LogiCORE IP
Aurora 8B/10B v8.1

User Guide

UG766 April 24, 2012
Notice of Disclaimer

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available “AS IS” and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.

© Copyright 2010–2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, ARM, ARM1176JZ-S, CoreSight, Cortex, and PrimeCell are trademarks of ARM in the EU and other countries. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/21/10</td>
<td>1.0</td>
<td>First release of the core with AXI interface support. The previous release of this document was UG353.</td>
</tr>
<tr>
<td>03/01/11</td>
<td>1.1</td>
<td>ISE 13.1 tool release with v6.2 core. Removed references to Virtex-5 devices, including Appendix. Changed &lt;component_name&gt; to &lt;component name&gt; 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 &lt;component name&gt;.xise to Table 11-1, &lt;component name&gt;.v[hd] to Table 11-4, and implement_planahead.tcl to Table 11-9, simulate_isim.bat, simulate_isim.sh, and wave_isim.tcl to Table 11-11. In Table 11-11, changed the name of mti_wave.do to wave_mti.do. Added note on page 91 to Appendix D. Added step 9 to Case 2: Virtex-6 FPGA GTX Wrapper. Added step 11 to Case 3: Spartan-6 FPGA GTP Wrapper.</td>
</tr>
<tr>
<td>Date</td>
<td>Version</td>
<td>Revision</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| 10/19/11   | 1.2     | ISE v13.3 tool release with v7.1 core. Added support for Virtex-7 and Kintex-7 FPGAs. Removed references to RX/TX Simplex operation. Removed Preface.  
  • Chapter 1: Added *List of Acronyms.*  
  • Chapter 3: Changed valid range in *Aurora Lanes,* page 13. Updated *Lane Width,* page 13.  
  • Chapter 4: Updated *Figure 4-8* and associated note.  
  • Chapter 6: Added CPLLLOCKDETCLK_IN to Table 6-1.  
  • Chapter 8: Added text about simulating simplex cores at the end of *Simulating the Example Design,* page 73.  
  • Added *Chapter 10, Optional Features.*  
  • Chapter 11: Added Aurora core top-level module to Table 11-1. Changed name of README file in Table 11-2. Added version information file to Table 11-3. Added NGC and script files to Table 11-9. Added Cadence IES simulator scripts to Table 11-11.  
  • Added *Appendix A, Additional Resources.*  
  • Appendix E: Moved and updated the *Limitations* section. |
| 04/24/12   | 1.3     | ISE v14.1 tool release with v8.1 core. Updated GUI screen captures. Updated component, UCF, example design, and test bench names.  
  • Chapter 2: Added *Use Scrambler/Descrambler.*  
  • Chapter 4: Added M_AXI_RX_SNF and M_AXI_RX_FC_NB[0:3] (NFC ports).  
  • Chapter 6: Removed CPLLLOCKDETCLK_IN from Table 6-1.  
  • Chapter 7: Added third paragraph about hot-plug logic to *Introduction.*  
  • Renamed *Chapter 10, Optional Features* and added *Using the Scrambler/Descrambler* section.  
  • Appendix E: Added statements about TKEEP to *Limitations.* |
# Table of Contents

Revision History ................................................................. 2

Chapter 1: Introduction
   About the Core ........................................................... 9
   Supported Tools and System Requirements ......................... 9
   Recommended Design Experience .................................... 10
   Technical Support ......................................................... 10
   Feedback ................................................................. 10

Chapter 2: Customizing the Aurora 8B/10B Core
   Introduction ............................................................. 11
   Using the IP Customizer ............................................... 11
   Using the Build Script ............................................... 15
   Designing with the Core ............................................. 16

Chapter 3: User Interface
   Introduction ............................................................. 19
   Top-Level Architecture ............................................... 19
   Framing Interface ..................................................... 22
   Streaming Interface .................................................. 31

Chapter 4: Flow Control
   Introduction ............................................................. 35
   Native Flow Control .................................................. 36
   User Flow Control ..................................................... 39

Chapter 5: Status, Control, and the Transceiver Interface
   Introduction ............................................................. 45
   Full-Duplex Cores ...................................................... 46
   Simplex Cores .......................................................... 49
   Reset and Power Down ................................................ 54

Chapter 6: Clock Interface and Clocking
   Introduction ............................................................. 57
   Clock Interface Ports for GTP/GTX Transceiver Cores ............. 59
   Clocking from a Neighboring GTX_QUAD Tile for Virtex-6 FPGA Designs ........................................................... 59
   Reference Clocks for Spartan-6 FPGA GTP Transceiver Designs ........................................................... 62
   Clock Rates for GTP/GTX Transceiver Designs ....................... 63
Chapter 7: Clock Compensation
Introduction ................................................................. 65
Clock Compensation Interface ........................................... 66

Chapter 8: Quick Start Example Design
Overview ................................................................. 71
Generating the Core .................................................. 71
Simulating the Example Design ........................................ 73
Implementing the Example Design .................................... 74

Chapter 9: Example Design Overview
Introduction ................................................................. 75

Chapter 10: Optional Features
Using the Scrambler/Descrambler .................................... 77
Using ChipScope Pro Analyzer Cores .............................. 77

Chapter 11: Project Directory Structure
Directory and File Structure ............................................ 79
Directory and File Contents ............................................ 80

Appendix A: Additional Resources
Xilinx Resources ............................................................. 85
Solution Centers ............................................................ 85
References ................................................................. 85
Additional Core Resources .............................................. 85

Appendix B: Handling Timing Errors Due To Far Apart Transceiver Selection
Example Solutions ............................................................. 87

Appendix C: Performance and Core Latency
Introduction ................................................................. 89
Latency of the Frame Path ................................................. 89

Appendix D: Generating a Wrapper File from the Transceiver Wizard
Case 1: Virtex-7/Kintex-7 FPGA Wrapper Compatibility ............. 92
Case 2: Virtex-6 FPGA GTX Wrapper .................................. 93
Case 3: Spartan-6 FPGA GTP Wrapper .................................. 94
Appendix E: Aurora AXI4-Stream Migration Guide

- Introduction ......................................................... 95
- Overview of Major Changes ................................. 95
- Block Diagram ...................................................... 96
- Migration Steps .................................................... 97
- GUI Changes ....................................................... 98
Chapter 1

Introduction

This user guide provides information for generating a LogiCORE™ IP Aurora 8B/10B core using Virtex®-7/Kintex™-7 FPGA GTX transceivers, Virtex-6 FPGA GTX transceivers and Spartan®-6 FPGA GTP transceivers. The core implements the Aurora 8B/10B protocol using the high-speed serial transceivers on the Virtex-7/Kintex-7 families (including lower power), Virtex-6 LXT, SXT, CXT, HXT families (including lower power) and the Spartan-6 LXT family. The LogiCORE IP Aurora 8B/10B v8.1 core supports the AMBA® protocol AXI4-Stream user interface.

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, using Virtex-7/Kintex-7 FPGA GTX transceivers, Virtex-6 FPGA GTX transceivers, and Spartan-6 FPGA GTP transceivers. The core is delivered as open-source code and supports both Verilog and VHDL design environments. Each core comes with an example design and supporting modules.

About the Core

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

An IP license is not required to use the Aurora 8B/10B core for Xilinx® FPGAs.

Supported Tools and System Requirements

Operating System Requirements

For a list of system requirements, see the ISE Design Suite 14: Release Notes Guide.

Tools

For the supported versions of the tools, see the ISE Design Suite 14: Release Notes Guide.

- ISE® tool v14.1
- ISim v14.1
- PlanAhead™ tool
- Mentor Graphics ModelSim
- Cadence Incisive Enterprise Simulator (IES)
- Synopsys Synplify Pro
Chapter 1: Introduction

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 Transceiver Interface, carefully.

Consult the PCB design requirements information in:
- 7 Series FPGAs GTX Transceivers User Guide
- 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.

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 v8.1 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™ tool. This chapter details the available customization parameters and how these parameters are specified within the IP Customizer interface.

Using the IP Customizer

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

IP Customizer

Figure 2-1 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 12 for Virtex®-7/Kintex™-7/Virtex-6 FPGA GTX transceivers
- Figure 2-3, page 13 for Spartan®-6 FPGA GTP transceivers
Chapter 2: Customizing the Aurora 8B/10B Core

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

Figure 2-2: Second GUI Page for Virtex-7/Kintex-7/Virtex-6 FPGA GTX 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. The top-level module for the core uses `<Component Name>_exdes`.

Default: aurora_8b10b_v8_1

Lane Assignment

Refer to the diagram in the information area in Figure 2-2 and Figure 2-3. Each row represents a GTPA1_DUAL tile in Spartan-6 FPGAs or two columns represent a GTX Quad in Virtex-7, Kintex-7, and 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 is from 1 to 16 and depends on the target device selected.

Default: 1

Lane Width

Select the byte width of the GTP/GTX transceivers used in the core. This parameter defines the TXDATA/RXDATA width of the transceiver and the user interface data bus width as well. Valid values are 2 and 4.

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

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, and Duplex. See Chapter 5, Status, Control, and the Transceiver 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 \((0.8 \times \text{line rate}) \times \text{Aurora 8B/10B lanes}\).

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-7/Kintex-7/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, Virtex-7, and Kintex-7 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 Scrambler/Descrambler**

Select to include the 16-bit additive scrambler/descrambler to the Aurora 8B/10B design. See Chapter 10, Optional Features, for more information

Default: Unchecked

**Use ChipScope Pro Analyzer**

Select to add ChipScope™ Pro cores to the Aurora 8B/10B core (see Using ChipScope Pro Analyzer Cores in Chapter 10). This option provides 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 tool project directory using the same name as the top level of the core. See Chapter 11, 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. 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 tool 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 shows Aurora 8B/10B top level for a duplex configuration. The top-level file is the starting point for a user design.
Chapter 3: User Interface

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 3-1: 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™ tool.
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.

Framing TX Ports

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 and simplex RX framing cores.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M AXI RX TDATA[0:8(n-1)]</td>
<td>Output</td>
<td>Incoming data from channel partner (Ascending bit order).</td>
</tr>
<tr>
<td>M AXI RX TLAST</td>
<td>Output</td>
<td>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).</td>
</tr>
<tr>
<td>M AXI RX TKEEP[0:(n-1)]</td>
<td>Output</td>
<td>Specifies the number of valid bytes in the last data beat; valid only when M AXI RX TLAST is asserted.</td>
</tr>
<tr>
<td>M AXI RX TVALID</td>
<td>Output</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

![AXI4-Stream Interface Bit Ordering](image-url)
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>
<thead>
<tr>
<th>Table 3-3: Typical Channel Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>/SCP/1</td>
</tr>
</tbody>
</table>
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 \( 3n \) 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 \( S_{AXI \_TX \_TVALID} \) 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 \( S_{AXI \_TX \_TLAST} \), 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](image)
Example B: Data Transfer with Pad

Figure 3-6 shows an example of a \((3n-1)\)-byte data transfer that requires the use of a pad. 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 \(3n-1\) data bytes requires two full \(n\)-byte data words and one partial data word. In this example, \(S_{\text{AXI TX TKEEP}}\) is set to N-1 to indicate \(n\)-1 valid bytes in the last data word.

Example C: Data Transfer with Pause

Figure 3-7 shows how a user can pause data transmission during a frame transfer. In this example, the user is sending \(3n\) bytes of data, and pauses the data flow after the first \(n\) bytes. After the first data word, the user deasserts \(S_{\text{AXI TX TVALID}}\), 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 \(S_{\text{AXI TX TVALID}}\) is deasserted.
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 sequence.

![Figure 3-8: Data Transfer Paused by Clock Compensation](image)

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 datapath 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 3n bytes of received data interrupted by a pause. Data is presented on the M_AXIS_RX_TDATA bus. When the first n bytes are placed on the bus, M_AXIS_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_AXIS_RX_TVALID, indicating to the user that there is a pause in the data flow.

After the pause, the core asserts M_AXIS_RX_TVALID and continues to assemble the remaining data on the M_AXIS_RX_TDATA bus. At the end of the frame, the core asserts M_AXIS_RX_TLAST. The core also computes the value of M_AXIS_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 datapath

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.

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

**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

<table>
<thead>
<tr>
<th>User Data Bytes</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>92.92%</td>
</tr>
<tr>
<td>1,000</td>
<td>99.14%</td>
</tr>
<tr>
<td>10,000</td>
<td>99.81%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Lane</th>
<th>Clock</th>
<th>Function</th>
<th>Character or Data Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Byte 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Start of channel frame</td>
<td>/SCP/1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Channel frame data</td>
<td>D0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Channel frame data</td>
<td>D2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Channel frame data</td>
<td>D4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>33</td>
<td>Channel frame data</td>
<td>D254</td>
</tr>
<tr>
<td>1</td>
<td>33</td>
<td>Transmit idles</td>
<td>/I/</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>Transmit idles</td>
<td>/I/</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>End of channel frame</td>
<td>/ECP/1</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>S_AXI_TX_TKEEP Bus Value (in Binary)</th>
<th>SCP</th>
<th>Pad</th>
<th>ECP</th>
<th>Idles</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000_0000</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>1100_0000</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>1110_0000</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1111_0000</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>1111_1000</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1111_1100</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1111_1110</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1111_1111</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Streaming Interface

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

![Streaming Interface Diagram](image)

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 and simplex TX framing cores.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_AXI_TX_TDATA[0:(8n-1)]</td>
<td>Input</td>
<td>Outgoing data (ascending bit order).</td>
</tr>
<tr>
<td>S_AXI_TX_TREADY</td>
<td>Output</td>
<td>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.</td>
</tr>
<tr>
<td>S_AXI_TX_TVALID</td>
<td>Input</td>
<td>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.</td>
</tr>
</tbody>
</table>
Streaming RX Ports

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_AXI_RX_TDATA[0:(8n-1)]</td>
<td>Output</td>
<td>Incoming data from channel partner (Ascending bit order).</td>
</tr>
<tr>
<td>M_AXI_RX_TVALID</td>
<td>Output</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 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**
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

<table>
<thead>
<tr>
<th>S_AXI_NFC_NB</th>
<th>Idle Cycles Requested</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0 (XON)</td>
</tr>
<tr>
<td>0001</td>
<td>2</td>
</tr>
<tr>
<td>0010</td>
<td>4</td>
</tr>
<tr>
<td>0011</td>
<td>8</td>
</tr>
<tr>
<td>0100</td>
<td>16</td>
</tr>
<tr>
<td>0101</td>
<td>32</td>
</tr>
<tr>
<td>0110</td>
<td>64</td>
</tr>
<tr>
<td>0111</td>
<td>128</td>
</tr>
<tr>
<td>1000</td>
<td>256</td>
</tr>
<tr>
<td>1001 to 1110</td>
<td>Reserved</td>
</tr>
<tr>
<td>1111</td>
<td>Infinite (XOFF)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_AXI_NFC_ACK</td>
<td>Output</td>
<td>Asserted when an Aurora 8B/10B core accepts an NFC request (active-High).</td>
</tr>
<tr>
<td>S_AXI_NFC_NB[0:3]</td>
<td>Input</td>
<td>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.</td>
</tr>
<tr>
<td>S_AXI_NFC_REQ</td>
<td>Input</td>
<td>Asserted to request an NFC message be sent to the channel partner (active-High). Must be held until S_AXI_NFC_ACK is asserted.</td>
</tr>
<tr>
<td>M_AXI_RX_SNF</td>
<td>Output</td>
<td>Indicates an NFC message is received from the partner. This port is asserted for one USER_CLK cycle.</td>
</tr>
<tr>
<td>M_AXI_RX_FC_NB[0:3]</td>
<td>Output</td>
<td>Indicates the PAUSE value of the received NFC message. This port should be sampled with M_AXI_RX_SNF.</td>
</tr>
</tbody>
</table>

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 several 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. $S_{AXI\_TX\_TREADY}$ 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.

![Transmitting an NFC Message](image)

*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

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_AXI_UFC_TX_REQ</td>
<td>Input</td>
<td>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 after it has started.</td>
</tr>
<tr>
<td>S_AXI_UFC_TX_MS[0:2]</td>
<td>Input</td>
<td>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 40.</td>
</tr>
<tr>
<td>S_AXI_UFC_TX_ACK</td>
<td>Output</td>
<td>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.</td>
</tr>
<tr>
<td>M_AXI_UFC_RX_TDATA[0:(8n-1)]</td>
<td>Output</td>
<td>Incoming UFC message data from the channel partner (n = 16 bytes maximum).</td>
</tr>
<tr>
<td>M_AXI_UFC_RX_TVALID</td>
<td>Output</td>
<td>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).</td>
</tr>
<tr>
<td>M_AXI_UFC_RX_TLAST</td>
<td>Output</td>
<td>Signals the end of the incoming UFC message (active-High).</td>
</tr>
<tr>
<td>M_AXI_UFC_RX_TKEEP[0:(n-1)]</td>
<td>Output</td>
<td>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).</td>
</tr>
</tbody>
</table>
Chapter 4: Flow Control

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**

<table>
<thead>
<tr>
<th>SIZE Field Contents</th>
<th>UFC Message Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2 bytes</td>
</tr>
<tr>
<td>001</td>
<td>4 bytes</td>
</tr>
<tr>
<td>010</td>
<td>6 bytes</td>
</tr>
<tr>
<td>011</td>
<td>8 bytes</td>
</tr>
<tr>
<td>100</td>
<td>10 bytes</td>
</tr>
<tr>
<td>101</td>
<td>12 bytes</td>
</tr>
<tr>
<td>110</td>
<td>14 bytes</td>
</tr>
<tr>
<td>111</td>
<td>16 bytes</td>
</tr>
</tbody>
</table>

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 41 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

<table>
<thead>
<tr>
<th>UFC Message</th>
<th>S_AXI_UFC_TX_MS Value</th>
<th>AXI4 Interface Width</th>
<th>Number of Data Beats</th>
<th>AXI4 Interface Width</th>
<th>Number of Data Beats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2 Bytes</td>
<td>1</td>
<td>10 Bytes</td>
<td>1</td>
</tr>
<tr>
<td>2 Bytes</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Bytes</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Bytes</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 Bytes</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 Bytes</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 Bytes</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 Bytes</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 Bytes</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 Bytes</td>
<td>2</td>
<td>12 Bytes</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 Bytes</td>
<td>3</td>
<td>14 Bytes</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 Bytes</td>
<td>4</td>
<td>16 Bytes or more</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Bytes</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 Bytes</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 Bytes</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 Bytes</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 Bytes</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 Bytes</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 Bytes</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 Bytes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Bytes</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 Bytes</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 Bytes</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 Bytes</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 Bytes</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 Bytes</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 Bytes</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 Bytes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

Example B: Transmitting a Multicycle UFC Message

The procedure for transmitting a two-cycle UFC message is shown in Figure 4-6. In this case the user application is sending a 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 4-5: Transmitting a Single-Cycle UFC Message

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.

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 four bytes of the data are valid.
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 Transceiver 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.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANNEL_UP</td>
<td>Output</td>
<td>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.</td>
</tr>
<tr>
<td>LANE_UP[0:m-1]</td>
<td>Output</td>
<td>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.</td>
</tr>
<tr>
<td>FRAME_ERR</td>
<td>Output</td>
<td>Channel frame/protocol error detected. This port is active-High and is asserted for a single clock.</td>
</tr>
<tr>
<td>HARD_ERR</td>
<td>Output</td>
<td>Hard error detected. (Active High, asserted until Aurora 8B/10B core resets). See Error Signals in Full-Duplex Cores, page 47 for more details.</td>
</tr>
<tr>
<td>POWER_DOWN</td>
<td>Input</td>
<td>Drives the power-down input of the GTP/GTX transceiver (active-High).</td>
</tr>
<tr>
<td>RESET</td>
<td>Input</td>
<td>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.</td>
</tr>
</tbody>
</table>
Full-Duplex Cores

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 \texttt{SOFT\_ERR} signal on every cycle they are detected.

The core also monitors each GTP/GTX transceiver for hardware errors such as buffer overflow/underflow and loss of lock. The core reports hardware errors by asserting the \texttt{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 \textit{Aurora 8B/10B Protocol Specification} to detect large numbers of soft errors occurring in a short period of time, and asserts the \texttt{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

---

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOFT_ERR</td>
<td>Output</td>
<td>Soft error detected in the incoming serial stream. See Error Signals in Full-Duplex Cores, page 47 for more details. (Active-High, asserted for a single clock).</td>
</tr>
<tr>
<td>RXP[0:m-1]</td>
<td>Input</td>
<td>Positive differential serial data input pin.</td>
</tr>
<tr>
<td>RXN[0:m-1]</td>
<td>Input</td>
<td>Negative differential serial data input pin.</td>
</tr>
<tr>
<td>TXP[0:m-1]</td>
<td>Output</td>
<td>Positive differential serial data output pin.</td>
</tr>
<tr>
<td>TXN[0:m-1]</td>
<td>Output</td>
<td>Negative differential serial data output pin.</td>
</tr>
<tr>
<td>GT_RESET</td>
<td>Input</td>
<td>The reset signal for the PMA modules in the transceivers is connected to the top level through a debouncer. The GT_RESET port 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 INIT_CLK. See the Reset section in the respective transceiver user guide for further details.</td>
</tr>
<tr>
<td>INIT_CLK</td>
<td>Input</td>
<td>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 &lt;component name&gt;_exdes.ucf. Users need to update this clock constraint with respect to their board clock frequency.</td>
</tr>
</tbody>
</table>

Notes:
1. \( m \) is the number of GTP/GTX transceivers
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**

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXAligned</td>
<td>Input</td>
<td>Asserted when RX channel partner has completed lane initialization for all lanes. Typically connected to RXAligned.</td>
</tr>
<tr>
<td>TXBonded</td>
<td>Input</td>
<td>Asserted when RX channel partner has completed channel bonding. Not needed for single-lane channels. Typically connected to RXBonded.</td>
</tr>
<tr>
<td>TXVerify</td>
<td>Input</td>
<td>Asserted when RX channel partner has completed verification. Typically connected to RXVerify.</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX_RESET</td>
<td>Input</td>
<td>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.</td>
</tr>
<tr>
<td>TX_CHANNEL_UP</td>
<td>Output</td>
<td>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.</td>
</tr>
<tr>
<td>TX_LANE_UP[0:m-1]</td>
<td>Output</td>
<td>Asserted for each lane upon successful lane initialization, with each bit representing one lane (active-High).</td>
</tr>
<tr>
<td>TX_HARD_ERR</td>
<td>Output</td>
<td>Hard error detected. (Active-High, asserted until Aurora 8B/10B core resets). See Error Signals in Simplex Cores, page 52 for more details.</td>
</tr>
<tr>
<td>POWER_DOWN</td>
<td>Input</td>
<td>Drives the powerdown input of the GTP/GTX transceiver (active-High).</td>
</tr>
<tr>
<td>TX_SYSTEM_RESET</td>
<td>Input</td>
<td>Resets the Aurora 8B/10B core (active-High).</td>
</tr>
<tr>
<td>T XP[0:m-1]</td>
<td>Output</td>
<td>Positive differential serial data output pin.</td>
</tr>
<tr>
<td>T X N[0:m-1]</td>
<td>Output</td>
<td>Negative differential serial data output pin.</td>
</tr>
</tbody>
</table>

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 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX_ALIGNED</td>
<td>Output</td>
<td>Asserted when RX module has completed lane initialization. Typically connected to TX_ALIGNED.</td>
</tr>
<tr>
<td>RX_BONDED</td>
<td>Output</td>
<td>Asserted when RX module has completed channel bonding. Not used for single-lane channels. Typically connected to TX_BONDED.</td>
</tr>
<tr>
<td>RX_VERIFY</td>
<td>Output</td>
<td>Asserted when RX module has completed verification. Typically connected to TX_VERIFY.</td>
</tr>
<tr>
<td>RX_RESET</td>
<td>Output</td>
<td>Asserted when the RX module needs the TX module to restart initialization. Typically connected to TX_RESET.</td>
</tr>
<tr>
<td>RX_CHANNEL_UP</td>
<td>Output</td>
<td>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.</td>
</tr>
<tr>
<td>RX_LANE_UP[0:m-1]</td>
<td>Output</td>
<td>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.</td>
</tr>
<tr>
<td>FRAME_ERR</td>
<td>Output</td>
<td>Channel frame/protocol error detected. This port is active-High and is asserted for a single clock.</td>
</tr>
<tr>
<td>RX_HARD_ERR</td>
<td>Output</td>
<td>Hard error detected. (Active-High, asserted until Aurora 8B/10B core resets). See Error Signals in Simplex Cores, page 52 for more details.</td>
</tr>
<tr>
<td>POWER_DOWN</td>
<td>Input</td>
<td>Drives the power-down input of the GTP/GTX transceiver (active-High).</td>
</tr>
<tr>
<td>RX_SYSTEM_RESET</td>
<td>Input</td>
<td>Resets the Aurora 8B/10B core (active-High).</td>
</tr>
<tr>
<td>SOFT_ERR</td>
<td>Output</td>
<td>Soft error detected in the incoming serial stream. See Error Signals in Simplex Cores, page 52 for more details. (Active-High, asserted for a single clock).</td>
</tr>
<tr>
<td>RXP[0:m-1]</td>
<td>Input</td>
<td>Positive differential serial data input pin.</td>
</tr>
<tr>
<td>RXN[0:m-1]</td>
<td>Input</td>
<td>Negative differential serial data input pin.</td>
</tr>
</tbody>
</table>

**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.
Error Signals in Simplex Cores

The 8B/10B encoding allows RX simplex 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/underflow 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 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. 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 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 5-5 summarizes the error conditions simplex Aurora 8B/10B cores can detect and the error signals used to alert the user application.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
<th>TX</th>
<th>RX</th>
</tr>
</thead>
<tbody>
<tr>
<td>HARD_ERR</td>
<td>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.</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Bad Control Character: The protocol engine attempts to send a bad control character. This is an indication of design corruption or catastrophic failure.</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Simplex Cores

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.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
<th>TX</th>
<th>RX</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOFT_ERR</td>
<td>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.</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>No Data in Frame: A channel frame is received with no data.</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>FRAME_ERR</td>
<td>Truncated Frame: A channel frame is started without ending the previous channel frame, or a channel frame is ended without being started.</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Invalid Control Character: The protocol engine receives a control character that it does not recognize.</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Invalid UFC Message Length: A UFC message is received with an invalid length.</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

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

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.

No Data in Frame: A channel frame is received with no data.

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.

Invalid UFC Message Length: A UFC message is received with an invalid length.
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 7 Series FPGAs GTX Transceivers User Guide, the Virtex-6 FPGA GTX Transceivers User Guide, or the Spartan-6 FPGA GTX Transceivers User Guide for the device being used for details about powering down GTP/GTX transceivers.
Timing

Figure 5-5 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.

![Diagram of Reset and Power Down Timing]

Figure 5-5: 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 transceivers. It also requires at least one frequency locked parallel clock for synchronous operation with the user application.

The Virtex®-7/Kintex™-7/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-7/Kintex-7/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>
<thead>
<tr>
<th>Clock Ports</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLL_NOT_LOCKED</td>
<td>Input</td>
<td>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.</td>
</tr>
<tr>
<td>USER_CLK</td>
<td>Input</td>
<td>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 outputs of a PLL whose input is derived from TX_OUT_CLK. These clock generations are available in &lt;component name&gt;_clock_module file. The Spartan-6 FPGA uses the GTPCLKOUT port to derive USER_CLK and SYNC_CLK outputs. USER_CLK goes as the TXUSRCLK2 input to the transceiver. See the respective transceiver user guide for more information.</td>
</tr>
<tr>
<td>SYNC_CLK</td>
<td>Input</td>
<td>Parallel clock used by the internal synchronization logic of the GTP/GTX transceivers in the Aurora 8B/10B core. SYNC_CLK goes as the TXUSRCLK input to the transceiver. See the respective transceiver user guide for more information.</td>
</tr>
<tr>
<td>GT_REFCLK</td>
<td>Input</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 61 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.

**Note:** Virtex-7/Kintex-7 FPGA reference clock sharing guidelines are the same as the guidelines for Virtex-6 FPGAs.
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](UG766_07_08_072610)

**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.

![Diagram showing clocking from a neighboring GTPA1_DUAL Tile](UG766_07_07_072610)

**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 tool in response to the line rate and reference clock selections made in the Configuration GUI window for the core. Manual edits of the attributes are not recommended, but are possible using the recommendations in the *7 Series FPGAs GTX Transceivers User Guide*, *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.

Hot-plug logic in Aurora 8B/10B designs with Virtex®-6 FPGAs is based on the received clock compensation characters. If clock compensation characters are not received in a predetermined time, the hot-plug logic resets the core and the transceiver. It is recommended that the clock compensation module be used for Aurora 8B/10B designs with Virtex-6 FPGAs. To disable hot-plug logic, set the ENABLE_HOTPLUG parameter to 0 in the <component name>/<component name>_hotplug.v[hd] module. Hot-plug logic will then repeatedly reset the core when looking for clock compensation characters in the received data.
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO_CC</td>
<td>Input</td>
<td>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.</td>
</tr>
<tr>
<td>WARN_CC</td>
<td>Input</td>
<td>The Aurora 8B/10B core does 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.</td>
</tr>
</tbody>
</table>
Figure 7-2 and Figure 7-3 are waveform diagrams showing how the DO_CC signal works.

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**

<table>
<thead>
<tr>
<th>Lane Width</th>
<th>USER_CLK Cycles Between DO_CC</th>
<th>DO_CC Duration (USER_CLK cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5000</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>3000</td>
<td>3</td>
</tr>
</tbody>
</table>

WARN_CC is for cores with user flow control (UFC) and/or native flow control (NFC). Driving this signal before DO_CC is asserted prevents the UFC interface from acknowledging and sending UFC messages too close to a clock correction sequence. This precaution is necessary because data corruption occurs when CC sequences and UFC messages overlap. The number of lookahead cycles required to prevent a 16-byte UFC message from colliding with a clock compensation sequence depends on the number of lanes in the channel and the width of each lane. Table 7-3 shows the number of lookahead cycles required for each combination of lane width, channel width, and maximum UFC message size.
Native flow control message requests are not acknowledged during assertion of WARN_CC and DO_CC signals. This helps to prevent the collision of an NFC message and the 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™ tool in the cc_manager subdirectory. It automatically generates pulses to create Aurora 8B/10B compliant clock compensation sequences on the DO_CC port and sufficiently early pulses on the WARN_CC port to prevent UFC collisions with maximum-sized UFC messages. This module always be connected to the clock compensation port on the Aurora 8B/10B module, except in special cases. Table 7-4 shows the port description for the standard CC module.

Table 7-3: Lookahead Cycles

<table>
<thead>
<tr>
<th>Data Interface Width</th>
<th>Max UFC Size</th>
<th>WARN_CC Lookahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>2-4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>6-8</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>10-12</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>14-16</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>2-6</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>8-12</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>14-16</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>2-8</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>10-16</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>2-10</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>12-16</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>2-12</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>14-16</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>2-14</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>≥16</td>
<td>2-16</td>
<td>3</td>
</tr>
</tbody>
</table>
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 67) because the reference clock frequencies on both sides of the module are locked. In this case, \texttt{WARN\_CC} and \texttt{DO\_CC} should both be tied to ground. Additionally, the \texttt{CLK\_CORRECT\_USE} attribute can be set to \texttt{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 \texttt{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
- \texttt{DO\_CC} has no effect until after \texttt{CHANNEL\_UP}; \texttt{DO\_CC} should be asserted immediately after \texttt{CHANNEL\_UP} because no clock compensation can occur during initialization

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
Name & Direction & Description \\
\hline
\texttt{WARN\_CC} & Output & Connect this port to the \texttt{WARN\_CC} input of the Aurora 8B/10B core when using UFC. \\
\hline
\texttt{DO\_CC} & Output & Connect this port to the \texttt{DO\_CC} input of the Aurora 8B/10B core. \\
\hline
\texttt{CHANNEL\_UP} & Input & Connect this port to the \texttt{CHANNEL\_UP} output of a full-duplex core, or to the \texttt{TX\_CHANNEL\_UP} output of a simplex TX port. \\
\hline
\end{tabular}
\caption{Standard CC I/O Port}
\end{table}
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 (<component name>_tb). For detailed information about the example design provided with the Aurora 8B/10B core, see Chapter 11, 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 tool from a required directory.
   For help starting and using the CORE Generator tool, see CORE Generator Help in the ISE® tool 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 gray 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 v8.1 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 v8.1 core 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_v8_1.
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 11, 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 v14.1 tool for instructions on how to compile ISE tool simulation libraries.

Table 8-1: Required Simulation Libraries

<table>
<thead>
<tr>
<th>HDL</th>
<th>Library</th>
<th>Source Directories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verilog</td>
<td>UNISIMS_VER</td>
<td>&lt;Xilinx dir&gt;/verilog/src/unisims</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Xilinx dir&gt;/secureip/&lt;SIMULATOR&gt;</td>
</tr>
<tr>
<td>VHDL</td>
<td>UNISIM</td>
<td>&lt;Xilinx dir&gt;/vhdl/src/unisims</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Xilinx dir&gt;/secureip/&lt;SIMULATOR&gt;</td>
</tr>
</tbody>
</table>

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_v8_1/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.

Because cores are generated one at a time, simulating simplex cores requires additional steps. To simulate a simplex TX or simplex RX core, perform these steps:

1. Generate the core for simulation.
2. Generate a complementary simplex core. Go to the implement directory of the first core generated.
3. Set the environment variable SIMPLEX_PARTNER to point to the directory for the complementary core.
4. Run the script according to the instructions in this section.

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

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

For Windows

```
ms-dos> cd aurora_8b10b_v8_1/implement
ms-dos> .\implement.bat
```

For Linux

```
% cd aurora_8b10b_v8_1/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.
Chapter 9

Example Design Overview

Introduction

Each Aurora 8B/10B core includes an example design (<component name>_exdes) that uses the core in a simple data transfer system. For more details about the example_design directory, see Chapter 11, Project Directory Structure.

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

The example designs use 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>_exdes.ucf file in the example_design subdirectory to supply the correct pins and clock constraints.
### Chapter 9: Example Design Overview

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

<table>
<thead>
<tr>
<th>Port</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RXN[0:m-1]</td>
<td>Input</td>
<td>Negative differential serial data input pin.</td>
</tr>
<tr>
<td>RXP[0:m-1]</td>
<td>Input</td>
<td>Positive differential serial data input pin.</td>
</tr>
<tr>
<td>TXN[0:m-1]</td>
<td>Output</td>
<td>Negative differential serial data output pin.</td>
</tr>
<tr>
<td>TXP[0:m-1]</td>
<td>Output</td>
<td>Positive differential serial data output pin.</td>
</tr>
<tr>
<td>ERR_COUNT[0:7]</td>
<td>Output</td>
<td>Count of the number of data words received by the frame checker that did not match the expected value.</td>
</tr>
<tr>
<td>RESET</td>
<td>Input</td>
<td>Reset signal for the example design. The reset is debounced using a USER_CLK signal generated from the reference clock input.</td>
</tr>
<tr>
<td>&lt;reference clock(s)&gt;</td>
<td>Input</td>
<td>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.</td>
</tr>
<tr>
<td>&lt;core error signals&gt;</td>
<td>Output</td>
<td>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 Transceiver Interface for details.</td>
</tr>
<tr>
<td>&lt;core channel up signals&gt;</td>
<td>Output</td>
<td>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 Transceiver Interface for details.</td>
</tr>
<tr>
<td>&lt;core lane up signals&gt;</td>
<td>Output</td>
<td>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 Transceiver Interface for details.</td>
</tr>
<tr>
<td>&lt;simplex initialization signals&gt;</td>
<td>Input/Output</td>
<td>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 Transceiver Interface for details.</td>
</tr>
</tbody>
</table>
Chapter 10

Optional Features

Using the Scrambler/Descrambler

An 16-bit additive scrambler/descrambler, implemented for data, is available in the `<component name>_scrambler.v[hd]` module. It ensures non-occurrence of repetitive data over long periods of time. The scrambler and descrambler are synchronized based on reception of the clock compensation characters. DO_CC must be transmitted to load the seed value of the scrambler and descrambler simultaneously. Thus, the `standard_cc_module` that comes with the Aurora example design should always be used, if the Use Scramble/Descramble option is selected in the GUI.

Using ChipScope Pro Analyzer Cores

The ICON and VIO cores in the ChipScope™ Pro Analyzer help to debug and validate the design in boards. These cores 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>_exdes` module can be set to 1 before running implementation.
Chapter 11

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™ tool 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 tool 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 file for the clock compensation block
    - `/example_design/clock_module`
      Verilog/VHDL design file 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
Chapter 11: Project Directory Structure

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 11-1: project Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;project directory&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;.v[hd]</td>
<td>Aurora core top-level module</td>
</tr>
<tr>
<td>&lt;coregen project filename&gt;.cgp</td>
<td>CORE Generator tool project file</td>
</tr>
<tr>
<td>&lt;component name&gt;.xise</td>
<td>Xilinx® ISE® tool project file</td>
</tr>
</tbody>
</table>

Back to Top

<project directory>/<component name>

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

Table 11-2: component name Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;project directory&gt;/&lt;component name&gt;</td>
<td></td>
</tr>
<tr>
<td>aurora_8b10b_v8_1_readme.txt</td>
<td>Release notes file</td>
</tr>
</tbody>
</table>

Back to Top

<component name>/doc

The doc directory contains the product documentation.

Table 11-3: doc Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;component name&gt;/doc</td>
<td></td>
</tr>
<tr>
<td>aurora_8b10b_v8_1_vinfo.html</td>
<td>Version information file</td>
</tr>
</tbody>
</table>

Back to Top
<component name>/example_design

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

Table 11-4: example_design Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;component name&gt;/example_design</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_exdes.v[hd]</td>
<td>Example design top-level file</td>
</tr>
<tr>
<td>&lt;component name&gt;_exdes.ucf</td>
<td>Aurora 8B/10B example design UCF constraints</td>
</tr>
<tr>
<td>&lt;component name&gt;_reset_logic.v[hd]</td>
<td>Aurora 8B/10B reset logic</td>
</tr>
<tr>
<td>v6_icon.ngc</td>
<td>Virtex-6 family NGC files for the debug cores compatible with the ChipScope™ Pro Analyzer tool</td>
</tr>
<tr>
<td>v6_vio.ngc</td>
<td></td>
</tr>
<tr>
<td>s6_icon.ngc</td>
<td>Spartan®-6 family NGC files for the debug cores compatible with the ChipScope Pro Analyzer tool</td>
</tr>
<tr>
<td>s6_vio.ngc</td>
<td></td>
</tr>
<tr>
<td>v7_icon.ngc</td>
<td>Virtex-7/Kintex-7 FPGA NGC files for the debug cores compatible with the ChipScope™ Pro Analyzer tool</td>
</tr>
<tr>
<td>v7_vio.ngc</td>
<td></td>
</tr>
</tbody>
</table>

Back to Top

/example_design/cc_manager

The cc_manager directory contains the clock compensation source file.

Table 11-5: cc_manager Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;component name&gt;/example_design/cc_manager</td>
<td>Clock compensation module source file</td>
</tr>
</tbody>
</table>

Back to Top

/example_design/clock_module

The clock_module directory contains the clock module source file.

Table 11-6: clock_module Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;component name&gt;/example_design/clock_module</td>
<td>Clock module source file</td>
</tr>
</tbody>
</table>

Back to Top
Chapter 11: Project Directory Structure

/example_design/gt

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;component name&gt;/example_design/gt</td>
<td>Verilog/VHDL wrapper files for the transceiver</td>
</tr>
<tr>
<td>&lt;component name&gt;_gtx.v[hd]</td>
<td>Verilog/VHDL wrapper files for the transceiver</td>
</tr>
<tr>
<td>&lt;component name&gt;_transceiver_wrapper.v[hd]</td>
<td>Verilog/VHDL wrapper files for the transceiver</td>
</tr>
</tbody>
</table>

Table 11-8: traffic_gen_check Directory

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;component name&gt;/example_design/traffic_gen_check</td>
<td>Example design traffic generation and checker files</td>
</tr>
<tr>
<td>&lt;component name&gt;_frame_check.v[hd]</td>
<td>Example design traffic generation and checker files</td>
</tr>
<tr>
<td>&lt;component name&gt;_frame_gen.v[hd]</td>
<td>Example design traffic generation and checker files</td>
</tr>
</tbody>
</table>

<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>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;component name&gt;/implement</td>
<td>Windows batch file that processes the example design through the Xilinx ISE tool flow</td>
</tr>
<tr>
<td>implement.bat</td>
<td>Windows batch file that processes the example design through the Xilinx ISE tool flow</td>
</tr>
<tr>
<td>implement.sh</td>
<td>Linux shell script that processes the example design through the Xilinx ISE tool flow</td>
</tr>
<tr>
<td>xst.scr</td>
<td>XST script file for the example design</td>
</tr>
<tr>
<td>xst.prj</td>
<td>XST project file for the example design</td>
</tr>
<tr>
<td>planAhead_ise.sh</td>
<td>PlanAhead™ script files for the example design using ISE flow for Virtex-6 and Spartan-6 devices</td>
</tr>
<tr>
<td>planAhead_ise.bat</td>
<td>PlanAhead™ script files for the example design using ISE flow for Virtex-6 and Spartan-6 devices</td>
</tr>
<tr>
<td>planAhead_ise.tcl</td>
<td>PlanAhead™ script files for the example design using ISE flow for Virtex-6 and Spartan-6 devices</td>
</tr>
<tr>
<td>synplify_pro.prj</td>
<td>Synplify Pro script files for Aurora 8B/10B example design</td>
</tr>
<tr>
<td>implement_synplify.sh</td>
<td>Synplify Pro script files for Aurora 8B/10B example design</td>
</tr>
<tr>
<td>implement_synplify.bat</td>
<td>Synplify Pro script files for Aurora 8B/10B example design</td>
</tr>
</tbody>
</table>
**<component name>/simulation**

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

*Table 11-10: simulation Directory*

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;component name&gt;/simulation</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_tb.v[hd]</td>
<td>Test bench file for simulating the example design</td>
</tr>
</tbody>
</table>

**/simulation/functional**

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

*Table 11-11: functional Directory*

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;component name&gt;/simulation/functional</td>
<td></td>
</tr>
<tr>
<td>simulate_isim.bat</td>
<td>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.</td>
</tr>
<tr>
<td>simulate_isim.sh</td>
<td>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.</td>
</tr>
<tr>
<td>wave_isim.tcl</td>
<td>ISim macro file that opens a Wave window with top-level signals.</td>
</tr>
<tr>
<td>simulate_mti.do</td>
<td>ModelSim macro files that compile the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.</td>
</tr>
<tr>
<td>simulate_mti.sh</td>
<td>ModelSim macro file that opens a Wave window.</td>
</tr>
<tr>
<td>simulate_mti.bat</td>
<td>ModelSim macro files that compile the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.</td>
</tr>
<tr>
<td>wave_mti.do</td>
<td>ModelSim macro files that compile the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.</td>
</tr>
<tr>
<td>simulate_ncsim.sh</td>
<td>Cadence IES simulator scripts that run functional simulation of the example design.</td>
</tr>
<tr>
<td>simulate_ncsim.bat</td>
<td></td>
</tr>
<tr>
<td>wave_ncsim.sv</td>
<td></td>
</tr>
</tbody>
</table>

Back to Top
Chapter 11: Project Directory Structure

/simulation/timing

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

**Table 11-12: timing Directory**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>simulate_mti.do</td>
<td>ModelSim macro files that compile the post place and route netlist of the example design along with standard delay format (SDF) back annotation then runs timing simulation to completion</td>
</tr>
<tr>
<td>simulate_mti.sh</td>
<td></td>
</tr>
<tr>
<td>simulate_mti.bat</td>
<td></td>
</tr>
</tbody>
</table>

<component name>/src

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

**Table 11-13: src Directory**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;component name&gt;_aurora_lane.v</td>
<td>Aurora 8B/10B source files</td>
</tr>
<tr>
<td>&lt;component name&gt;_aurora_pkg.vhd</td>
<td>(VHDL Only)</td>
</tr>
<tr>
<td>&lt;component name&gt;_axi_to_ll.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_channel_err_detect.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_channel_init_sm.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_chbond_count_dec.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_err_detect.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_global_logic.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_idle_and_ver_gen.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_lane_init_sm.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_ll_to_axi.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_rx_ll.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_rx_ll_pdu_datapath.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_sym_dec.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_sym_gen.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_tx_ll.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_tx_ll_control.v</td>
<td></td>
</tr>
<tr>
<td>&lt;component name&gt;_tx_ll_datapath.v</td>
<td></td>
</tr>
</tbody>
</table>
Appendix A

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

http://www.xilinx.com/support

For a glossary of technical terms used in Xilinx documentation, see:

http://www.xilinx.com/company/terms.htm

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

Prior to generating an Aurora 8B/10B core, users should be familiar with the following:

- **SP002, Aurora 8B/10B Protocol Specification**
- **AMBA AXI4-Stream Protocol Specification**
- **UG476, 7 Series FPGAs GTX Transceivers User Guide**
- **UG366, Virtex-6 FPGA GTX Transceivers User Guide**
- **UG386, Spartan-6 FPGA GTP Transceivers User Guide**
- ISE® tool 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 v8.1 Data Sheet**
- UG058: *Aurora 8B/10B Bus Functional Model User Guide* (Contact: auratoramkt@xilinx.com)
- Aurora 8B/10B Release Notes
- Aurora Solution list – AR#21263
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 datapath width (2/4 bytes) and
- Number of unused transceivers between two selected transceivers

As a result of one or more of these 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® Design Suite.
   a. See the ISE tool documentation for instructions to use smartexplorer.
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 C-1 illustrates the latency of the frame path.

Maximum latency for a 2-byte designs from TX_SOF_N to RX_SOF_N is approximately 52 USER_CLK cycles in simulation.

Maximum latency for a 4-byte designs from TX_SOF_N to RX_SOF_N is approximately 61 USER_CLK cycles in simulation.

The pipeline delays are designed to maintain the clock speed.
Generating a Wrapper File from the Transceiver Wizard

The transceiver attributes play a vital role in the functionality of the Aurora 8B/10B core. Use the most current GTP/GTX Transceiver Wizard to generate the transceiver wrapper file,

Note: Xilinx strongly recommends that you update the transceiver wrapper file in ISE tool releases when the transceiver wizard has been updated but the Aurora core has not.

This appendix provides instructions to generate these transceiver wrapper files:

- Case 1: Virtex-7/Kintex-7 FPGA Wrapper Compatibility, page 92
- Case 2: Virtex-6 FPGA GTX Wrapper, page 93
- Case 3: Spartan-6 FPGA GTP Wrapper, page 94
Appendix D: Generating a Wrapper File from the Transceiver Wizard

Case 1: Virtex-7/Kintex-7 FPGA Wrapper Compatibility

Use these steps to generate the transceiver wrapper file using the 7 Series FPGAs Transceivers Wizard:

1. Using the CORE Generator™ tool, run the latest version of the 7 Series FPGAs Transceivers 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 the number of lane(s) and lane width:
   - Aurora 8B/10B single lane 2 byte
   - Aurora 8B/10B single lane 4 byte
   - Aurora 8B/10B multi lane 2 byte
   - Aurora 8B/10B multi lane 4 byte

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>_gt.v[hd]` file in the `example_design/gt` directory available in the Aurora 8B/10B core with the generated `<component name>_gt.v[hd]` file generated from the 7 Series FPGAs Transceivers Wizard.

The transceiver settings for the Aurora 8B/10B core are up to date now.
Case 2: 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.

The transceiver settings for the Aurora 8B/10B core are up to date now.
Case 3: 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.

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

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® v14.1 tool build containing the Aurora 8B/10B v8.1 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)

Limitations

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

1. The Aurora 8B/10B core supports only continuous aligned streams and continuous unaligned streams. The position bytes are valid only at the end of packet. TKEEP is sampled only at TLAST assertion.
2. The AXI4-Stream protocol supports transfers with zero data at the end of packet, but the Aurora 8B/10B core expects at least one byte to be valid at the end of packet. TKEEP should contain a non-zero value during TLAST assertion.

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 E-1_ shows an example Aurora design using the legacy LocalLink interface. _Figure E-2_ shows an example Aurora design using the AXI-4 Stream interface.
Migration Steps

Generate an AXI4-Stream Aurora core from the CORE Generator™ tool using the ISE® tool v14.1 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 the aura8b10b_v8_1 version of the core.
2. See Figure E-2, page 97 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 E-2, page 97.

6. The latest AXI4-Stream Aurora core can be plugged into any existing LL design environment.

GUI Changes

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

![AXI4-Stream Signals](image)