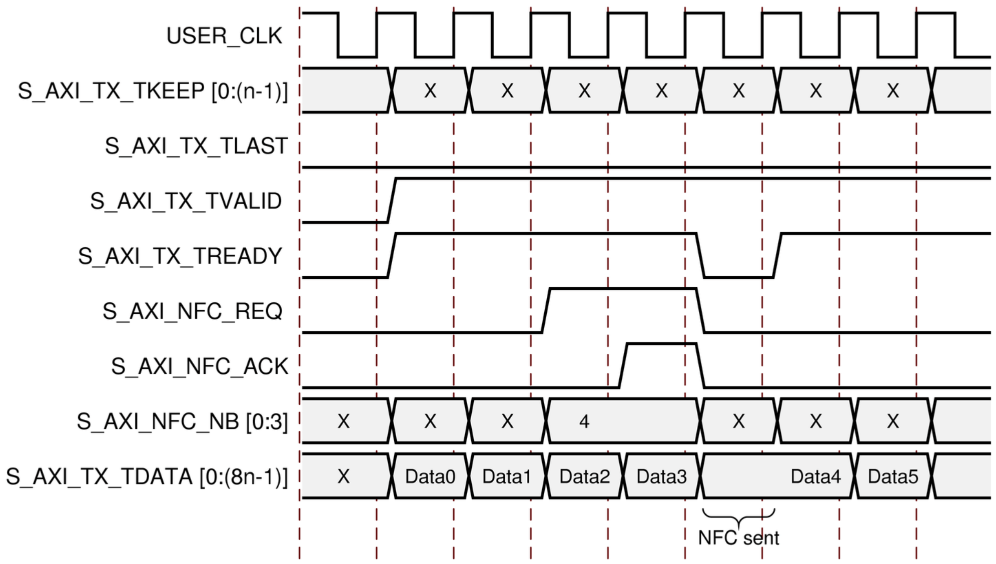


Figure 4-2: **Transmitting an NFC Message**

Example B: Receiving a Message with NFC Idles Inserted

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

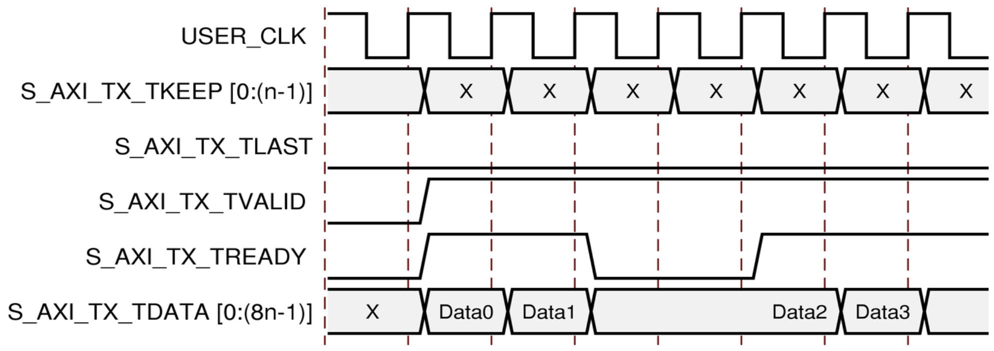


Figure 4-3: **Transmitting a Message with NFC Idles Inserted**

User Flow Control

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

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

Table 4-3: **UFC I/O Ports**

Name	Direction	Description
S_AXI_UFC_TX_REQ	Input	Asserted to request a UFC message be sent to the channel partner (active-High). Must be held until S_AXI_UFC_TX_ACK is asserted. Do not assert this signal unless the entire UFC message is ready to be sent; a UFC message cannot be interrupted once it has started.
S_AXI_UFC_TX_MS[0:2]	Input	Specifies the size of the UFC message that is sent. The SIZE encoding is a value between 0 and 7. See Table 4-4, page 48.
S_AXI_UFC_TX_ACK	Output	Asserted when an Aurora 8B/10B core is ready to read the contents of the UFC message (active-High). On the cycle after the S_AXI_UFC_TX_ACK signal is asserted, data on the S_AXI_TX_TDATA port is treated as UFC data. S_AXI_TX_TDATA data continues to be used to fill the UFC message until enough cycles have passed to send the complete message. Unused bytes from a UFC cycle are discarded.
M_AXI_UFC_RX_TDATA[0:(8n-1)]	Output	Incoming UFC message data from the channel partner (n = 16 bytes maximum).
M_AXI_UFC_RX_TVALID	Output	Asserted when the values on the M_AXI_UFC_RX ports are valid. When this signal is not asserted, all values on the M_AXI_UFC_RX ports should be ignored (active-High).
M_AXI_UFC_RX_TLAST	Output	Signals the end of the incoming UFC message (active-High).
M_AXI_UFC_RX_TKEEP[0:(n-1)]	Output	Specifies the number of valid bytes of data presented on the M_AXI_UFC_RX_TDATA port on the last word of a UFC message. Valid only when M_AXI_UFC_RX_TLAST is asserted (*n* = 16 bytes maximum).

Transmitting UFC Messages

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

Table 4-4: **SIZE Encoding**

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

To send a UFC message, the user application asserts S_AXI_UFC_TX_REQ while driving the S_AXI_UFC_TX_MS port with the desired SIZE code. S_AXI_UFC_TX_REQ must be held until the Aurora 8B/10B core asserts the S_AXI_UFC_TX_ACK signal, indicating that the core is ready to send the UFC message. The data for the UFC message must be placed on the S_AXI_TX_TDATA port of the data interface, starting on the first cycle after S_AXI_UFC_TX_ACK is asserted. The core deasserts S_AXI_TX_TREADY while the S_AXI_TX_TDATA port is being used for UFC data.

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

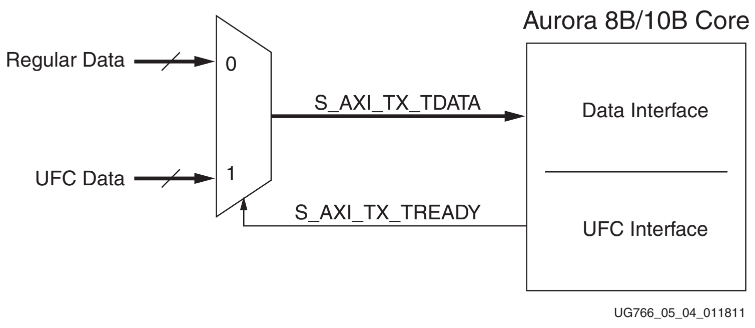


Figure 4-4: **Data Switching Circuit**

Table 4-5, page 49 shows the number of cycles required to transmit UFC messages of different sizes based on the width of the AXI4-Stream data interface. UFC messages should never be started until all message data is available. Unlike regular data, UFC messages cannot be interrupted after S_AXI_UFC_TX_ACK has been asserted.

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

UFC Message	S_AXI_UFC_TX_MS Value	AXI Interface Width	Number of Data Beats	AXI Interface Width	Number of Data Beats
2 Bytes	0	2 Bytes	1	10 Bytes	1
4 Bytes	1		2		
6 Bytes	2		3		
8 Bytes	3		4		
10 Bytes	4		5		
12 Bytes	5		6		2
14 Bytes	6		7		
16 Bytes	7		8		
2 Bytes	0	4 Bytes	1	12 Bytes	1
4 Bytes	1				
6 Bytes	2		2		
8 Bytes	3				
10 Bytes	4		3		
12 Bytes	5				
14 Bytes	6		4		2
16 Bytes	7				
2 Bytes	0	6 Bytes	1	14 Bytes	1
4 Bytes	1				
6 Bytes	2				
8 Bytes	3		2		
10 Bytes	4				
12 Bytes	5				
14 Bytes	6		3		
16 Bytes	7				2
2 Bytes	0	8 Bytes	1	16 Bytes or more	1
4 Bytes	1				
6 Bytes	2				
8 Bytes	3				
10 Bytes	4		2		
12 Bytes	5				
14 Bytes	6				
16 Bytes	7				

Example A: Transmitting a Single-Cycle UFC Message

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

Note: S_AXI_TX_TREADY is deasserted for two cycles. Aurora 8B/10B cores use this gap in the data flow to transmit the UFC header and message data.

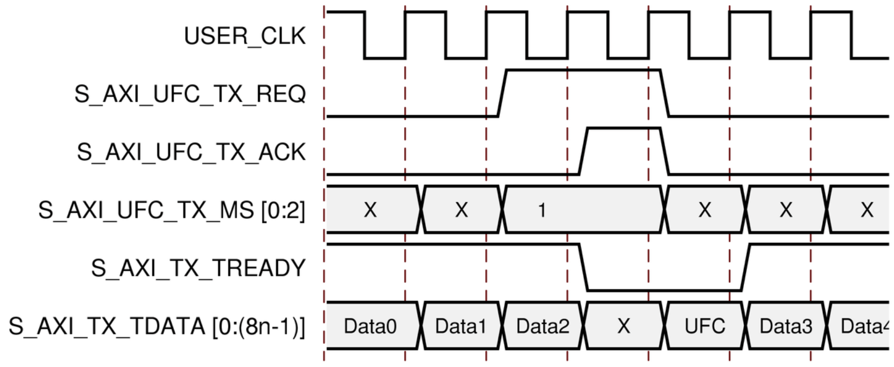


Figure 4-5: **Transmitting a Single-Cycle UFC Message**

Example B: Transmitting a Multicycle UFC Message

The procedure for transmitting a two-cycle UCF message is shown in Figure 4-6. In this case the user application is sending a 4-byte message using a 2-byte interface. S_AXI_TX_TREADY is asserted for three cycles; one cycle for the UFC header which is sent during the S_AXI_UFC_TX_ACK cycle, and two cycles for UFC data.

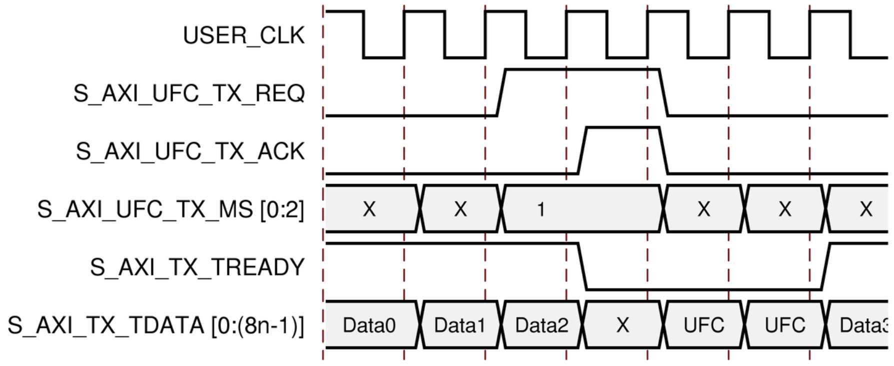


Figure 4-6: **Transmitting a Multicycle UFC Message**

Receiving User Flow Control Messages

When the Aurora 8B/10B core receives a UFC message, it passes the data from the message to the user application through a dedicated UFC AXI4-Stream interface. The data is presented on the M_AXI_UFC_RX_TDATA port; M_AXI_UFC_RX_TVALID indicates the start of the message data and M_AXI_UFC_RX_TLAST indicates the end. M_AXI_UFC_RX_TKEEP is used to show the number of valid bytes on M_AXI_UFC_RX_TDATA during the last cycle of the message (for example, while M_AXI_UFC_RX_TLAST is asserted). Signals on the M_AXI_UFC_RX AXI4-Stream interface are only valid when M_AXI_UFC_RX_TVALID is asserted.

Example C: Receiving a Single-Cycle UFC Message

Figure 4-7 shows an Aurora 8B/10B core with a 4-byte data interface receiving a 4-byte UFC message. The core presents this data to the user application by asserting M_AXI_UFC_RX_TVALID and M_AXI_UFC_RX_TLAST to indicate a single cycle frame. M_AXI_UFC_RX_TKEEP is set to `4'hF`, indicating only the four most significant bytes of the interface are valid.

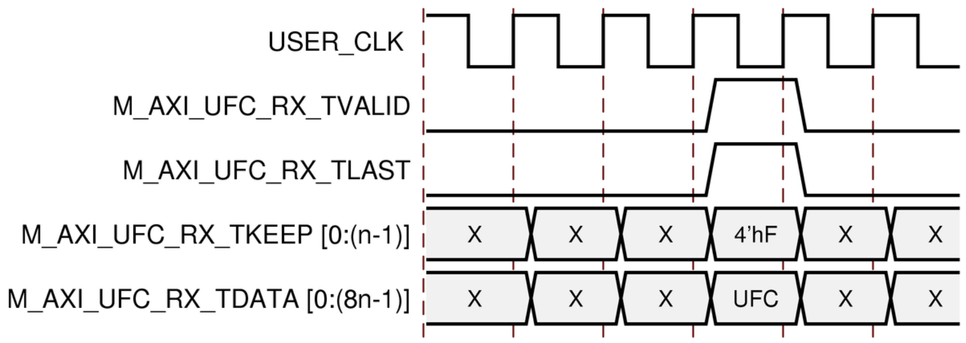


Figure 4-7: **Receiving a Single-Cycle UFC Message**

Example D: Receiving a Multicycle UFC Message

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

Note: The resulting frame is two cycles long, with M_AXI_UFC_RX_TKEEP set to `4'hF` on the second cycle indicating that all eight bytes of the data are valid.

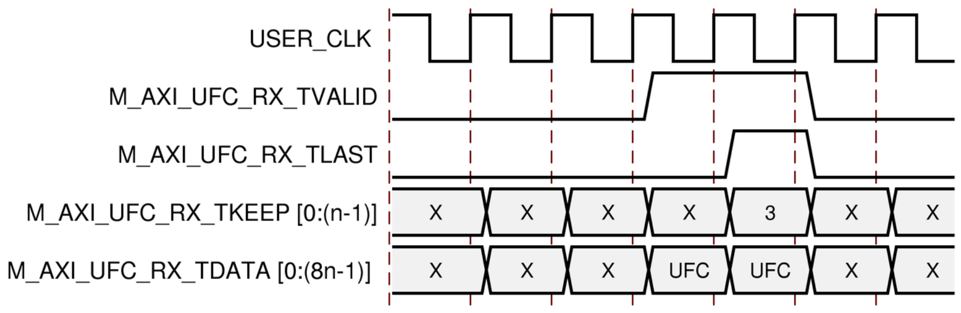


Figure 4-8: **Receiving a Multicycle UFC Message**

Status, Control, and the GTP/GTX Block Interface

Introduction

The status and control ports of the Aurora 8B/10B core allow user applications to monitor the Aurora 8B/10B channel and use built-in features of the GTP/GTX transceivers. Aurora 8B/10B cores can be configured as full-duplex or simplex modules. Full-duplex modules provide high-speed TX and RX links. Simplex modules provide a link in only one direction and are initialized using sideband ports or with a built-in timer. This chapter provides diagrams and port descriptions for the Aurora 8B/10B core's status and control interface, along with the GTP/GTX transceiver serial I/O interface and the sideband initialization ports that are used exclusively for simplex modules.

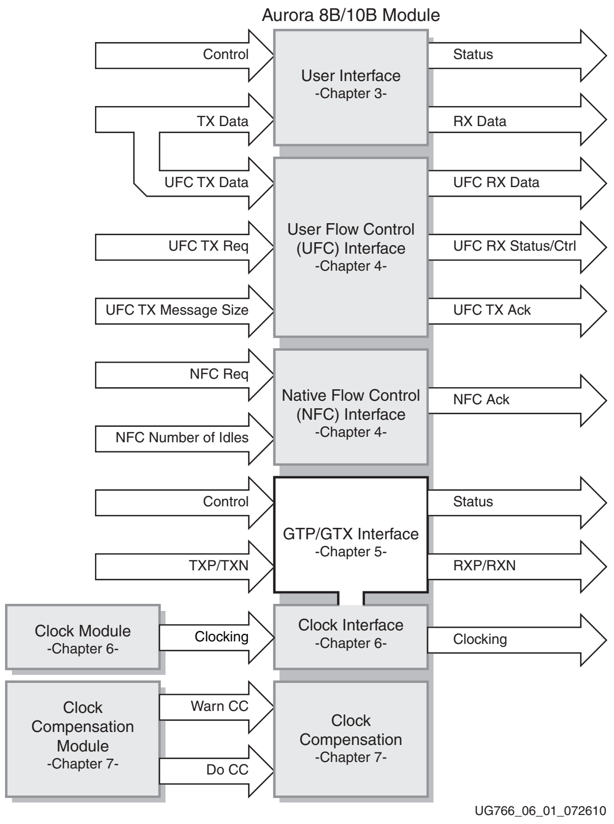


Figure 5-1: **Top-Level GTP/GTX Block Interface**

Full-Duplex Cores

Full-Duplex Status and Control Ports

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

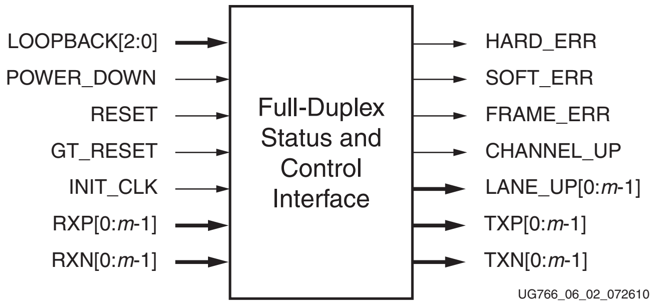


Figure 5-2: **Status and Control Interface for Full-Duplex Cores**

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

Name	Direction	Description
CHANNEL_UP	Output	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data. The Aurora 8B/10B core cannot receive data before CHANNEL_UP.
LANE_UP[0:*m*-1]	Output	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High). The Aurora 8B/10B core can only receive data after all LANE_UP signals are High.
FRAME_ERR	Output	Channel frame/protocol error detected. This port is active-High and is asserted for a single clock.
HARD_ERR	Output	Hard error detected. (Active-High, asserted until Aurora 8B/10B core resets). See Error Signals in Full-Duplex Cores, page 55 for more details.
LOOPBACK[2:0]	Input	The LOOPBACK[2:0] port selects between the normal operation mode and the different loopback modes. See the *Virtex-6 FPGA GTX Transceivers User Guide* and the *Spartan-6 FPGA GTP Transceivers User Guide* for details about loopback. See Introduction in Chapter 1.
POWER_DOWN	Input	Drives the power-down input of the GTP/GTX transceiver (active-High).
RESET	Input	Resets the Aurora 8B/10B core (active-High). This signal must be synchronous to USER_CLK and must be asserted for at least one USER_CLK cycle.

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

Name	Direction	Description
SOFT_ERR	Output	Soft error detected in the incoming serial stream. See Error Signals in Full-Duplex Cores, page 55 for more details. (Active-High, asserted for a single clock).
RXP[0:*m*-1]	Input	Positive differential serial data input pin.
RXN[0:*m*-1]	Input	Negative differential serial data input pin.
TXP[0:*m*-1]	Output	Positive differential serial data output pin.
TXN[0:*m*-1]	Output	Negative differential serial data output pin.
GT_RESET	Input	The reset signal for the PMA modules in the transceivers is connected to the top level through a debouncer. The GT_RESET should be asserted (active-High) when the module is first powered up in hardware. This systematically resets all PCS and PMA subcomponents of the transceiver. The signal is debounced using the INIT_CLK. See the Reset section in the respective transceiver user guide for further details.
INIT_CLK	Input	INIT_CLK is used to register and debounce the GT_RESET signal. INIT_CLK is required because USER_CLK stops when GT_RESET is asserted. INIT_CLK should be set to a slow rate, preferably slower than the reference clock. INIT_CLK is a board clock. For example, the ML623 board has a 200 MHz crystal oscillator, and it is constrained for this frequency by default in `<component name>_example_design.ucf`. Users need to update this clock constraint with respect to their board clock frequency.

Notes:

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

Error Signals in Full-Duplex Cores

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

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

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

Aurora 8B/10B cores with a AXI4-Stream data interface can also detect errors in Aurora 8B/10B frames. Errors of this type include frames with no data, consecutive Start of Frame symbols, and consecutive End of Frame symbols. When the core detects a frame problem, it asserts the FRAME_ERR signal. This signal is usually asserted close to a SOFT_ERR assertion, with soft errors being the main cause of frame errors.

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

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

Signal	Description
HARD_ERR	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. RX Overflow/Underflow: The elastic buffer for RX data overflows or underflows. This can occur when the clock source frequencies for the two channel partners are not within ± 100 ppm. Bad Control Character: The protocol engine attempts to send a bad control character. This is an indication of design corruption or catastrophic failure. Soft Errors: There are too many soft errors within a short period of time. The Aurora 8B/10B protocol defines a leaky bucket algorithm for determining the acceptable number of soft errors within a given time period. When this number is exceeded, the physical connection might be too poor for communication using the current voltage swing and pre-emphasis settings.
SOFT_ERR	Invalid Code: The 10-bit code received from the channel partner was not a valid code in the 8B/10B table. This usually means a bit was corrupted in transit, causing a good code to become unrecognizable. Typically, this also results in a frame error or corruption of the current channel frame. Disparity Error: The 10-bit code received from the channel partner did not have the correct disparity. This error is also usually caused by corruption of a good code in transit, and can result in a frame error or bad data if it occurs while a frame is being sent.
FRAME_ERR	Truncated Frame: A channel frame is started without ending the previous channel frame, or a channel frame is ended without being started. Invalid Control Character: The protocol engine receives a control character that it does not recognize. No Data in Frame: A channel frame is received with no data.

Full-Duplex Initialization

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

Aurora 8B/10B cores cannot receive data before CHANNEL_UP is asserted. Only the M_AXI_RX_TVALID 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, because no transmission can occur until after CHANNEL_UP. If user application modules need to be reset before data reception, one of the LANE_UP signals can be inverted and used. Data cannot be received until after all the LANE_UP signals are asserted.

Simplex Cores

Simplex TX Status and Control Ports

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

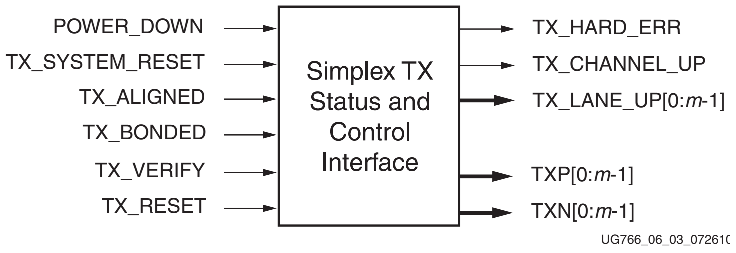


Figure 5-3: **Status and Control Interface for Simplex TX Core**

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

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

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

Name	Direction	Description
TX_RESET	Input	Asserted when reset is required because of initialization status of RX channel partner. This signal must be synchronous to USER_CLK and must be asserted for at least one USER_CLK cycle. Typically connected to RX_RESET.
TX_CHANNEL_UP	Output	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data. The Aurora 8B/10B core cannot receive data before TX_CHANNEL_UP.
TX_LANE_UP[0:*m*-1]	Output	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High).
TX_HARD_ERR	Output	Hard error detected. (Active-High, asserted until Aurora 8B/10B core resets). See Error Signals in Simplex Cores, page 62 for more details.
POWER_DOWN	Input	Drives the powerdown input of the GTP/GTX transceiver (active-High).
TX_SYSTEM_RESET	Input	Resets the Aurora 8B/10B core (active-High).
TXP[0:*m*-1]	Output	Positive differential serial data output pin.
TXN[0:*m*-1]	Output	Negative differential serial data output pin.

Notes:

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

Simplex RX Status and Control Ports

Simplex RX cores allow user applications to receive data from a simplex TX core. Figure 5-4 shows the status and control interface for a simplex RX core. Table 5-4, page 59 describes the function of each of the ports in the interface.

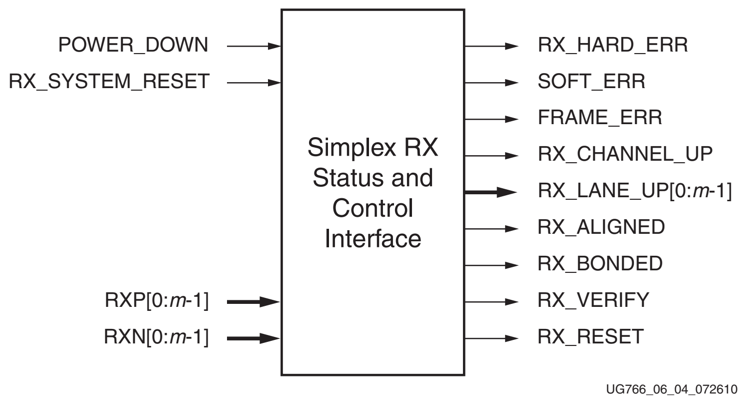


Figure 5-4: **Status and Control Interface for Simplex RX Core**

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

Name	Direction	Description
RX_ALIGNED	Output	Asserted when RX module has completed lane initialization. Typically connected to TX_ALIGNED.
RX_BONDED	Output	Asserted when RX module has completed channel bonding. Not used for single-lane channels. Typically connected to TX_BONDED.
RX_VERIFY	Output	Asserted when RX module has completed verification. Typically connected to TX_VERIFY.
RX_RESET	Output	Asserted when the RX module needs the TX module to restart initialization. Typically connected to TX_RESET.
RX_CHANNEL_UP	Output	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data. The Aurora 8B/10B core cannot receive data before RX_CHANNEL_UP.
RX_LANE_UP[0:*m*-1]	Output	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High). The Aurora 8B/10B core can only receive data after all RX_LANE_UP signals are High.
FRAME_ERR	Output	Channel frame/protocol error detected. This port is active-High and is asserted for a single clock.
RX_HARD_ERR	Output	Hard error detected. (Active-High, asserted until Aurora 8B/10B core resets). See Error Signals in Simplex Cores, page 62 for more details.
POWER_DOWN	Input	Drives the power-down input of the GTP/GTX transceiver (active-High).
RX_SYSTEM_RESET	Input	Resets the Aurora 8B/10B core (active-High).
SOFT_ERR	Output	Soft error detected in the incoming serial stream. See Error Signals in Simplex Cores, page 62 for more details. (Active-High, asserted for a single clock).
RXP[0:*m*-1]	Input	Positive differential serial data input pin.
RXN[0:*m*-1]	Input	Negative differential serial data input pin.

Notes:

1. *m* is the number of GTP/GTX transceivers.
2. RX_ALIGNED, RX_BONDED, RX_VERIFY, and RX_RESET are available as output signals even when the simplex partner is timer based, but functionally these signals are not required.

Simplex Both (RX and TX) Status and Control Ports

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

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

Figure 5-5 shows the status and control interface for a simplex Both core. Table 5-5, page 60 describes the function of each of the ports in the interface.

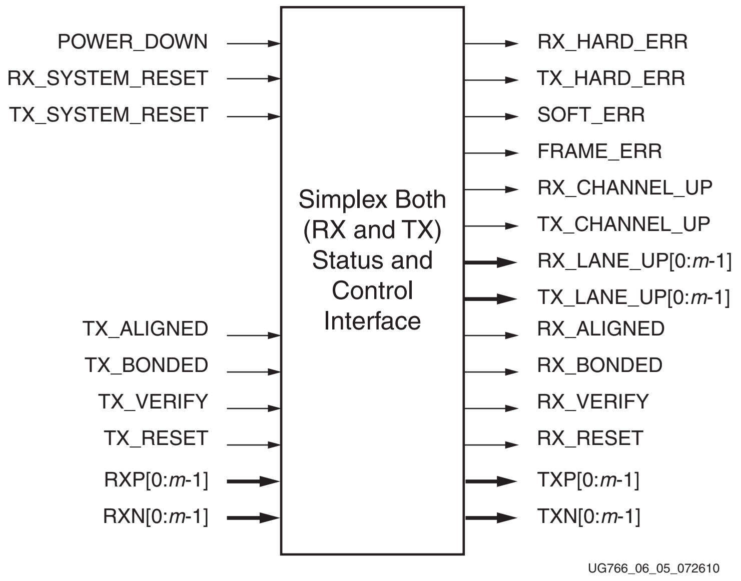


Figure 5-5: **Status and Control Interface for Simplex Both Cores**

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

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

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

Name	Direction	Description
TX_VERIFY	Input	Asserted when RX channel partner has completed verification. Typically connected to RX_VERIFY.
RX_VERIFY	Output	Asserted when RX module has completed verification. Typically connected to TX_VERIFY.
TX_RESET	Input	Asserted when reset is required because of initialization status of RX channel partner. Typically connected to RX_RESET.
RX_RESET	Output	Asserted when the RX module needs the TX module to restart initialization. Typically connected to TX_RESET.
RX_CHANNEL_UP	Output	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to receive data. The Aurora 8B/10B core cannot receive data before CHANNEL_UP.
TX_CHANNEL_UP	Input	Asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data. The Aurora 8B/10B core cannot receive data before CHANNEL_UP.
RX_LANE_UP[0:*m*-1]	Output	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High). The Aurora 8B/10B core can only receive data after all LANE_UP signals are High.
TX_LANE_UP[0:*m*-1]	Input	Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High). The Aurora 8B/10B core can only transmit data after all LANE_UP signals are High.
FRAME_ERR	Output	Channel frame/protocol error detected. This port is active-High and is asserted for a single clock.
RX_HARD_ERR TX_HARD_ERR	Output	Hard error detected. (Active-High, asserted until Aurora 8B/10B core resets). See Error Signals in Simplex Cores, page 62 for more details.
POWER_DOWN	Input	Drives the power-down input of the GTP/GTX transceiver (active-High).
RX_SYSTEM_RESET TX_SYSTEM_RESET	Input	Resets the Aurora 8B/10B core (active-High).
SOFT_ERR	Output	Soft error detected in the incoming serial stream. See Error Signals in Simplex Cores, page 62 for more details. (Active-High, asserted for a single clock).
RXP[0:*m*-1]	Input	Positive differential serial data input pin.
RXN[0:*m*-1]	Input	Negative differential serial data input pin.
TXP[0:*m*-1]	Output	Positive differential serial data output pin.
TXN[0:*m*-1]	Output	Negative differential serial data output pin.

Notes:

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

Error Signals in Simplex Cores

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

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

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

Simplex RX and simplex Both cores with a AXI4-Stream data interface can also detect errors in Aurora 8B/10B frames when they are received. Errors of this type include frames with no data, consecutive Start of Frame symbols, and consecutive End of Frame symbols. When the core detects a frame problem, it asserts the FRAME_ERR signal. This signal is usually asserted close to a SOFT_ERR assertion, as soft errors are the main cause of frame errors. Simplex TX modules do not use the FRAME_ERR port.

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

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

Signal	Description	TX	RX	Both
HARD_ERR	TX Overflow/Underflow: The elastic buffer for TX data overflows or underflows. This can occur when the user clock and the reference clock sources are not running at the same frequency.	x		x
	RX Overflow/Underflow: The elastic buffer for RX data overflows or underflows. This can occur when the clock source frequencies for the two channel partners are not within ± 100 ppm.		x	x
	Bad Control Character: The protocol engine attempts to send a bad control character. This is an indication of design corruption or catastrophic failure.	x		x
	Soft Errors: There are too many soft errors within a short period of time. The Aurora 8B/10B protocol defines a leaky bucket algorithm for determining the acceptable number of soft errors within a given time period. When this number is exceeded, the physical connection might be too poor for communication using the current voltage swing and pre-emphasis settings.		x	x

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

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

Simplex Initialization

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

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

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

Both initialization methods are described in the following sections.

Using a Back Channel

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

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

Using Timers

For some systems a back channel is not possible. In these cases, serial channels can be initialized by driving the TX simplex initialization with a set of timers. The timers must be designed carefully to meet the needs of the system because the average time for initialization depends on many channel specific conditions such as clock rate, channel latency, skew between lanes, and noise. C_ALIGNED_TIMER, C_BONDED_TIMER and C_VERIFY_TIMER are timers used for assertion of TX_ALIGNED, TX_BONDED and TX_VERIFY signals, respectively. These timers use worst-case values obtained from corner case functional simulations and implemented in the <component name> module.

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

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

Reset and Power Down

Reset

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

On full-duplex modules, the RESET signal resets both the TX and RX sides of the channel when asserted on the positive edge of USER_CLK. On simplex modules, the resets for the TX and RX channels are separate. TX_SYSTEM_RESET resets TX channels; RX_SYSTEM_RESET resets RX channels. The TX_SYSTEM_RESET is separate from the TX_RESET and RX_RESET signals used on the simplex sideband interface.

Power Down

This is an active-High signal. When POWER_DOWN is asserted, the GTP/GTX transceivers in the Aurora 8B/10B core are turned off, putting them into a non-operating low-power mode. When POWER_DOWN is deasserted, the core automatically resets. Be careful when asserting this signal on cores that use TX_OUT_CLK (see the Chapter 6, Clock Interface and Clocking). TX_OUT_CLK stops when the GTP/GTX transceivers are powered down. See the *Virtex-6 FPGA GTX Transceivers User Guide* or the *Spartan-6 FPGA GTP Transceivers User Guide* for the device being used for details about powering down GTP/GTX transceivers.

Timing

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

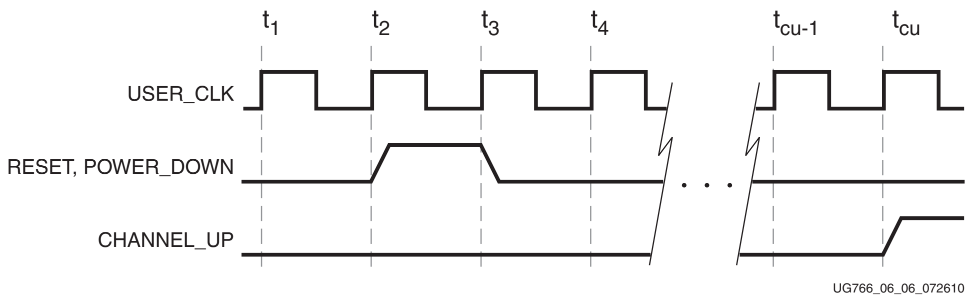


Figure 5-6: **Reset and Power Down Timing**

Chapter 6

Clock Interface and Clocking

Introduction

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

The Virtex®-6 FPGA has four GTX transceivers in a Quad. The Spartan®-6 FPGA GTP architecture has a pair of transceivers in each GTPA1_DUAL tile. The Virtex-6 FPGA GTX transceiver has individual PLLs for both TX and RX portion of the transceivers. The Spartan-6 FPGA has individual PLLs for each transceiver in a GTPA1_DUAL tile. The reference clock is used to produce the PLL clock, which is divided to make individual TX and RX serial clocks and parallel clocks in each GTP/GTX transceiver.

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

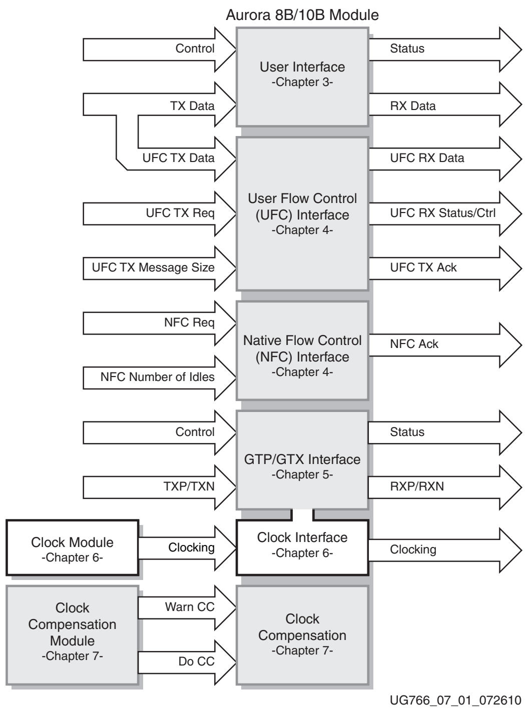


Figure 6-1: **Top-Level Clocking**

Clock Interface Ports for GTP/GTX Transceiver Cores

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

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

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

Clocking from a Neighboring GTX_QUAD Tile for Virtex-6 FPGA Designs

The Xilinx implementation tools make necessary adjustments to the north-south routing shown in Figure 6-2, page 70 as well as pin swapping necessary to GTXE1 clock inputs to route clocks from one Quad to another when required.

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

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

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

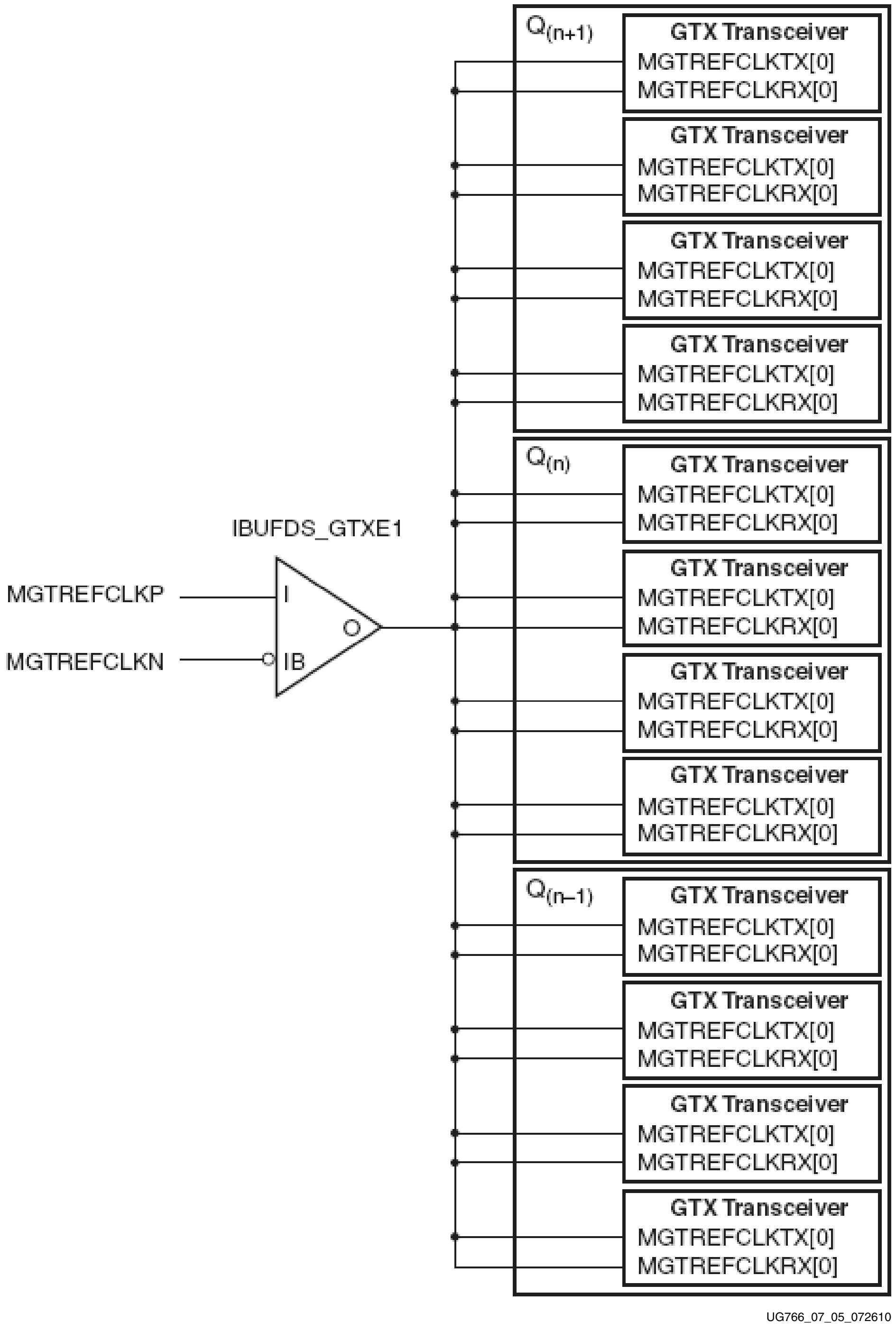


Figure 6-2: **North-South Routing Adjustments in Virtex-6 FPGAs**

Reference Clocks for Spartan-6 FPGA GTP Transceiver Designs

In Spartan-6 FPGA transceiver designs, the reference clock is GTPD, which is a differential input clock for each GTPA1_DUAL. The reference clock for GTPA1_DUAL is provided through the CLK00 and CLK01 ports. The two possible use models for distributing a reference clock to drive the CLK00 and CLK01 ports are:

- Clocking from and External Source
- Clocking from a Neighboring GTPA1_DUAL Tile

Clocking from and External Source

Each GTPA1_DUAL tile has a pair of dedicated pins that can be connected to an external clock source. To use these pins, a IBUFDS primitive is instantiated. In the user constraints file (`.ucf`), the IBUFDS input pins are set to the dedicated clock pins for the tile. In the design, the output of the IBUFDS is connected to the CLK00 and CLK01 ports. Each GTPA1_DUAL takes differential GTPD clock inputs, which are directly bonded to the FPGA pins. For multilane Aurora 8B/10B designs, GTPD clock of any GTPA1_DUAL can be used as the reference clock for the Aurora 8B/10B design. Using a low-jitter oscillator delivers a high-quality clock suitable for top-speed operation. Figure 6-3 shows a differential GTPA1_DUAL clock pin pair sourced by an external oscillator on the board. This clocking mechanism is used for Spartan-6 FPGA single lane and 2-lane designs.

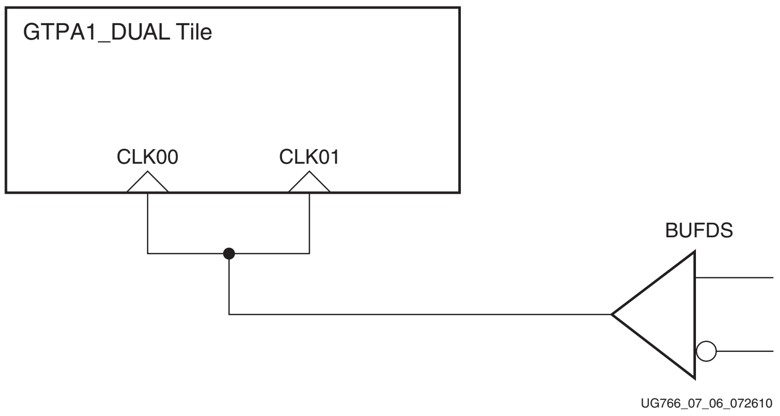


Figure 6-3: **Single GTPA1_DUAL Tile Clocked Externally**

Clocking from a Neighboring GTPA1_DUAL Tile

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

The example in Figure 6-4 uses the clock from one GTPA1_DUAL tile to clock neighboring tiles. A GTPA1_DUAL tile shares its clock with its neighbors using dedicated clock routing resources. This clocking mechanism is used for Spartan-6 FPGA 4-lane design.

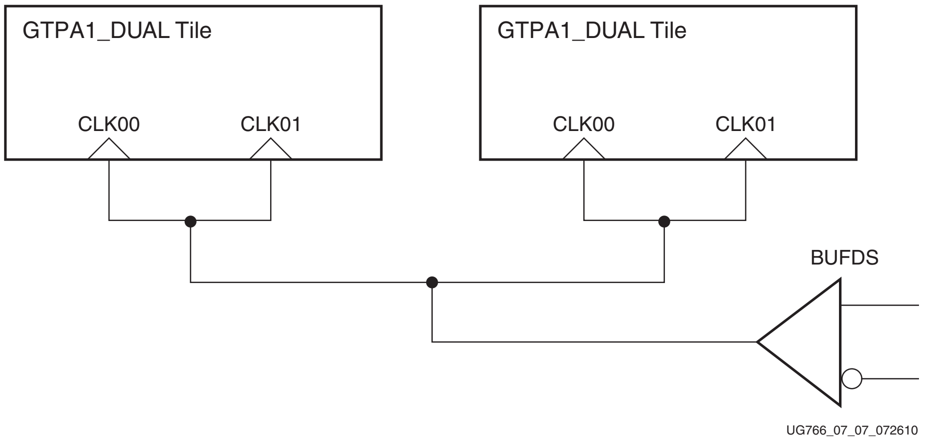


Figure 6-4: **GTPA1_DUAL Tiles with Shared Reference Clock**

Clock Rates for GTP/GTX Transceiver Designs

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

Chapter 7

Clock Compensation

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 8B/10B channel. This feature is used in systems where a separate reference clock source is used for each device connected by the channel, and where the same USER_CLK is used for transmitting and receiving data.

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

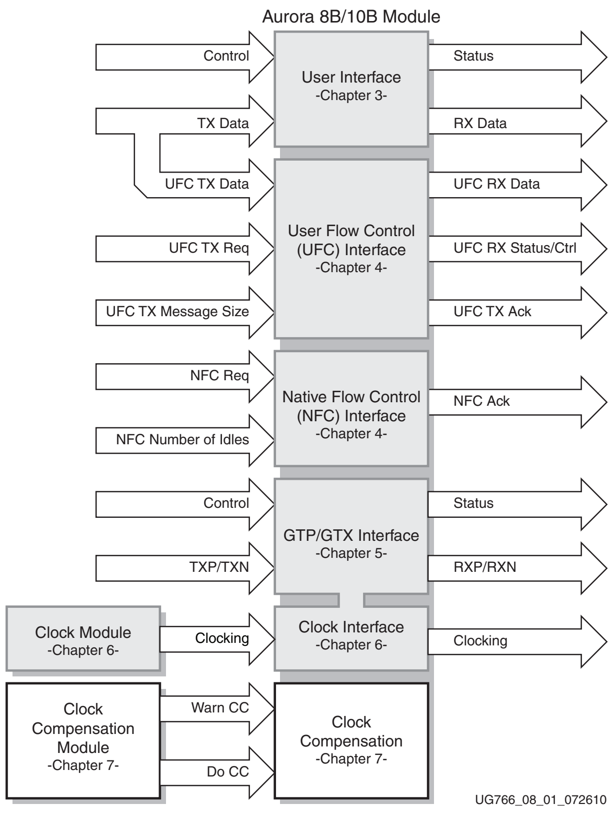


Figure 7-1: **Top-Level Clock Compensation**

Clock Compensation Interface

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

Table 7-1: **Clock Compensation I/O Ports**

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

Figure 7-2 and Figure 7-3 are waveform diagrams showing how the DO_CC signal works.

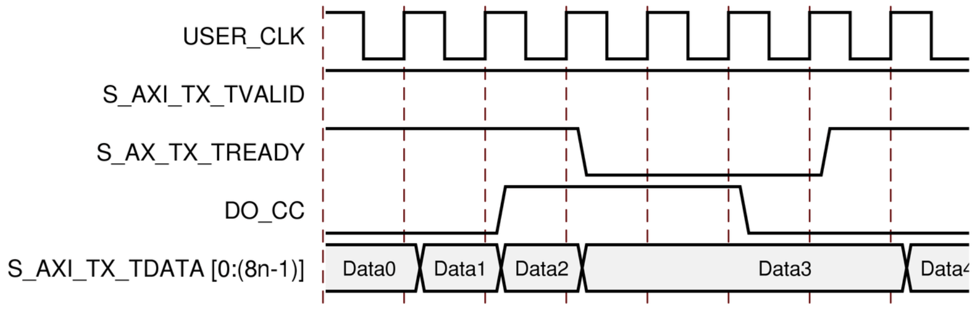


Figure 7-2: **Streaming Data with Clock Compensation Inserted**

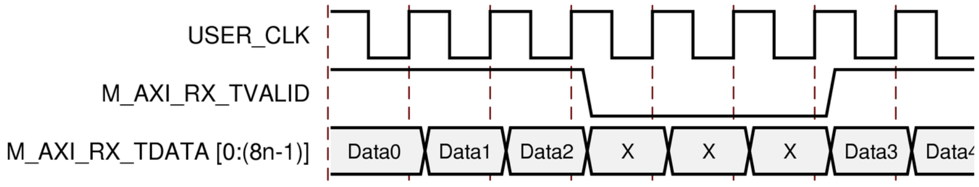


Figure 7-3: **Data Reception Interrupted by Clock Compensation**

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

Table 7-2: **Clock Compensation Cycles**

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

WARN_CC is for cores with user flow control (UFC) and/or native flow control (NFC). Driving this signal before DO_CC is asserted prevents the UFC interface from acknowledging and sending UFC messages too close to a clock correction sequence. This precaution is necessary because data corruption occurs when CC sequences and UFC messages overlap. The number of lookahead cycles required to prevent a 16-byte UFC message from colliding with a clock compensation sequence depends on the number of lanes in the channel and the width of each lane. Table 7-3, page 76 shows the number of lookahead cycles required for each combination of lane width, channel width, and maximum UFC message size.

Table 7-3: **Lookahead Cycles**

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

Native flow control message requests are not acknowledged during assertion of WARN_CC and DO_CC signals. This helps to prevent the collision of NFC message and clock compensation sequence.

To make Aurora 8B/10B compliance easy, a standard clock compensation module is generated along with each Aurora 8B/10B core from the CORE Generator™ software in the cc_manager subdirectory. It automatically generates pulses to create Aurora 8B/10B compliant clock compensation sequences on the DO_CC port and sufficiently early pulses on the WARN_CC port to prevent UFC collisions with maximum-sized UFC messages. This module always be connected to the clock compensation port on the Aurora 8B/10B module, except in special cases. Table 7-4, page 77 shows the port description for the standard CC module.

Table 7-4: **Standard CC I/O Port**

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

Clock compensation is not needed when both sides of the Aurora 8B/10B channel are being driven by the same clock (see Figure 7-3, page 75) because the reference clock frequencies on both sides of the module are locked. In this case, WARN_CC and DO_CC should both be tied to ground. Additionally, the CLK_CORRECT_USE attribute can be set to false in the transceiver interface module for the core. This can result in lower latencies for single lane modules.

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

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

Chapter 8

Quick Start Example Design

This chapter introduces the example design that is included with the LogiCORE™ IP Aurora 8B/10B core. The quick start instructions are a step-by-step procedure for generating an Aurora 8B/10B core, implementing the core in hardware using the accompanying example design, and simulating the core with the provided demonstration test bench (demo_tb). For detailed information about the example design provided with the Aurora 8B/10B core, see Chapter 10, Project Directory Structure.

Overview

The quick start example consists of the following components:

- An instance of the Aurora 8B/10B core generated using the default parameters
  - Full-duplex with a single GTP/GTX transceiver
  - AXI4-Stream interface
- A demonstration test bench to simulate two instances of the example design

The Aurora 8B/10B example design has been tested with XST for synthesis and Mentor Graphics ModelSim for simulation.

Generating the Core

To generate an Aurora 8B/10B core with default values using the CORE Generator™ tool:

1. Start the CORE Generator software from a required directory.

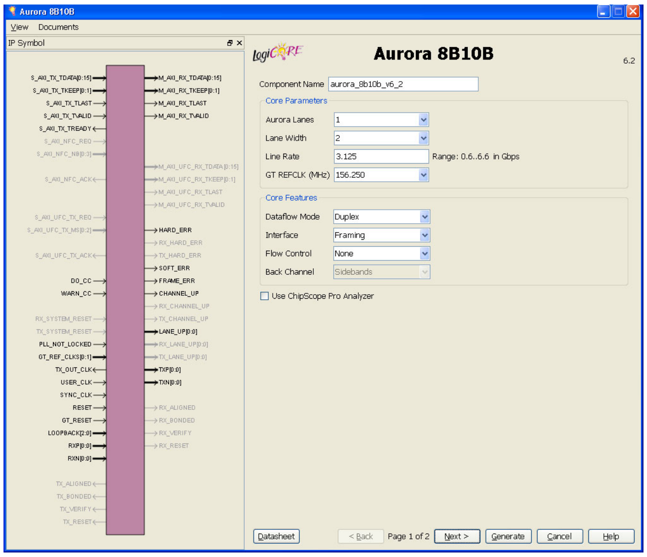
   For help starting and using the CORE Generator software, see *CORE Generator Help* in the ISE® software documentation.
2. Choose **File** → **New Project**.
3. Type a project name.
4. To set project options:
   - From the Part tab, select a silicon family, part, speed grade, and package that supports the Aurora 8B/10B core, for example, Virtex®-6 FPGAs.

     Note: If an unsupported silicon family is selected, the Aurora 8B/10B appears light grey in the taxonomy tree and cannot be customized. Only devices containing GTP/GTX transceivers are supported by the core. For a list of supported architectures, see the *LogiCORE IP Aurora 8B/10B v6.2 Data Sheet*.
   - No further project options need to be set.
   - Optionally, on the Generation tab, set the Design Entry pull-down to **Verilog**.

5. After creating the project, locate the Aurora 8B/10B core v6.2 in the taxonomy tree under:

   `/Communication_&_Networking/Serial_Interfaces`

6. Double-click the core.
7. In the Component Name field (Figure 8-1), enter a name for the core instance. This example uses the name **aurora_8b10b_v6_2**.



Aurora 8B10B 6.2

Component Name: aurora_8b10b_v6_2

Core Parameters
- Aurora Lanes: 1
- Lane Width: 2
- Line Rate: 3.125 Range: 0.6..6.6 in Gbps
- GT REFCLK (MHz): 156.250

Core Features
- Dataflow Mode: Duplex
- Interface: Framing
- Flow Control: None
- Back Channel: Sidebands

☐ Use ChipScope Pro Analyzer

Figure 8-1: **CORE Generator Aurora 8B/10B Customization Screen**

8. Click **Generate**.

The core and its supporting files, including the example design, are generated in the project directory. For detailed information about the example design files and directories, see Chapter 10, Project Directory Structure.

Simulating the Example Design

The Aurora 8B/10B core provides a quick way to simulate and observe the behavior of the core using the provided example design. Prior to simulating the core, the functional (gate-level) simulation models must be generated. You must compile all source files in the following directories to a single library as shown in Table 8-1. Refer to the *Synthesis and Verification Design Guide* for ISE 13.1 software for instructions on how to compile ISE software simulation libraries.

Table 8-1: **Required Simulation Libraries**

HDL	Library	Source Directories
Verilog	UNISIMS_VER	<*Xilinx dir*>/verilog/src/unisims <*Xilinx dir*>/ secureip/<*SIMULATOR*>
VHDL	UNISIM	<*Xilinx dir*>/vhdl/src/unisims <*Xilinx dir*>/ secureip/<*SIMULATOR*>

Notes:
1. SIMULATOR can be ModelSim.

The Aurora 8B/10B core provides a command line script to simulate the example design. To run a VHDL or Verilog ModelSim simulation of the Aurora 8B/10B core, use the following instructions:

1. Launch the ModelSim simulator and set the current directory to:

   ```
   <project directory>/aurora_8b10b_v6_2/simulation/functional
   ```

2. Set the MTI_LIBS variable:

   ```
   modelsim> setenv MTI_LIBS <path to compiled libraries>
   ```

3. Launch the simulation script:

   ```
   modelsim> do simulate_mti.do
   ```

The ModelSim script compiles the example design and test bench, and adds the relevant signals to the wave window. After the design is compiled and the wave window is displayed, run the simulation to see the Aurora 8B/10B core power up, followed by Aurora 8B/10B channel initialization and data transfer. Data transfer begins after the CHANNEL_UP signal goes High.

Implementing the Example Design

After the core is generated, the design can be processed by the Xilinx implementation tools. The generated output files include several scripts to assist the user in running the Xilinx software.

From the command prompt, navigate to the project directory and type the following:

For Windows

```
ms-dos> cd aurora_8b10b_v6_2\implement
ms-dos> .\implement.bat
```

For Linux

```
% cd aurora_8b10b_v6_2/implement
%./implement.sh
```

These commands execute a script that synthesizes, translates, maps, place-and-routes the example design and produces a bitstream file. The resulting files are placed in the results directory created within the implement directory.

Using ChipScope Pro Cores with the Aurora 8B/10B Core

Description

The ChipScope™ Pro ICON and VIO cores aid in debugging and validating the design in boards and are provided with the Aurora 8B/10B core. Select the **Use ChipScope Pro Analyzer** checkbox from the core GUI to include it as a part of the example design. Alternatively, the USE_CHIPSCOPE parameter in the `<component name>_example_design` module can be set to 1 before running implementation.

Chapter 9

Example Design Overview

Introduction

Each Aurora 8B/10 B core includes an example design (aurora_example) that uses the core in a simple data transfer system. For more details about the example_design directory, see Chapter 10, Project Directory Structure.

Figure 9-1 illustrates the block diagram of the example design for a full-duplex core. Table 9-1, page 84 describes the ports of the example design.

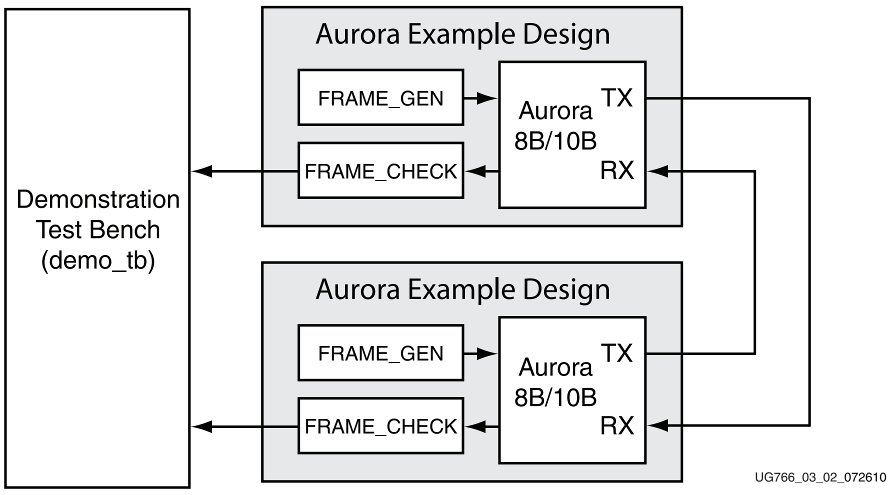


Figure 9-1: **Example Design**

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

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

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

Table 9-1: **Example Design I/O Ports**

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

Chapter 10

Project Directory Structure

The customized Aurora 8B/10B core is delivered as a set of HDL source modules in the language selected in the CORE Generator™ software project with supporting script and documentation files. These files are arranged in a predetermined directory structure under the project directory name provided to the CORE Generator software when the project is created as shown in this section.

Directory and File Structure

<project directory>
Top-level project directory; name is user-defined.

- <project directory>/<component name>
  Top-level core directory; contains the core deliverables and the readme file
  - <component name>/doc
    Product documentation
  - <component name>/example_design
    Example design and user constraints files
    - /example_design/cc_manager
      Verilog/VHDL design files for the clock compensation block
    - /example_design/clock_module
      Verilog/VHDL design files for the clocking blocks
    - /example_design/gt
      Verilog/VHDL wrapper files for the GTP/GTX transceivers
    - /example_design/traffic_gen_check
      Verilog/VHDL design files for the frame generator and checker
  - <component name>/implement
    Implementation scripts and support files
  - <component name>/simulation
    Simulation test bench and simulation script files
    - /simulation/functional
      Functional simulation files
    - /simulation/timing
      Timing simulation file
  - <component name>/src
    Verilog/VHDL files for the core

Directory and File Contents

The Aurora 8B/10B core directories and their associated files are defined in the following sections.

<project directory>

The project directory contains the CORE Generator tool project files.

Table 10-1: **project Directory**

Name	Description
`<project directory>`	
`<coregen project filename>.cgp`	CORE Generator tool project file
`<component name>.xise`	Xilinx® ISE® software project file

Back to Top

<project directory>/<component name>

This top-level core directory contains the core deliverables and the readme file.

Table 10-2: **component name Directory**

Name	Description
`<project directory>/<component name>`	
`aurora_8b10b_readme.txt`	Release notes file

Back to Top

<component name>/doc

The doc directory contains the product documentation.

Table 10-3: **doc Directory**

Name	Description
`<component name>/doc`	
`aurora_8b10b_ds797.pdf`	*LogiCORE IP Aurora 8B/10B v6.2 Data Sheet*
`aurora_8b10b_ug766.pdf`	*LogiCORE IP Aurora 8B/10B v6.2 User Guide*

Back to Top

<component name>/example_design

The example_design directory contains the example design and user constraints files provided with the core.

Table 10-4: **example_design Directory**

Name	Description
`<component name>/example_design`	
`<component name>_example_design.v[hd]`	Example design top-level file
`<component name>_example_design.ucf`	Aurora 8B/10B example design constraints
`<component name>_reset_logic.v[hd]`	Aurora 8B/10B reset logic
`<component name>.v[hd]`	Aurora 8B/10B core source file

Back to Top

/example_design/cc_manager

The cc_manager directory contains the clock compensation source file.

Table 10-5: **cc_manager Directory**

Name	Description
`<component name>/example_design/cc_manager`	
`<component name>_standard_cc_module.v[hd]`	Clock compensation module source file

Back to Top

/example_design/clock_module

The clock_module directory contains the clock module source file.

Table 10-6: **clock_module Directory**

Name	Description
`<component name>/example_design/clock_module`	
`<component name>_clock_module.v[hd]`	Clock module source file

Back to Top

/example_design/gt

The gt directory contains the Verilog/VHDL wrapper files for the GTP/GTX transceivers.

Table 10-7: **gt Directory**

Name	Description
`<component name>/example_design/gt`	

`<component name>_transceiver_tile.v[hd]`
`<component name>_transceiver_wrapper.v[hd]`
	Verilog/VHDL wrapper files for the transceiver

Back to Top

/example_design/traffic_gen_check

The traffic_gen_check directory contains frame generator and frame checker modules for Aurora 8B/10B core.

Table 10-8: **traffic_gen_check Directory**

Name	Description
`<component name>/example_design/traffic_gen_check`	

`<component name>_frame_check.v[hd]`
`<component name>_frame_gen.v[hd]`
	Example design traffic generation and checker files

Back to Top

<component name>/implement

The implement directory contains scripts and support files for both Linux and Windows operating systems. These scripts automate the process of synthesizing and implementing the files needed for the example design.

Table 10-9: **implement Directory**

Name	Description
`<component name>/implement`	
`implement.bat`	Windows batch file that processes the example design through the Xilinx ISE tool flow
`implement.sh`	Linux shell script that processes the example design through the Xilinx ISE tool flow
`implement_planahead.tcl`	Script that processes the example design through the Xilinx PlanAhead™ tool flow
`xst.scr`	XST script file for the example design
`xst.prj`	XST project file for the example design
`v6_icon.ngc` `v6_vio.ngc`	Virtex-6 family NGC files for the debug cores compatible with the ChipScope™ Pro Analyzer tool
`s6_icon.ngc` `s6_vio.ngc`	Spartan-6 family NGC files for the debug cores compatible with the ChipScope Pro Analyzer tool

Back to Top

<component name>/simulation

The simulation directory contains the test bench files for the example design.

Table 10-10: **simulation Directory**

Name	Description
<component name>/simulation	
`demo_tb.v[hd]`	Test bench file for simulating the example design

Back to Top

/simulation/functional

The functional directory contains functional simulation scripts provided with the core.

Table 10-11: **functional Directory**

Name	Description
`<component name>/simulation/functional`	
`simulate_isim.bat`	ISim macro file that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion in the Microsoft Windows operating system.
`simulate_isim.sh`	ISim macro file that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion in the Linux operating system.
`wave_isim.tcl`	ISim macro file that opens a Wave window with top-level signals.
`simulate_mti.do`	ModelSim macro file that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.
`wave_mti.do`	ModelSim macro file that opens a Wave window.

Back to Top

/simulation/timing

The timing directory contains the timing simulation scripts provided with the core.

Table 10-12: **timing Directory**

Name	Description
`<component name>/simulation/timing`	
`simulate_mti.do`	ModelSim macro file that compiles the post place and route netlist of the example design along with standard delay format (SDF) back annotation then runs timing simulation to completion

Back to Top

<component name>/src

The src directory contains the source files related to the Aurora 8B/10B example design.

Table 10-13: **src Directory**

Name	Description
`<component name>/src`	
`<component name>_aurora_lane.v[hd]`	Aurora 8B/10B source files
`<component name>_aurora_pkg.vhd (VHDL Only)`	
`<component name>_axi_to_ll.v[hd]`	
`<component name>_channel_err_detect.v[hd]`	
`<component name>_channel_init_sm.v[hd]`	
`<component name>_chbond_count_dec.v[hd]`	
`<component name>_err_detect.v[hd]`	
`<component name>_global_logic.v[hd]`	
`<component name>_idle_and_ver_gen.v[hd]`	
`<component name>_lane_init_sm.v[hd]`	
`<component name>_ll_to_axi.v[hd]`	
`<component name>_rx_ll.v[hd]`	
`<component name>_rx_ll_pdu_datapath.v[hd]`	
`<component name>_sym_dec.v[hd]`	
`<component name>_sym_gen.v[hd]`	
`<component name>_tx_ll.v[hd]`	
`<component name>_tx_ll_control.v[hd]`	
`<component name>_tx_ll_datapath.v[hd]`	

Back to Top

Handling Timing Errors Due To Far Apart Transceiver Selection

The Aurora 8B/10B core allows the user to select any combination of transceiver(s) during core generation. The design parameters that affect the timing performance are:

- Line rate
- Transceiver data path width (2/4 bytes) and
- Number of unused transceivers between two selected transceivers

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

- CHBONDO does not meet timing
- RXCHARISCOMMA, RXCHARISK, and RXCHANISALIGNED do not meet timing

Example Solutions

The following suggestions can be attempted to meet timing.

1. Select the transceivers consecutively.
   a. Use the Lane Assignment in the Aurora 8B/10B GUI to select the transceivers during core generation.

   Note: Most of the timing errors are due to unused transceivers and channel bonding signals connections among transceivers.
2. Use the smartexplorer tool provided with the ISE® software.
   a. Refer to the ISE software documentation for instructions to use smartexplorer.

Performance and Core Latency

Introduction

Latency through an Aurora 8B/10B core is caused by pipeline delays through the protocol engine (PE) and through the GTP/GTX transceivers. The PE pipeline delay increases as the AXI4-Stream interface width increases. The GTP/GTX transceivers delays are fixed per the features of the GTP/GTX transceivers.

This section outlines expected latency for the Aurora 8B/10B core's AXI4-Stream user interface in terms of USER_CLK cycles for 2-byte-per-lane and 4-byte-per-lane designs. For the purposes of illustrating latency, the Aurora 8B/10B modules partitioned into GTP/GTX transceivers logic and protocol engine (PE) logic implemented in the FPGA fabric.

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

Latency of the Frame Path

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

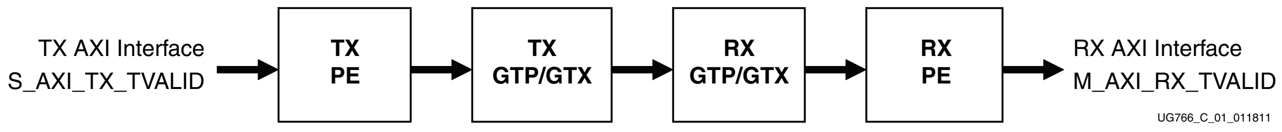


Figure B-1: **Latency of the Frame Path**

Maximum latency for a 2-byte designs from TX_SOF_N to RX_SOF_N is approximately 52 GTP/GTX transceivers clock cycles in simulation.

Maximum latency for a 4-byte designs from TX_SOF_N to RX_SOF_N is approximately 61 GTP/GTX transceivers clock cycles in simulation.

The pipeline delays are designed to maintain the clock speed.

Generating a Wrapper File from Its Respective GTP/GTX Transceiver Wizard

The transceiver attributes play a vital role in the functionality of the Aurora 8B/10B core. It is important to have the updated transceiver wrapper file. One way to achieve this is to generate the transceiver wrapper file from the latest Transceiver Wizard.

Note: This flow is strongly recommended only if the Aurora core is not released and the respective wizard core is released in the current ISE® software release.

This appendix provides instructions to generate these transceiver wrapper files:

- Case 1: Virtex-6 FPGA GTX Wrapper, page 96
- Case 2: Spartan-6 FPGA GTP Wrapper, page 97

Case 1: Virtex-6 FPGA GTX Wrapper

Use these steps to generate the transceiver wrapper file using the Virtex-6 FPGA GTX Transceiver Wizard.

1. Using the CORE Generator tool, run the latest version of the Virtex-6 FPGA GTX Transceiver Wizard. Make sure the Component Name of the transceiver wizard matches the Component Name of the Aurora 8B/10B core.
2. Select the protocol template from the following based on number of lane(s) and lane width:
   - Aurora 2-byte single lane
   - Aurora 4-byte single lane
   - Aurora 2-byte multi lane
   - Aurora 4-byte multi lane
3. Change the Line Rate in both TX and RX based on the application requirement.
4. Select the Reference Clock from the drop-down box menu in both TX and RX based on the application requirement.
5. Select transceiver(s) and the clock source(s) based on the application requirement.
6. Keep all other settings as default.
7. Generate the core.
8. Replace the `<component name>_gtx.v[hd]` file in the example_design/gt directory available in the Aurora 8B/10B core with the generated `<component name>_gtx.v[hd]` file generated from the Virtex-6 FPGA GTX Transceiver Wizard.
9. Edit the copied `<component name>_gtx.v` file and change the module name into all uppercase letters. This change is not required for VHDL designs.

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

Case 2: Spartan-6 FPGA GTP Wrapper

Use these steps to generate the transceiver wrapper file using the Spartan®-6 FPGA GTP Transceiver Wizard.

1. Using CORE Generator tool, run the latest version of Spartan-6 FPGA GTP Transceiver Wizard. Make sure the Component Name of the transceiver wizard matches the Component Name of the Aurora 8B/10B core.
2. Select transceiver(s) and the clock source(s) based on application requirement.
3. Select the protocol template to either aurora single lane or aurora multi lane based on the number of lane(s).
4. Change the Target Line Rate in Gbps based on application requirement.
5. Select the Reference Clock from the drop-down box menu based on application requirement.
6. Select RXCHARISCOMMA and RXCHARISK ports in 8B/10B Optional Ports, if not selected by default.
7. Select TXBUFSTATUS port in Synchronization and Clocking, if not selected by default.
8. Keep all other settings as default.
9. Generate the core.
10. Replace the `<component name>_tile.v[hd]` file in the example_design/gt directory available in the Aurora 8B/10B core with the generated `<component name>_tile.v[hd]` file generated from the Spartan-6 FPGA GTP Transceiver Wizard.
11. Edit the copied `<component name>_tile.v` file and change the module name into all uppercase letters. This change is not required for VHDL designs.

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

Appendix D

Aurora AXI4-Stream Migration Guide

Introduction

This guide explains about migrating legacy (LocalLink based) Aurora cores to the AXI4-Stream Aurora core.

Pre-Requisites

- ISE® 13.1 software build containing the Aurora 8B/10B v6.2 core supporting the AXI4-Stream protocol
- Familiarity with the Aurora directory structure
- Familiarity with running the Aurora example design
- Basic knowledge of the AXI4-Stream and LocalLink protocols
- Latest data sheet (DS797) and user guide (UG766) of the core with the AXI4-Stream updates
- Legacy data sheet (DS637) and data sheet (UG353) for reference
- Migration guide (this Appendix)

Overview of Major Changes

The major change to the core is the addition of the AXI4-Stream interface:

- The user interface is modified from the legacy LocalLink (LL) to AXI4-Stream
- All AXI4-Stream signals are active High, whereas LocalLink signals are active Low
- The user interface in the example design and design top file is AXI4-Stream
- A new shim module is introduced in the AXI4-Stream Aurora core to convert AXI4-Stream signals to LL and LL back to AXI4-Stream
  - The AXI4-Stream to LL shim on the transmit converts all AXI4-Stream signals to LL
  - The shim deals with active-High to active-Low conversions of signals between AXI4-Stream and LocalLink
  - Generation of SOF_N and REM bits mapping is handled by the shim
  - The LL to AXI4-Stream shim on the receive converts all LL signals to AXI4-Stream
- Each interface (PDU, UFC, and NFC) has a separate AXI4-Stream to LL and LL to AXI4-Stream shim instantiated from the design top file
- Frame generator and checker has respective LL to AXI4-Stream and AXI4-Stream to LL shim instantiated in the Aurora example design to interface with the generated AXI4-Stream design

Block Diagram

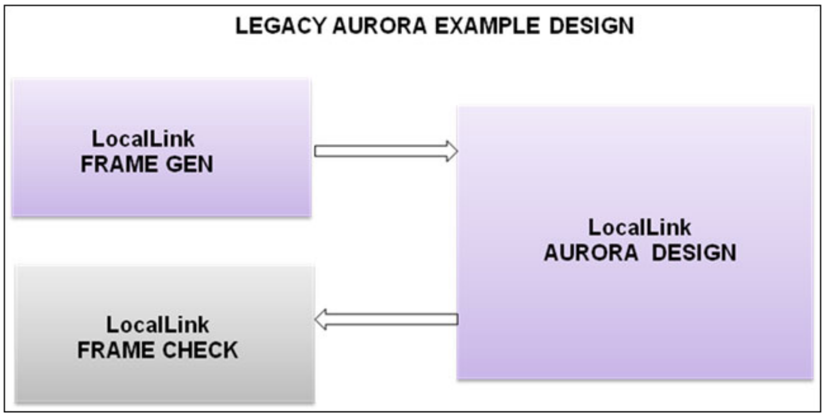


Figure D-1: **Legacy Aurora Example Design**

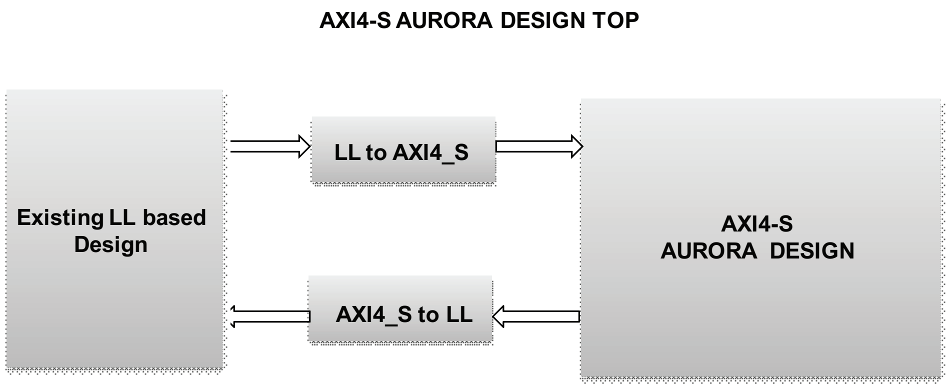


Figure D-2: **AXI4-Stream Aurora Example Design**

Migration Steps

Generate an AXI4-Stream Aurora core from the CORE Generator™ tool using the ISE® tool v13.1 release.

Simulate the Core

1. Run the `vsim -do simulate_mti.do` file from the `/simulation/functional` directory.
2. ModelSim GUI launches and compiles the modules.
3. The `wave_mti.do` file loads automatically and populates AXI4-Stream signals.
4. Allow the simulation to run. This might take some time.
   a. Initially lane up is asserted.
   b. Channel up is then asserted and the data transfer begins.
   c. Data transfer from all flow control interfaces now begins.
   d. Frame checker continuously checks the received data and reports for any data mismatch.
5. A 'TEST PASS' or 'TEST FAIL' status is printed on the ModelSim console providing the status of the test.

Implement the Core

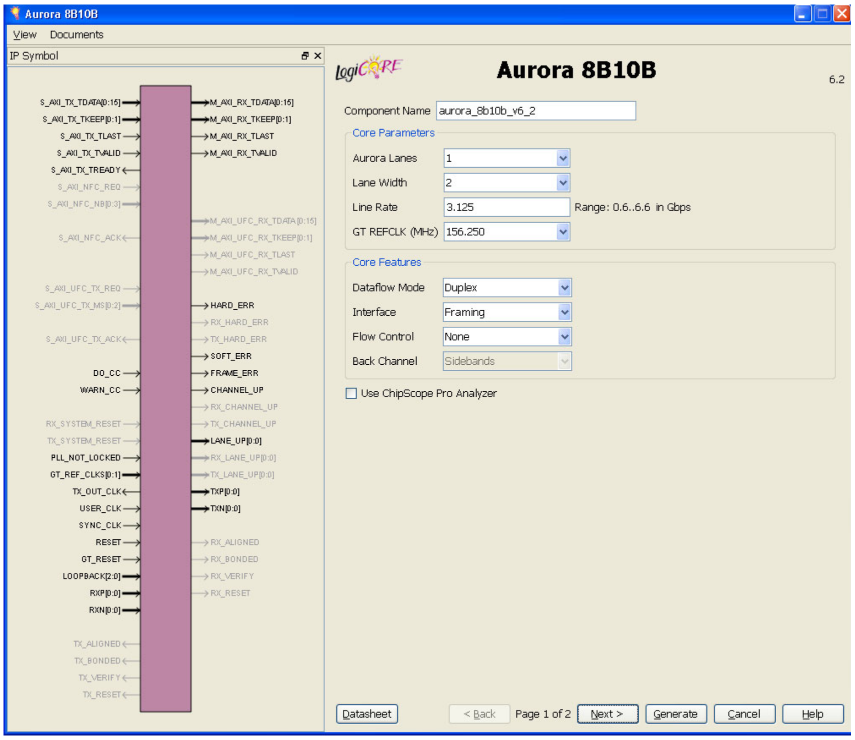
1. Run `./implement.sh` (for Linux) from the `/implement` directory.
2. The implement script compiles the core, runs through the ISE tool, and generates a bit file and netlist for the core.

Integrate to an Existing LocalLink-based Aurora Design

1. The Aurora core provides a light-weight 'shim' to interface to any existing LL based interface. The shims are delivered along with the core from aurora_8b10b_v6_2 version of the core.
2. See Figure D-2, page 100 for the emulation of an LL Aurora core from an AXI4-Stream Aurora core.
3. Two shims `<component name>_ll_to_axi.v[hd]` and `<component name>_axi_to_ll.v[hd]` are provided in the `src` directory of the AXI4-Stream Aurora core.
4. Instantiate both the shims along with `<component name>.v[hd]` in the existing LL based design top.
5. Connect the shim and AXI4-Stream Aurora design as shown in Figure D-2, page 100.
6. The latest AXI4-Stream Aurora core can be plugged into any existing LL design environment.

GUI Changes

Figure D-3 shows the AXI4-Stream signals in the IP Symbol diagram.



Aurora 8B10B 6.2

Component Name: aurora_8b10b_v6_2

Core Parameters

Aurora Lanes: 1
Lane Width: 2
Line Rate: 3.125 Range: 0.6..6.6 in Gbps
GT REFCLK (MHz): 156.250

Core Features

Dataflow Mode: Duplex
Interface: Framing
Flow Control: None
Back Channel: Sidebands

☐ Use ChipScope Pro Analyzer

Figure D-3: **AXI4-Stream Signals**

Limitations

This section outlines the limitations of the Aurora 8B/10B core for AXI4-Stream support. The user has to take care of these limitations while interfacing the Aurora 8B/10B core with the AXI4-Stream compliant interface core.

Limitation 1:

The AXI4-Stream specification supports four types of data stream:

1. Byte stream
2. Continuous aligned stream
3. Continuous unaligned stream
4. Sparse stream

The Aurora 8B/10B core supports only continuous aligned stream and continuous unaligned stream. The position bytes are valid only at the end of packet.

Limitation 2:

The AXI4-Stream protocol supports transfer with zero data at the end of packet, but the Aurora 8B/10B core expects at least one byte should be valid at the end of packet.