LogiCORE IP XAUI v10.2

User Guide

UG150 January 18, 2012

XILINX®

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 2004–2012 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. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/30/04	1.1	Initial Xilinx release.
04/28/05	2.0	Updated document to support XAUI core v6.0 and Xilinx software v7.1i.
07/29/05	2.1	Update Virtex®-4 FPGA clock diagrams to reflect v6.0 Patch 1 core.
01/18/06	2.2	Updated document to support XAUI core v6.1 and Xilinx software v8.1i.
07/13/06	2.3	Updated to core version 6.2 and Xilinx software 8.2i.
10/23/06	2.4	Updated to core version 7.0, added support for Virtex-5 FPGAs.
02/15/07	2.5	Updated to core version 7.1 and Xilinx software 9.1i.
08/08/07	2.6	Updated to core version 7.2 and Xilinx software 9.2i.
03/24/08	2.7	Updated to core version 7.3 and Xilinx software 10.1.
09/19/08	2.8	Updated to core version 7.4. Added support for Virtex-5 TXT FPGAs.
04/24/09	2.9	Updated to core version 8.1 and Xilinx software 11.1.
06/24/09	3.0	Updated to core version 8.2 and Xilinx software 11.2. Added Virtex-6 CXT FPGA support.
09/16/09	3.1	Updated to core version 9.1 and Xilinx software 11.3. Added Virtex-6 HXT, Virtex-6 -1L and Spartan®-6 FPGA support.
12/02/09	3.1.1	Documentation fixes; updated Figures 7-4, 7-8, and 7-9.
04/19/10	3.2	Updated to core version 9.2 and Xilinx software 12.1.
03/01/11	3.3	Updated to core version 10.1 and Xilinx software 13.1. Added Virtex-7, Kintex™-7, and 20G Virtex-6 support.

Date	Version	Revision

01/18/12	3.4	Summary of Core Changes
		Updated to core version 10.2 and Xilinx software 13.4.
		Summary of Major Documentation Changes
		• Removed Schedule of Figures, Schedule of Tables, and Preface.
		• Added Appendix A, Additional Resources.
		• Removed List of Acronyms. For the first occurrence of each acronym, spelled out occurrence followed by acronym. Example: Field Programmable Gate Array (FPGA)
		• Added information about Virtex-7 and Kintex-7 FPGA speed grades to Device, Package, and Speed Grade Selection section in Chapter 6.
		• Chapter 7 changes
		  • Added text to the 20G-XAUI subsection in Chapter 7.
		  • Added text to Multiple Core Instances section.
		  • Removed External XGMII Interface: No Transmit Elastic Buffer (Virtex-7 and Kintex-7 FPGAs) section.
		  • Removed External XGMII Interface: No Transmit Elastic Buffer (Virtex-6 FPGAs) section.
		  • Restructured some of the subsections.
		• Removed Post-Implementation Simulation section from Chapter 8.
		• Removed simulation/timing section and Timing subsection from Chapter 10.

Table of Contents

Chapter 5: Interfacing to the Core

Chapter 6: Constraining the Core

Chapter 7: Design Considerations

Chapter 8: Implementing the Core

Chapter 9: Quick Start Example Design

Chapter 10: Detailed Example Design

Appendix A: Additional Resources

Appendix B: Verification and Interoperability

Appendix C: Calculating the DCM/MMCM Phase Shift

Appendix D: Core Latency

Appendix E: Debugging Designs

XILINX®

Chapter 1

Introduction

The eXtended Attachment Unit Interface (XAUI) core is a fully-verified solution design that supports Verilog and VHSIC Hardware Description Language (VHDL). In addition, the example design in this guide is provided in both Verilog and VHDL.

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

System Requirements

For a list of System Requirements, see the ISE Design Suite 13: Release Notes Guide.

About the Core

The XAUI core is a Xilinx® CORE Generator™ Intellectual Property (IP) core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see the XAUI product page.

Licensing the Core

This version of the XAUI IP core does not require a license key. Previous versions of the XAUI IP core released in Integrated Software Environment (ISE®) v11.2 design suite and earlier did require a license key; see the version of the getting started guide for the version of the core you are using for information. The XAUI core is provided under the terms of the Xilinx End User License Agreement.

Recommended Design Experience

Although the XAUI 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 Field Programmable Gate Array (FPGA) designs using Xilinx implementation software and User Constraints File (UCF) is recommended.

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

Additional Core Resources

For detailed information about XAUI technology and updates to the XAUI core, see the following:

Documentation

From the XAUI product page:

- *XAUI Data Sheet*

From the document directory after generating the core:

- *XAUI Release Notes*

XAUI Technology

For information about XAUI technology basics, including features, FAQs, the XAUI device interface, typical applications, specifications, and other important information, see www.xilinx.com/products/ipcenter/XAUI.htm.

Ethernet Specifications

Relevant XAUI IEEE standards, which can be downloaded in PDF format from standards.ieee.org/getieee802/:

- *IEEE Std. 802.3-2008*

Other Information

The 10-Gigabit Ethernet Consortium at the University of New Hampshire Interoperability Lab is an excellent source of information on 10-Gigabit Ethernet technology: www.iol.unh.edu/consortiums/10gec/index.html.

Technical Support

For technical support, visit www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the XAUI core. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the XAUI core and the documentation supplied with the core.

Core

For comments or suggestions about the XAUI core, submit a webcase from www.xilinx.com/support. Be sure to include the following information:

- Product name
- Core version number
- Explanation of your comments

Document

For comments or suggestions about this document, submit a webcase from www.xilinx.com/support. Be sure to include the following information:

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

Chapter 2

Core Architecture

This chapter describes the overall architecture of the XAUI core and also describes the major interfaces to the core.

System Overview

XAUI is a four-lane, 3.125 Gb/s per-lane serial interface. The 20 G-XAUI is also supported in Virtex®-6 devices (-3 speed grades), Kintex™-7 and Virtex-7 devices (-2 and -3 speed grades) using four transceivers at 6.25 Gb/s. Each lane is a differential pair, carrying current mode logic (CML) signaling; the data on each lane is 8B/10B encoded before transmission. Special code groups are used to allow each lane to synchronize at a word boundary and to deskew all four lanes into alignment at the receiving end. The XAUI standard is fully specified in clauses 47 and 48 of the 10-Gigabit Ethernet specification *IEEE Std. 802.3-2008*.

The XAUI standard was initially developed as a means to extend the physical separation possible between Media Access Controller (MAC) and physical-side interface (PHY) components in a 10-Gigabit Ethernet system distributed across a circuit board, and to reduce the number of interface signals in comparison with the Ten Gigabit Ethernet Media Independent Interface (XGMII). Figure 2-1 shows the XAUI core being used to connect to a 10-Gigabit Expansion Pack (XPAK) optical module.

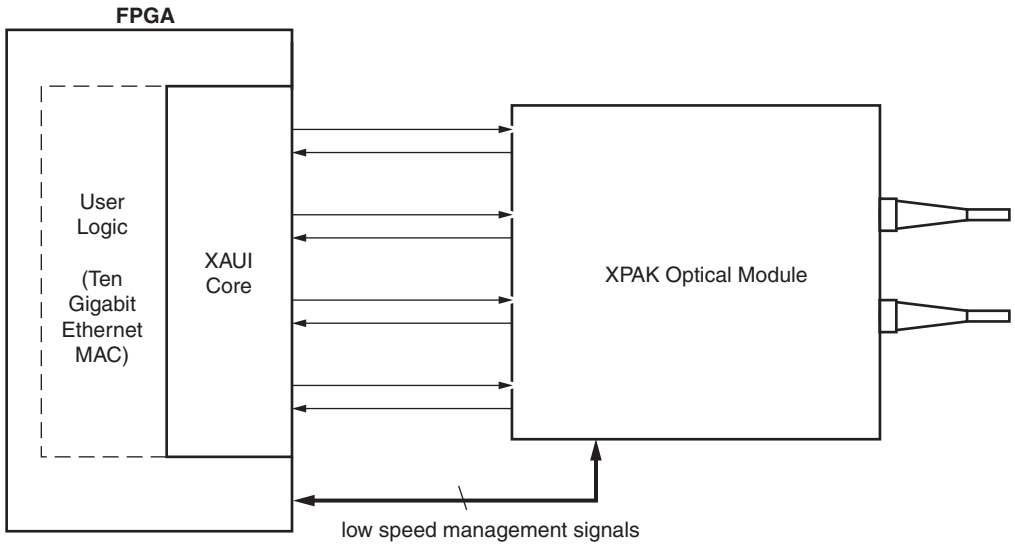

Figure 2-1: **Connecting XAUI to an Optical Module**

After its publication, the applications of XAUI have extended beyond 10-Gigabit Ethernet to the backplane and other general high-speed interconnect applications. A typical backplane application is shown in Figure 2-2.

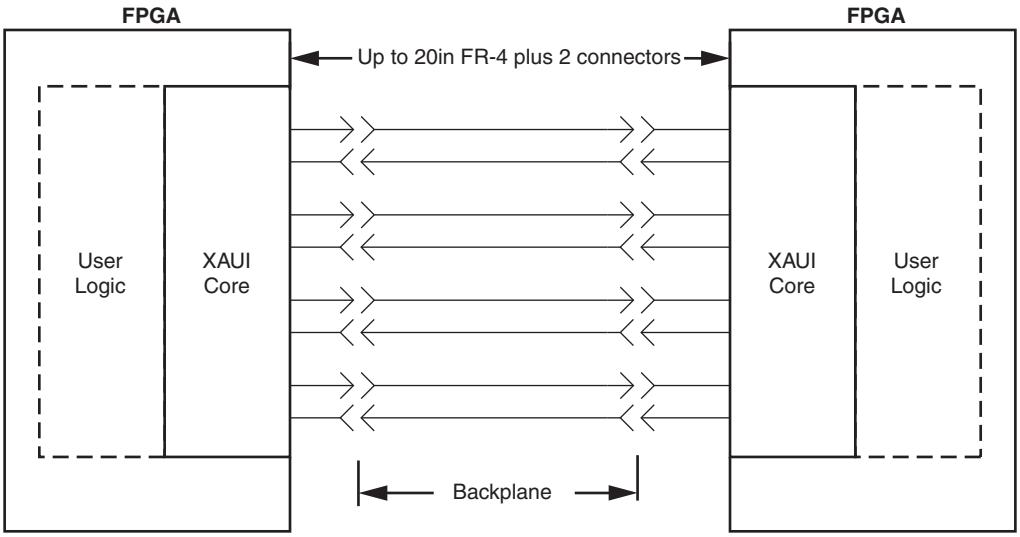

Figure 2-2: **Typical Backplane Application for XAUI**

Functional Description

Figure 2-3 shows a block diagram of the implementation of the XAUI core. The architecture is similar for Virtex®-7, Kintex™-7, Virtex-6, Virtex-5, Virtex-4 and Spartan®-6 FPGAs. The major functional blocks of the core include the following:

- Client-side interface

  If necessary, converts 32-bit Double Data Rate (DDR) data into 64-bit Single Data Rate (SDR) data and crosses clock domain for inbound XGMII data using an elastic buffer.

- Transmit idle generation logic

  Creates the code groups to allow synchronization and alignment at the receiver.

- Synchronization state machine (one per lane)

  Identifies byte boundaries in incoming serial data.

- Deskew state machine

  Deskews the four received lanes into alignment.

- Optional MDIO interface

  A 2-wire low-speed serial interface used to manage the core.

- Embedded Virtex-7 FPGA GTX, Kintex-7 FPGA GTX, Virtex-6 FPGA GTX, Virtex-5 FPGA RocketIO™ GTX transceiver, Virtex-5 FPGA RocketIO GTP transceiver and Virtex-4 FPGA RocketIO Multi-gigabit (MGT) transceivers and Spartan-6 FPGA GTP transceiver

  Provides high-speed transceivers as well as 8B/10B encode and decode, and elastic buffering in the receive datapath.

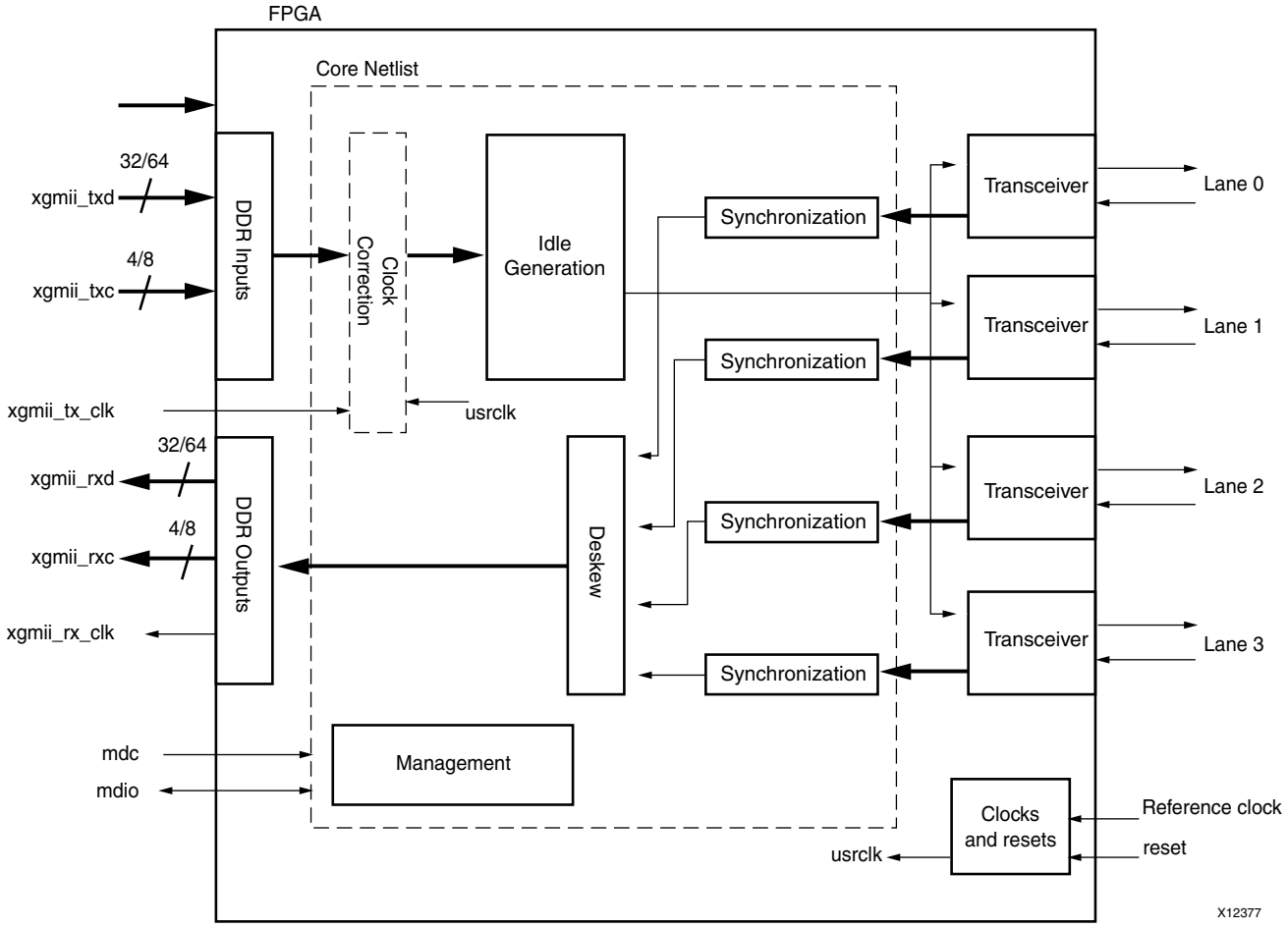

Figure 2-3: **Architecture of the XAUI Core with External XGMII Interface**

A significant number of customer applications do not require an external XGMII interface, but instead add user logic on the client-side interface. This application architecture is shown in Figure 2-4.

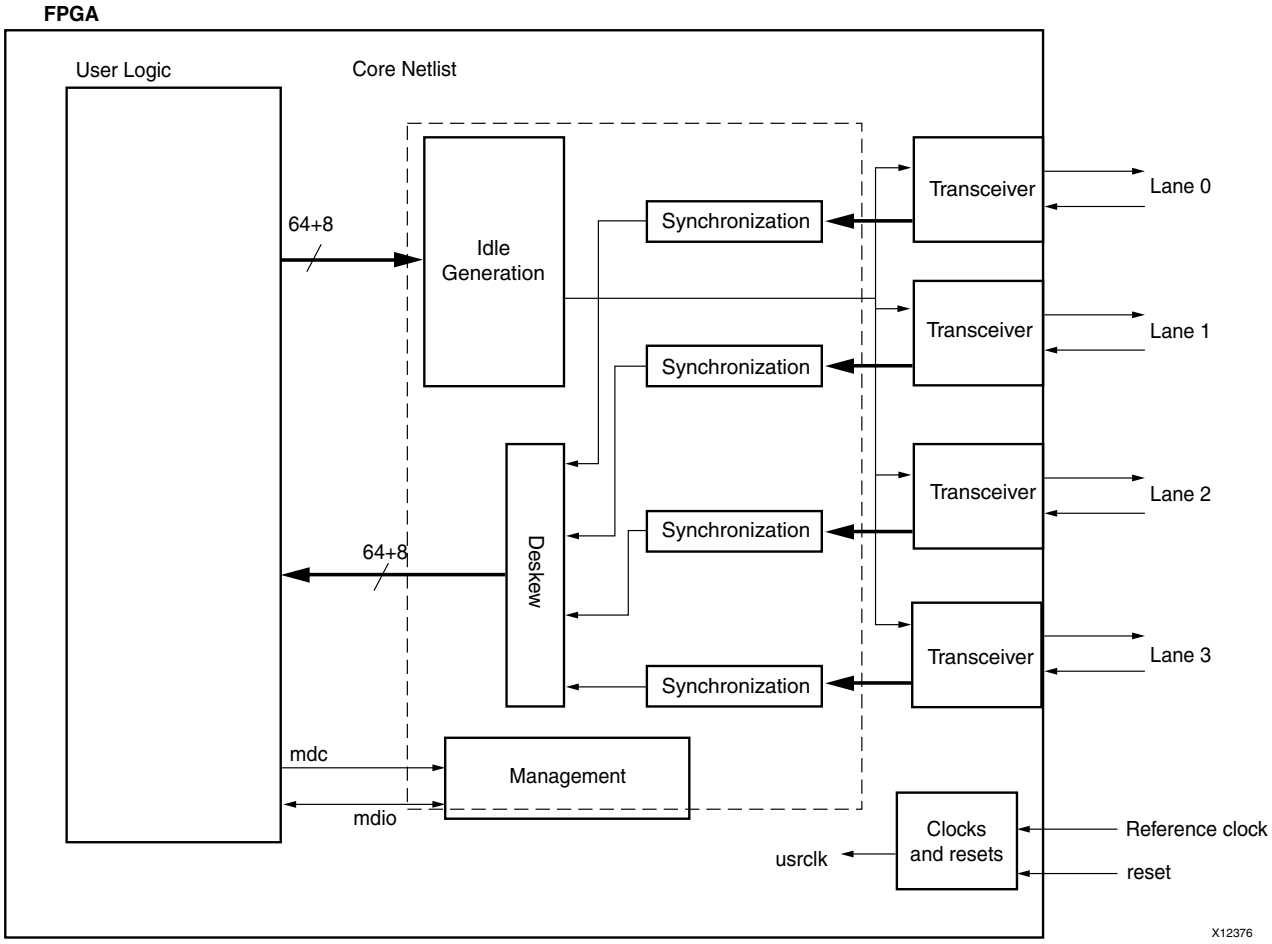

Figure 2-4: **Architecture of the XAUI Core with Client-Side User Logic**

Core Interfaces and Modules

Client-Side Interface

The signals of the client-side interface are shown in Table 2-1. See Chapter 5, Interfacing to the Core for more information on connecting to the client-side interface.

Table 2-1: **Client-Side Interface Ports**

Signal Name	Direction	Description
XGMII_TXD[63:0]	IN	Transmit data, eight bytes wide
XGMII_TXC[7:0]	IN	Transmit control bits, one bit per transmit data byte
TX_CLK	IN	DDR implementations with TX elastic buffer only: Forwarded clock for XGMII_TXD, XGMII_TXC
XGMII_RXD[63:0]	OUT	Received data, eight bytes wide
XGMII_RXC[7:0]	OUT	Receive control bits, one bit per received data byte

Transceiver Interface and Module

The interface to the device-specific transceivers is a simple pin-to-pin interface on those pins that need to be connected. The signals are described in Table 2-2. See Chapter 5, Interfacing to the Core for more information on connecting the device-specific transceivers to the XAUI core.

Table 2-2: **Transceiver Interface Ports**

Signal Name	Direction	Description
MGT_TXDATA[63:0]	OUT	Transceiver transmit data
MGT_TXCHARISK[7:0]	OUT	Transceiver transmit control flag
MGT_RXDATA[63:0]	IN	Transceiver receive data
MGT_RXCHARISK[7:0]	IN	Transceiver receive control signals
MGT_CODEVALID[7:0]	IN	Transceiver receive control signals
MGT_CODECOMMA[7:0]	IN	Transceiver receive control signals
MGT_ENABLE_ALIGN[3:0]	OUT	Transceiver control signals
MGT_ENCHANSYNC	OUT	Transceiver control signal
MGT_SYNCOK[3:0]	IN	Transceiver control signal
MGT_RXLOCK[3:0]	IN	RocketIO™ transceiver control signal. Virtex-4 and Virtex-5 FPGA cores only
MGT_LOOPBACK	OUT	Transceiver control signal
MGT_POWERDOWN	OUT	Transceiver control signal
SIGNAL_DETECT[3:0]	IN	Status signal from attached optical module

MDIO Interface

The MDIO Interface signals are shown in Table 2-3. More information on using this interface can be found in Chapter 5, Interfacing to the Core.

Table 2-3: **MDIO Management Interface Ports**

Signal Name	Direction	Description
MDC	IN	Management clock
MDIO_IN	IN	MDIO input
MDIO_OUT	OUT	MDIO output
MDIO_TRI	OUT	MDIO 3-state; '1' disconnects the output driver from the MDIO bus.
TYPE_SEL[1:0]	IN	Type select
PRTAD[4:0]	IN	MDIO port address; you should set this to provide a unique ID on the MDIO bus.

Configuration and Status Signals

The Configuration and Status Signals are shown in Table 2-4. See Configuration and Status Interfaces for more information on these signals, including a breakdown of the configuration and status vectors.

Table 2-4: **Configuration and Status Ports**

Signal Name	Direction	Description
CONFIGURATION_VECTOR[6:0]	IN	Configuration information for the core.
STATUS_VECTOR[7:0]	OUT	Status information from the core.
ALIGN_STATUS	OUT	'1' when the XAUI receiver is aligned across all four lanes, '0' otherwise.
SYNC_STATUS[3:0]	OUT	Each pin is '1' when the respective XAUI lane receiver is synchronized to byte boundaries, '0' otherwise.

Clocking and Reset Signals and Module

Included in the example design top-level sources are circuits for clock and reset management. These can include Digital Clock Managers (DCMs), Mixed-Mode Clock Managers (MMCMs), reset synchronizers, or other useful utility circuits that might be useful in your particular application.

Table 2-5 shows the ports on the netlist that are associated with system clocks and resets.

Table 2-5: **Clock and Reset Ports**

Signal Name	Direction	Description
USRCLK	IN	System clock for core; must also be used to clock the device-specific transceiver logic ports.
RESET	IN	Reset port synchronous to USRCLK.
SOFT_RESET	OUT	Reset signal controlled by MDIO register bit. This reset signal also resets the transceivers.

Chapter 3

Customizing and Generating the Core

The XAUI core is generated using the Xilinx® CORE Generator™ system. This chapter describes how to customize the XAUI core to your requirements and then generate the core netlist.

Graphical User Interface

Figure 3-1 shows the main screen for customizing the XAUI core.

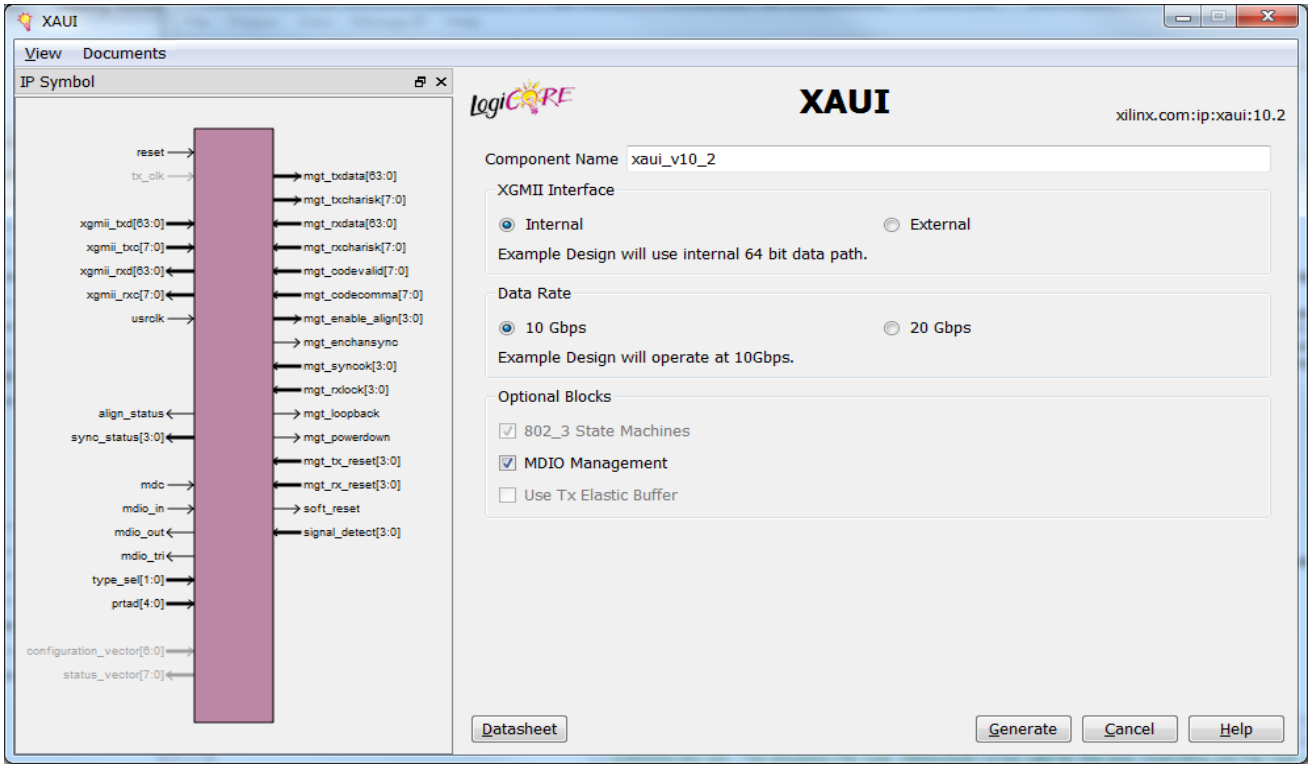

Figure 3-1: **XAUI Main Screen**

For general help with starting and using the CORE Generator tool on your development system, see the documentation supplied with the ISE® design suite.

Component Name

The component name is used as the base name of the output files generated for the core. Names must begin with a letter and must be composed from the following characters: a through z, 0 through 9 and "_" (underscore).

XGMII Interface

This control selects between the internal 64-bit client-side interface and the external XGMII client-side interface.

The default is the internal 64-bit interface.

Data Rate

This controls whether the example design is configured to run at the normal 10 Gb/s data rate or if it is over-clocked to run at 20 Gb/s.

The default is to run at 10 Gb/s.

802_3 State Machines

This controls whether the receive synchronization and alignment state machines are implemented as full *IEEE 802.3-2008* state machines in the logic of the FPGA or use the simplified state machines implemented inside the device-specific transceivers.

The default is to implement the *IEEE 802.3-2008* state machines.

MDIO Management

Select this option to implement the MDIO interface for managing the core. Deselect the option to remove the MDIO interface and expose a simple bit vector to manage the core.

The default is to implement the MDIO interface.

Use Tx Elastic Buffer

Select this option to implement a clock-correcting elastic buffer in the transmit path of the core. Deselect the option to omit the buffer and have a single-clock domain in the transmit path. See Transmit Elastic Buffer in Chapter 6 for more information on the use of the transmit elastic buffer.

The default is to omit the transmit elastic buffer.

Parameter Values in the XCO File

XCO files contain parameterization information for an instance of a core; an Xilinx CORE Generator (XCO) file is created when a core is generated and can be used to recreate a core. The text in an XCO file is case-insensitive.

Table 3-1 shows the XCO file parameters and values, and summarizes the Graphical User Interface (GUI) defaults. The following is an example extract from an XCO file:

```
SELECT XAUI family Xilinx,_Inc. 10.2
CSET component_name = the_core
CSET 802_3ae_state_machines = true
CSET data_rata TenGbps
CSET mdio_management = true
CSET use_tx_elastic_buffer = false
CSET xgmii_interface = internal
GENERATE
```

Table 3-1: **XCO File Values and Defaults**

Parameter	XCO File Values	Defaults
component_name	ASCII text starting with a letter and based upon the following character set: a...z, 0...9 and _	Blank
data_rate	TenGbps, TwentyGbps	TenGbps
802_3ae_state_machines	True, false	True
mdio_management	True, false	True
use_tx_elastic_buffer	True, false	False
xgmii_interface	Internal, external	Internal

Output Generation

The output files generated from the CORE Generator tool are placed in the project directory. The list of output files includes:

- The netlist files for the core
- XCO files
- Release notes and documentation
- An Hardware Description Language (HDL) example design
- Scripts to synthesize, implement and simulate the example design.

See Chapter 10, Detailed Example Design for a complete description of the CORE Generator tool output files and for details of the HDL example design.

XILINX®

Chapter 4

Designing with the Core

This chapter provides a general description of how to use the XAUI core in your designs and should be used in conjunction with Chapter 5, Interfacing to the Core which describes specific core interfaces.

This chapter also describes the steps required to turn a XAUI core into a fully-functioning design with user-application logic. It is important to realize that not all implementations require all of the design steps listed in this chapter. Follow the logic design guidelines in this manual carefully.

Use the Example Design as a Starting Point

Each instance of the XAUI core created by the Xilinx® CORE Generator™ tool is delivered with an example design that can be implemented in an FPGA and simulated. This design can be used as a starting point for your own design or can be used to sanity-check your application in the event of difficulty.

See Chapter 9, Quick Start Example Design for information about using and customizing the example designs for the XAUI core.

Know the Degree of Difficulty

XAUI designs are challenging to implement in any technology, and the degree of difficulty is further influenced by:

- Maximum system clock frequency
- Targeted device architecture
- Nature of your application

All XAUI implementations need careful attention to system performance requirements. Pipelining, logic mapping, placement constraints, and logic duplication 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 your application and the core. This means that all inputs and outputs from your application should come from, or connect to a flip-flop. While registering signals might not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx tools to place and route the design.

Recognize Timing Critical Signals

The UCF provided with the example design for the core identifies the critical signals and the timing constraints that should be applied. See Chapter 6, Constraining the Core for further information.

Use Supported Design Flows

The core is synthesized in the CORE Generator tool and is delivered to you as an NGC netlist. The example implementation scripts provided currently use Xilinx Synthesis Technology (XST) as the synthesis tool for the HDL example design that is delivered with the core. Other synthesis tools can be used for your application logic; the core is always unknown to the synthesis tool and appears as a black box.

Post synthesis, only Xilinx ISE® v13.4 tools are supported.

Make Only Allowed Modifications

The XAUI core is not user-modifiable. Do not make modifications as they might have adverse effects on system timing and protocol compliance. Supported user configurations of the XAUI core can only be made by selecting the options from within the CORE Generator tool when the core is generated. See Chapter 3, Customizing and Generating the Core.

XILINX®

Chapter 5

Interfacing to the Core

This chapter describes how to connect to the data interfaces of the core and configuration and status interfaces of the XAUI core.

Data Interface: Internal vs External XGMII Interfaces

External XGMII 32-bit DDR Client-Side Interface

Although used less often than the 64-bit interface described in the following section, the 32-bit DDR interface is functionally identical to the XGMII interface and is therefore easier to relate to the *IEEE Std. 802.3-2008* specification.

Virtex-7, Kintex-7, Virtex-6, Virtex-5 and Virtex-4 FPGAs

XGMII on Virtex®-7, Kintex™-7, Virtex-6, Virtex-5 and Virtex-4 FPGAs uses the IDDR and ODDR primitives to convert from single-data rate to double-data rate and back again.

On the transmit side of the core, the IDDR primitives receive the inbound data then separate it into a bus twice as wide as the input bus, with the data clocked in on the falling edge in the upper half of the output bus. Using the SAME_EDGE mode of the IDDR primitive, all bits across the bus are in the rising-edge clock domain. This is depicted in Figure 5-1, and a timing diagram is shown in Figure 5-2. Similarly on receive, the ODDR primitive is used in SAME_EDGE mode, and all outbound data is rising-edge clocked from the netlist.

For more information on the IDDR and ODDR primitives, see the following manuals:

- *7 Series FPGAs SelectIO Resources User Guide* (UG471)
- *Virtex-6 FPGA SelectIO Resources User Guide* (UG361)
- *Virtex-5 FPGA User Guide* (UG190)
- *Virtex-4 FPGA User Guide* (UG070)

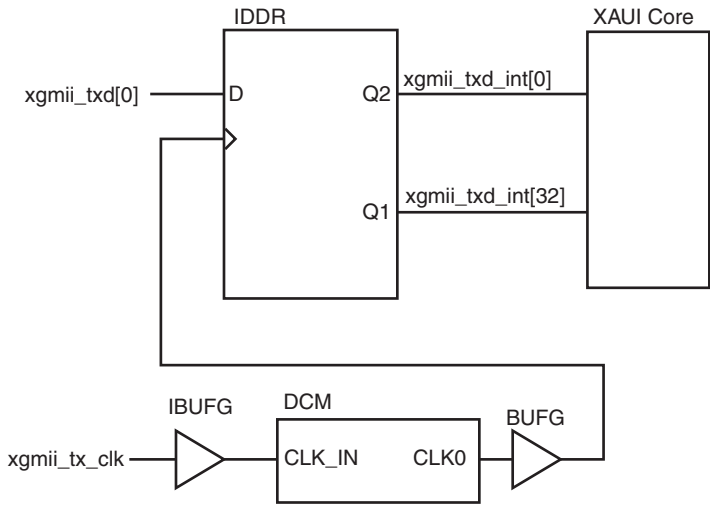

Figure 5-1: **Schematic of Inbound DDR Interface: Virtex-7, Kintex-7, Virtex-6, Virtex-5 and Virtex-4 FPGAs**

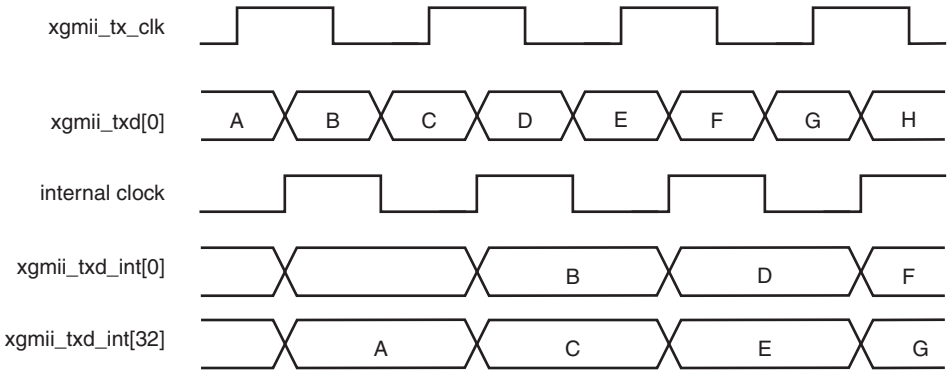

Figure 5-2: **Timing of Operation of Inbound DDR Interface: Virtex-7, Kintex-7, Virtex-6, Virtex-5 and Virtex-4 FPGAs**

Internal 64-bit SDR Client-side Interface

The 64-bit single-data rate (SDR) client-side interface is based upon the 32-bit XGMII-like interface described previously. The key difference is a demultiplexing of the bus from 32-bits wide to 64-bits wide on a single rising clock edge. This demultiplexing is done by extending the bus upwards so that there are now eight lanes of data numbered 0-7; the lanes are organized such that data appearing on lanes 4–7 is transmitted or received *later* in time than that in lanes 0-3.

The mapping of lanes to data bits is shown in Table 5-1. The lane number is also the index of the control bit for that particular lane; for example, `XGMII_TXC[2]` and `XGMII_TXD[23:16]` are the control and data bits respectively for lane 2.

Table 5-1: **XGMII_TXD, XGMII_RXD Lanes for Internal 64-bit Client-Side Interface**

Lane	XGMII_TXD, XGMII_RXD Bits
0	7:0
1	15:8
2	23:16
3	31:24
4	39:32
5	47:40
6	55:48
7	63:56

Definitions of Control Characters

Reference is regularly made to certain XGMII control characters signifying Start, Terminate, Error, and others. These control characters all have in common that the control line for that lane is 1 for the character and a certain data byte value. The relevant characters are defined in the *IEEE Std. 802.3-2008* and are reproduced in Table 5-2 for reference.

Table 5-2: **Partial List of XGMII Characters**

Data (Hex)	Control	Name, Abbreviation
00 to FF	0	Data (D)
07	1	Idle (I)
FB	1	Start (S)
FD	1	Terminate (T)
FE	1	Error (E)

Interfacing to the Transmit Client Interface

External 32-bit DDR Interface

The timing of a data frame transmission via the external XGMII interface is shown in Figure 5-3. The beginning of the data frame is marked by the presence of the Start Character (the S character in lane 0), followed by data characters in lanes 1, 2, and 3.

The termination of the data frame is marked by the occurrence of the Terminate character (the T in lane 2). The Terminate character can occur in any lane; the remaining lanes are padded by XGMII Idle characters.

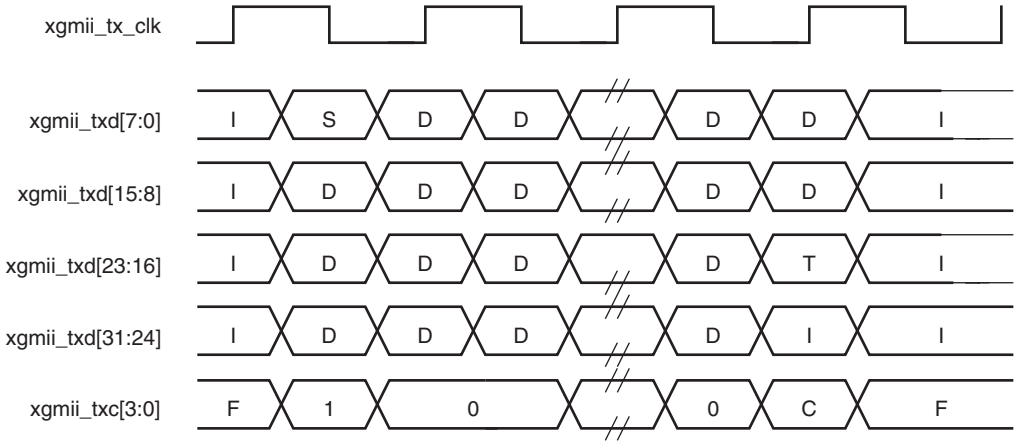

Figure 5-3: **Frame Transmission Across the 32-bit XGMII Interface**

The timing of a data frame transmission containing an error via the external XGMII interface is shown in Figure 5-4. The presence of the error is denoted by the letter E in lanes 0, 1, 2, and 3.

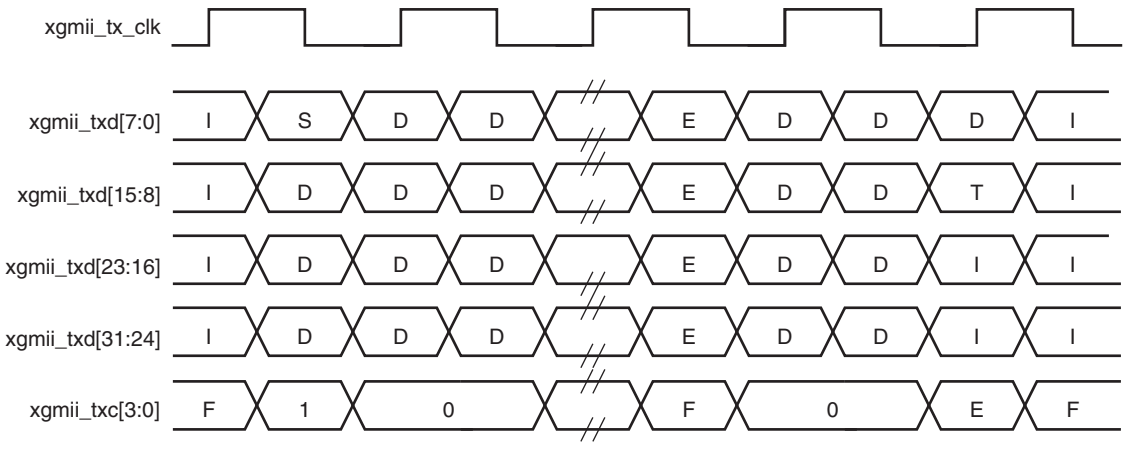

Figure 5-4: **Frame Transmission with Errors Across 32-bit XGMII Interface**

Internal 64-bit Client-Side Interface

The timing of a data frame transmission via the internal 64-bit client-side interface is shown in Figure 5-5. The beginning of the data frame is shown by the presence of the Start character (the /S/ codegroup in lane 4 of Figure 5-5) followed by data characters in lanes 5, 6, and 7. Alternatively the start of the data frame can be marked by the occurrence of a Start character in lane 0, with the data characters in lanes 1 to 7.

When the frame is complete, it is completed by a Terminate character (the T in lane 1 of Figure 5-5). The Terminate character can occur in any lane; the remaining lanes are padded by XGMII idle characters.

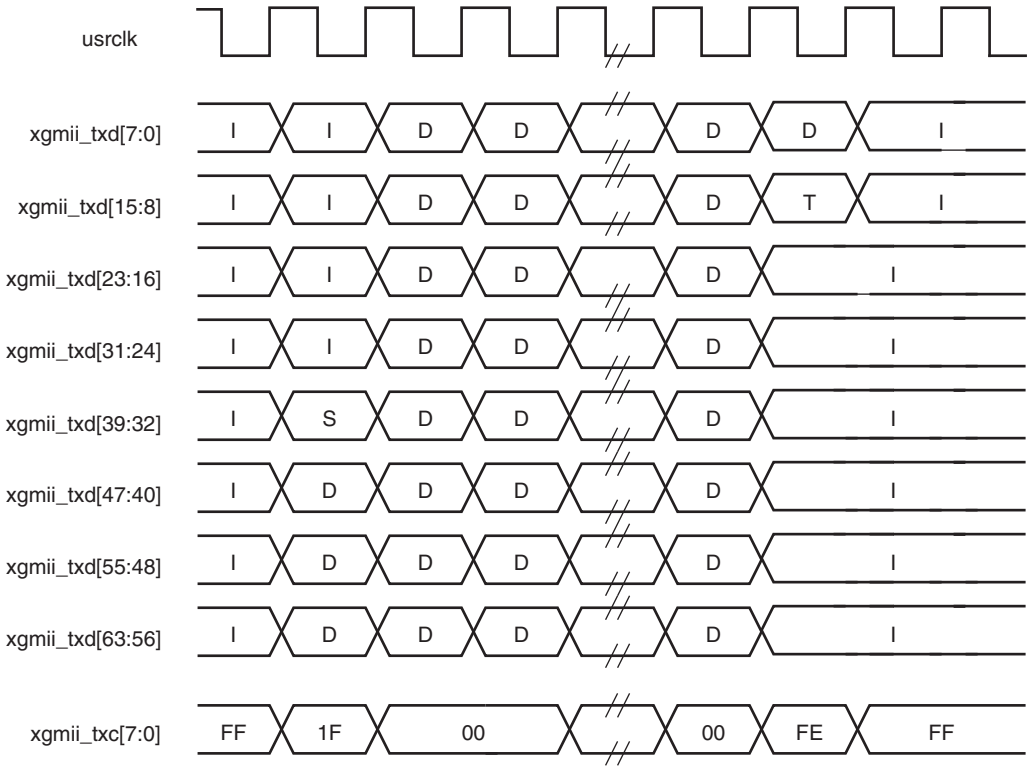

Figure 5-5: **Normal Frame Transmission Across the Internal 64-bit Client-Side I/F**

Figure 5-6 depicts a similar frame to that in Figure 5-5, with the exception that this frame is propagating an error. The error code is denoted by the letter E, with the relevant control bits set.

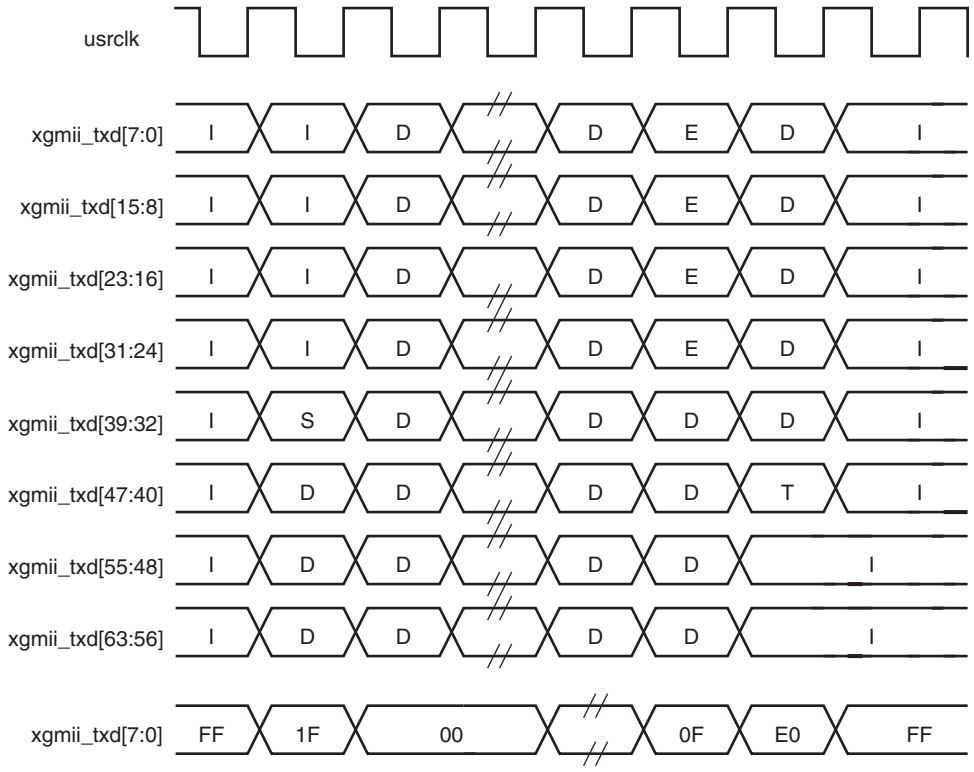

Figure 5-6: **Frame Transmission with Error Across Internal 64-bit Client-Side I/F**

Interfacing to the Receive Client Interface

External 32-bit DDR Client-Side Interface

Figure 5-7 shows a received frame transferred across the external XGMII. The beginning of the data frame is delimited by the presence of the Start Character (the S character in lane 0), followed by data characters in lanes 1, 2, and 3.

The termination of the data frame is marked by the occurrence of the Terminate character (the T character in lane 3). The Terminate character can occur in any lane.

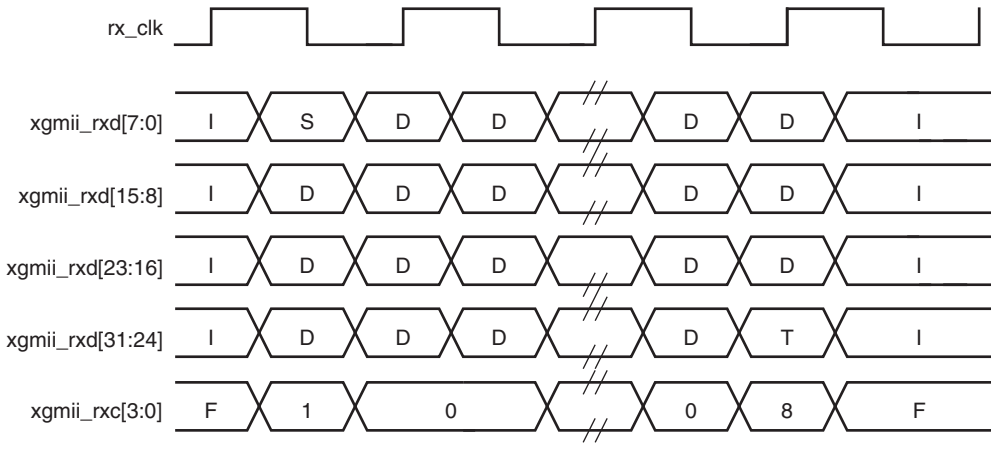

Figure 5-7: **Frame Reception Across External 32-bit XGMII Interface**

Figure 5-8 shows an inbound frame containing an error via the external XGMII. The error in the inbound frame is denoted by the letter E, in lanes 0 to 3.

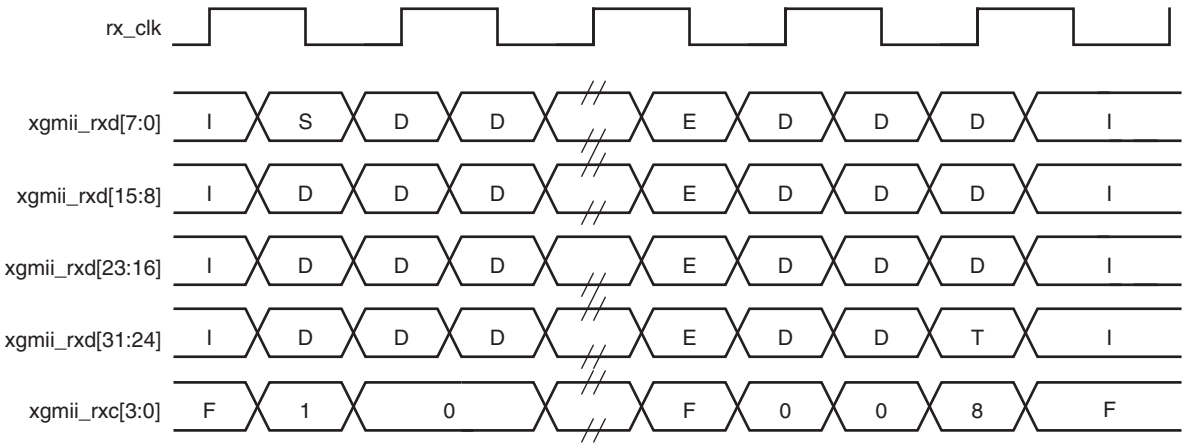

Figure 5-8: **Frame Reception with Error Across External 32-bit XGMII Interface**

Internal 64-bit Client-Side Interface

The timing of a normal inbound frame transfer is shown in Figure 5-9. As in the transmit case, the frame is delimited by a Start character (S) and by a Terminate character (T). The Start character in this implementation can occur in either lane 0 or in lane 4. The Terminate character, T, can occur in any lane.

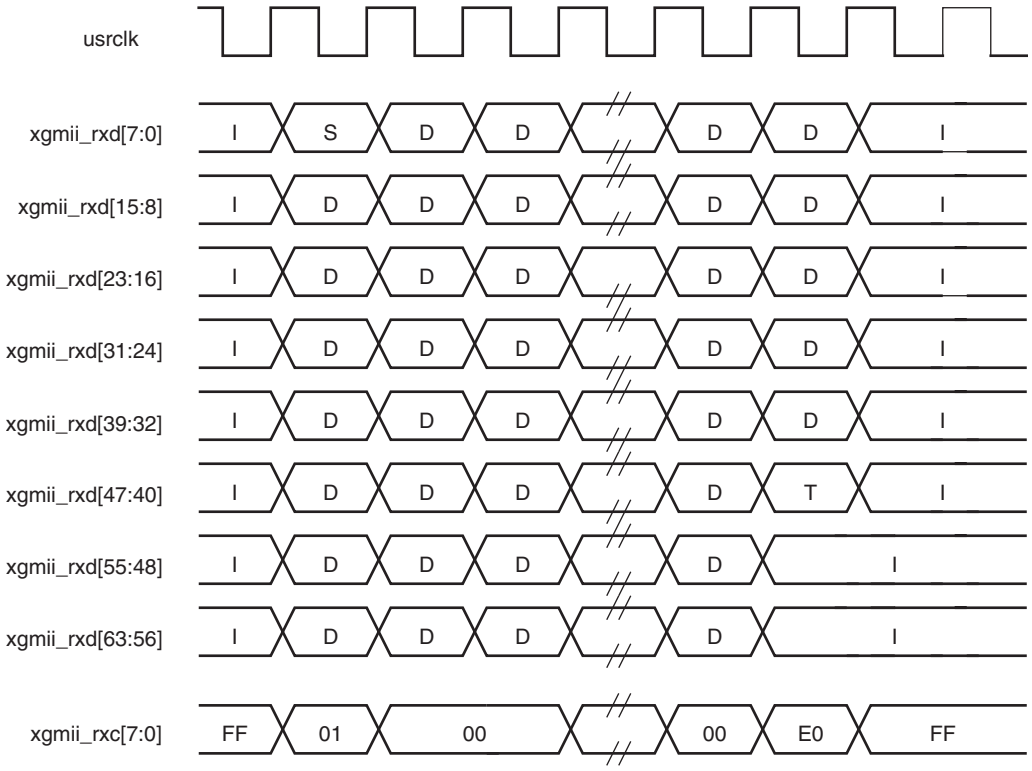

Figure 5-9: **Frame Reception Across the Internal 64-bit Client Interface**

Figure 5-10 shows an inbound frame of data propagating an error. In this instance, the error is propagated in lanes 4 to 7, shown by the letter E.

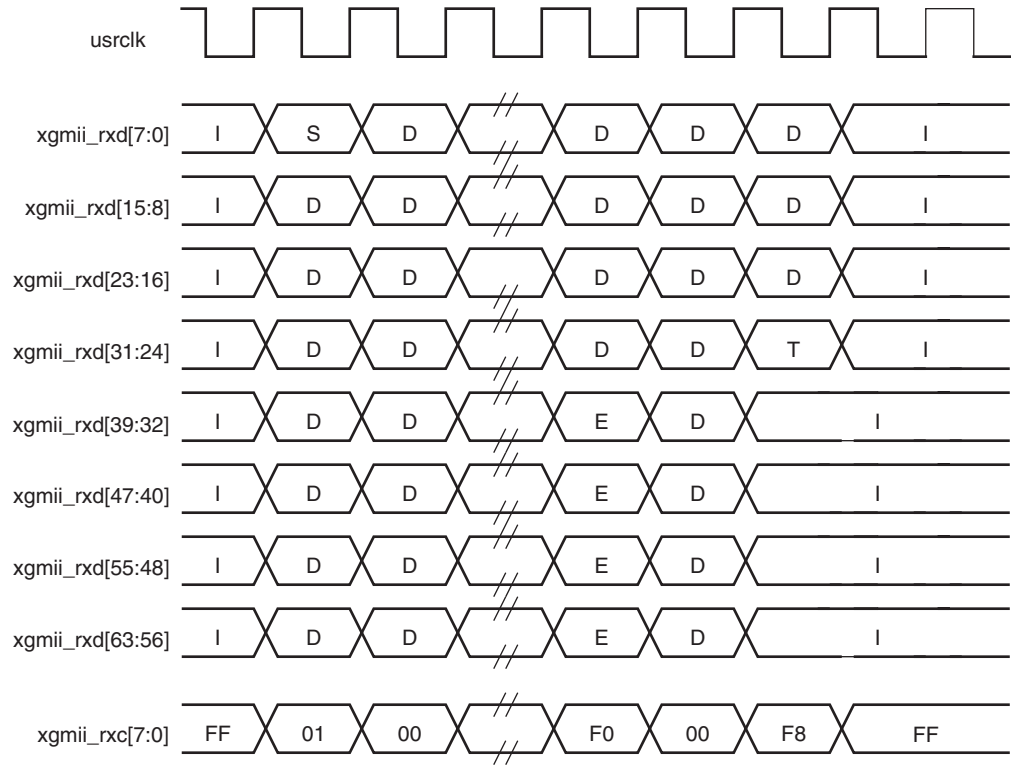

Figure 5-10: **Frame Reception with Error Across the Internal 64-bit Client Interface**

Interfacing to the Transceivers

The Virtex-4 transceivers require a Calibration Block to be included in the FPGA logic. (See the *Calibration Block User Guide* for more information. Information about the *Calibration Block User Guide* can be found in Answer Record 22477.) The example design provided with the XAUI core instantiates the calibration blocks required when targeting a FX60 device.

Virtex-7, Kintex-7, Virtex-6, Virtex-5 and Spartan-6 FPGAs

For Virtex-7 and Kintex-7 FPGA GTX, Virtex-6 FPGA GTX, Virtex-5 FPGA GTP/GTX and Spartan-6 FPGA GTP transceivers, Table 5-3 shows the ports of the netlist that are to be connected to the device-specific transceivers. The remainder of the device-specific transceiver ports are not connected to the netlist, but are connected in the core source code (device_specific_wrapper.vhd or device_specific_wrapper.v) or are wired to static values.

Table 5-3: **Transceiver Interface Ports**

Signal Name	Direction	Description
MGT_TXDATA[63:0]	OUT	Device-specific transceiver transmit data
MGT_TXCHARISK[7:0]	OUT	Device-specific transceiver transmit control flags
MGT_RXDATA[63:0]	IN	Device-specific transceiver receive data
MGT_RXCHARISK[7:0]	IN	Device-specific transceiver receive control signals
MGT_CODEVALID[7:0]	IN	Device-specific transceiver receive control signals
MGT_CODECOMMA[7:0]	IN	Device-specific transceiver receive control signals
MGT_ENABLE_ALIGN[3:0]	OUT	Device-specific transceiver control signals
MGT_ENCHANSYNC	OUT	Device-specific transceiver control signal
MGT_RXLOCK[3:0]	IN	Device-specific transceiver control signals
MGT_SYNCOK[3:0]	IN	Device-specific transceiver control signals
MGT_LOOPBACK	OUT	Device-specific transceiver control signal
MGT_POWERDOWN	OUT	Device-specific transceiver control signal
MGT_TX_RESET[3:0]	IN	Reset status signal from example design
MGT_RX_RESET[3:0]	IN	Reset status signal from example design
SIGNAL_DETECT[3:0]	IN	Status signal from attached optical module

The `SIGNAL_DETECT` signals are intended to be driven by an attached 10GBASE-LX4 optical module; they signify that each of the four optical receivers is receiving illumination and is therefore not just putting out noise. If an optical module is not in use, this four-wire bus should be tied to 1111.

The `MGT_TX_RESET` and `MGT_RX_RESET` signals are used to set the local fault registers in the management block of the core upon transceiver reset event. The example design connects these ports in the required way.

No timing diagrams are presented here for the device-specific transceiver signals. You should treat this interface as a black box. If customization of this interface is required, see the following for detailed descriptions of the transceiver ports.

- *7 Series FPGAs Transceivers User Guide (UG769)*
- *Virtex-6 FPGA GTX Transceiver User Guide (UG366)*
- *Spartan-6 FPGA GTP Transceiver User Guide* (UG386)
- *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* (UG196)
- *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* (UG198)

This chapter describes the interfaces available for dynamically setting the configuration and obtaining the status of the XAUI core. There are two interfaces for configuration; depending on the core customization, only one is available in a particular core instance.

In addition, there are output ports on the core signalling alignment and synchronization status. These ports are described in Alignment and Synchronization Status Ports.

Virtex-4 FPGAs

Table 5-4 shows the ports of the netlist that are to be connected to the device-specific RocketIO™ transceivers. The remainder of the device-specific transceiver ports are not connected to the netlist, but are connected in the core source code (rocketio_wrapper.vhd or rocketio_wrapper.v) or are wired to static values.

Table 5-4: **RocketIO Transceiver Interface Ports - Virtex-4 FPGAs**

Signal Name	Direction	Description
MGT_TXDATA[63:0]	OUT	RocketIO transceiver transmit data
MGT_TXCHARISK[7:0]	OUT	RocketIO transceiver transmit control flags
MGT_RXDATA[63:0]	IN	RocketIO transceiver receive data
MGT_RXCHARISK[7:0]	IN	RocketIO transceiver receive control signals
MGT_CODEVALID[7:0]	IN	RocketIO transceiver receive control signals
MGT_CODECOMMA[7:0]	IN	RocketIO transceiver receive control signals
MGT_ENABLE_ALIGN[3:0]	OUT	RocketIO transceiver control signals
MGT_ENCHANSYNC	OUT	RocketIO transceiver control signal
MGT_RXLOCK[3:0]	IN	RocketIO transceiver control signal
MGT_LOOPBACK	OUT	RocketIO transceiver control signal
MGT_POWERDOWN	OUT	RocketIO transceiver control signal
MGT_TX_RESET[3:0]	IN	Reset status signal from example design
MGT_RX_RESET[3:0]	IN	Reset status signal from example design
SIGNAL_DETECT[3:0]	IN	Status signal from attached optical module

The `SIGNAL_DETECT` signals are intended to be driven by an attached 10GBASE-LX4 optical module; they signify that each of the four optical receivers is receiving illumination and is therefore not just putting out noise. If an optical module is not in use, this four-wire bus should be tied to 1111.

The MGT_TX_RESET and MGT_RX_RESET signals are used to set the local fault registers in the management block of the core upon transceiver reset event. The example design connects these ports in the required way.

No timing diagrams are presented here for the device-specific RocketIO transceiver signals; you should treat this interface as a black box. If customization of this interface is required, see the *Virtex-4 FPGA RocketIO Multi-Gigabit Transceiver User Guide* (UG076) for detailed descriptions of the transceiver ports.

This section describes the interfaces available for dynamically setting the configuration and obtaining the status of the XAUI core. There are two interfaces for configuration; depending on the core customization, only one is available in a particular core instance. The interfaces are:

- MDIO Interface
- Configuration and Status Vectors

In addition, there are output ports on the core signalling alignment and synchronization status. These ports are described in Alignment and Synchronization Status Ports.

Configuration and Status Interfaces

This section describes the interfaces available for dynamically setting the configuration and obtaining the status of the XAUI core. There are two interfaces for configuration; depending on the core customization, only one is available in a particular core instance. The interfaces are:

- MDIO Interface
- Configuration and Status Vectors

In addition, there are output ports on the core signalling alignment and synchronization status. These ports are described in Alignment and Synchronization Status Ports.

MDIO Interface

The Management Data Input/Output (MDIO) interface is a simple, low-speed two-wire interface for management of the XAUI core consisting of a clock signal and a bidirectional data signal. It is defined in clause 45 of *IEEE Standard 802.3-2008*.

An MDIO bus in a system consists of a single Station Management (STA) master management entity and a number of MDIO Managed Device (MMD) slave entities.

Figure 5-11 illustrates a typical system. All transactions are initiated by the Station Management Entity (STA) entity. The XAUI core implements an MMD.

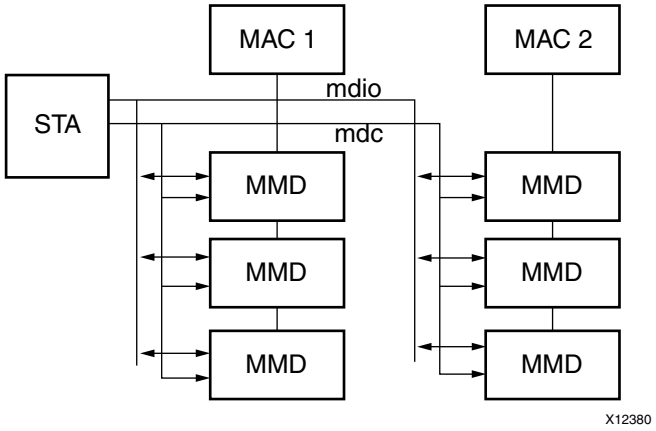

Figure 5-11: **A Typical MDIO-Managed System**

MDIO Ports

The core ports associated with MDIO are shown in Table 5-5.

Table 5-5: **MDIO Management Interface Port Description**

Signal Name	Direction	Description
MDC	IN	Management clock
MDIO_IN	IN	MDIO input
MDIO_OUT	OUT	MDIO output
MDIO_TRI	OUT	MDIO 3-state. 1 disconnects the output driver from the MDIO bus.
TYPE_SEL[1:0]	IN	Type select
PRTAD[4:0]	IN	MDIO port address

If implemented, the MDIO interface is implemented as four unidirectional signals. These can be used to drive a 3-state buffer either in the FPGA SelectIO™ interface buffer or in a separate device. Figure 5-12 illustrates the use of a Virtex-4 FPGA SelectIO interface 3-state buffer as the bus interface.

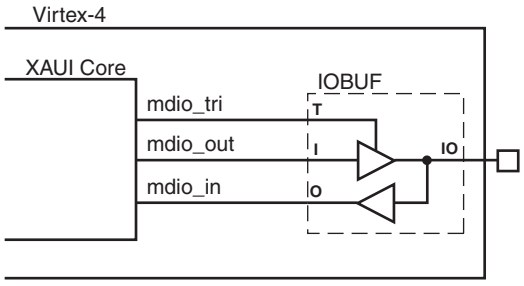

Figure 5-12: **Using a SelectIO Interface 3-State Buffer to Drive MDIO**

The `type_sel` port is registered into the core at FPGA configuration and core hard reset; changes after that time are ignored by the core. Table 5-6 shows the mapping of the `type_sel` setting to the implemented register map.

Table 5-6: **Mapping of type_sel Port Settings to MDIO Register Type**

type_sel setting	MDIO Register	Description
00or 01	10GBASE-X PCS/PMA	When driving a 10GBASE-X PHY
10	Data Terminal Equipment (DTE) XGMII Extender Sublayer (XGXS)	When connected to a 10GMAC via XGMII
11	PHY XGXS	When connected to a PHY via XGMII

The `prtad[4:0]` port sets the port address of the core instance. Multiple instances of the same core can be supported on the same MDIO bus by setting `prtad[4:0]` to a unique value for each instance; the XAUI core ignores transactions with the PRTAD field set to a value other than that on its `prtad[4:0]` port.

MDIO Transactions

The MDIO interface should be driven from a STA master according to the protocol defined in *IEEE Std. 802.3-2008*. An outline of each transaction type is described in the following sections. In these sections, the following abbreviations apply:

- PRE: preamble
- ST: start
- OP: operation code
- PRTAD: port address
- DEVAD: device address
- TA: turnaround

Set Address Transaction

Figure 5-13 shows an Address transaction defined by OP=00. Set Address is used to set the internal 16-bit address register of the XAUI core for subsequent data transactions (called the “current address” in the following sections).

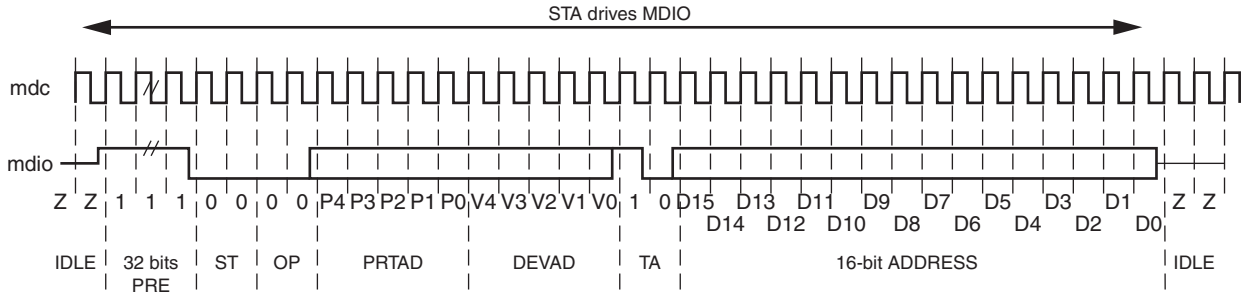

Figure 5-13: **MDIO Set Address Transaction**

Write Transaction

Figure 5-14 shows a Write transaction defined by OP=01. The XAUI core takes the 16-bit word in the data field and writes it to the register at the current address.

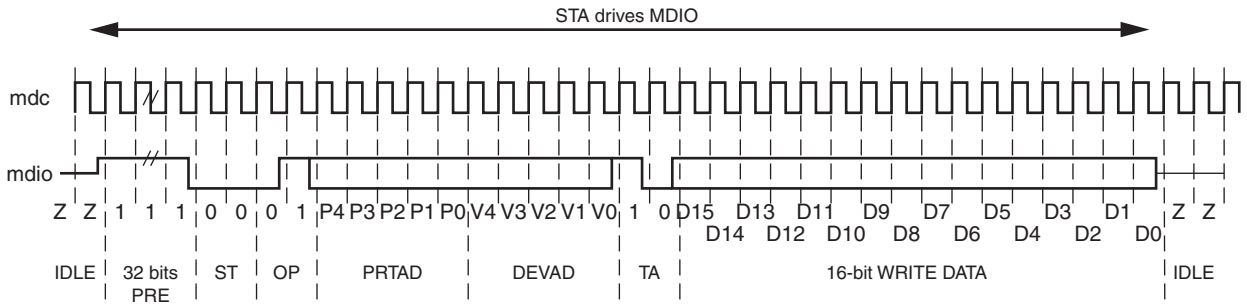

Figure 5-14: **MDIO Write Transaction**

Read Transaction

Figure 5-15 shows a Read transaction defined by OP=11. The XAUI core returns the 16-bit word from the register at the current address.

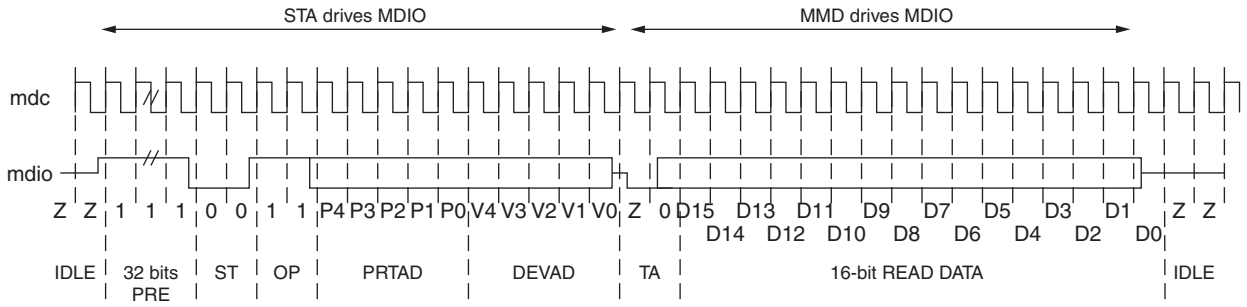

Figure 5-15: **MDIO Read Transaction**

Post-read-increment-address Transaction

Figure 5-16 shows a Post-read-increment-address transaction, defined by OP=10. The XAUI core returns the 16-bit word from the register at the current address then increments the current address. This allows sequential reading or writing by a STA master of a block of register addresses.

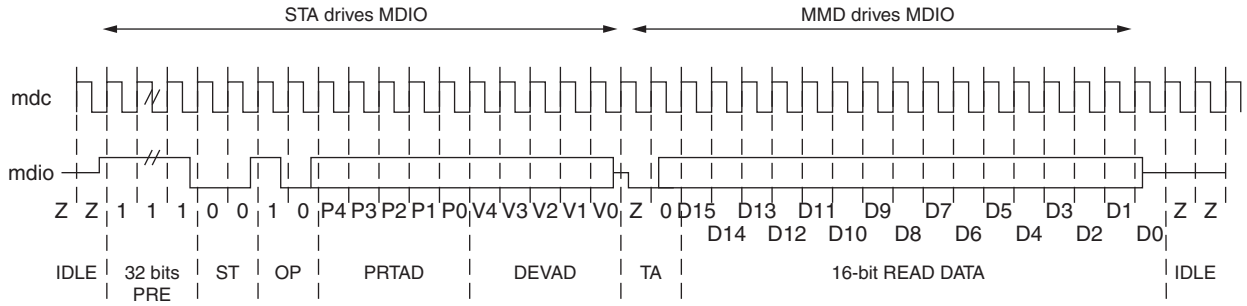

Figure 5-16: **MDIO Read-and-increment Transaction**

10GBASE-X PCS/PMA Register Map

When the core is configured as a 10GBASE-X Physical Coding Sublayer/Physical Medium Attachment (PCS/PMA), it occupies MDIO Device Addresses 1 and 3 in the MDIO register address map, as shown in Table 5-7.

Table 5-7: **10GBASE-X PCS/PMA MDIO Registers**

Register Address	Register Name
1.0	Physical Medium Attachment/Physical Medium Dependent (PMA/PMD) Control 1
1.1	PMA/PMD Status 1
1.2,1.3	PMA/PMD Device Identifier
1.4	PMA/PMD Speed Ability
1.5, 1.6	PMA/PMD Devices in Package
1.7	10G PMA/PMD Control 2
1.8	10G PMA/PMD Status 2
1.9	Reserved
1.10	10G PMD Receive Signal OK
1.11 TO 1.13	Reserved
1.14, 1.15	PMA/PMD Package Identifier
1.16 to 1.65 535	Reserved
3.0	PCS Control 1
3.1	PCS Status 1
3.2, 3.3	PCS Device Identifier
3.4	PCS Speed Ability
3.5, 3.6	PCS Devices in Package
3.7	10G PCS Control 2
3.8	10G PCS Status 2
3.9 to 3.13	Reserved
3.14, 3.15	Package Identifier
3.16 to 3.23	Reserved
3.24	10GBASE-X PCS Status
3.25	10GBASE-X Test Control
3.26 to 3.65 535	Reserved

MDIO Register 1.0: PMA/PMD Control 1

Figure 5-17 shows the MDIO Register 1.0: PMA/PMD Control 1.

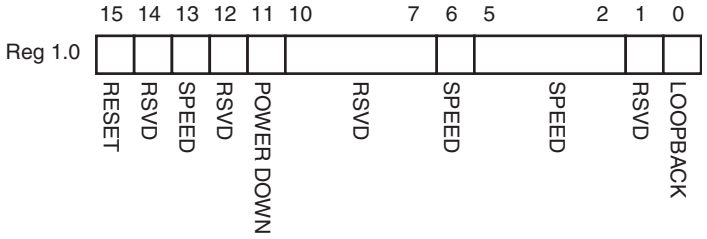

Figure 5-17: **PMA/PMD Control 1 Register**

Table 5-8 shows the PMA Control 1 register bit definitions.

Table 5-8: **PMA/PMD Control 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.0.15	Reset	1 = Block reset 0 = Normal operation The XAUI block is reset when this bit is set to 1. It returns to 0 when the reset is complete. The soft_reset pin is connected to this bit. This can be connected to the reset of any other MMDs.	R/W Self-clearing	0
1.0.14	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0
1.0.13	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
1.0.12	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0
1.0.11	Power down	1 = Power down mode 0 = Normal operation When set to 1, the serial transceivers are placed in a low power state. Set to 0 to return to normal operation	R/W	0
1.0.10:7	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
1.0.6	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
1.0.5:2	Speed Selection	The block always returns 0s for these bits and ignores writes.	R/O	All 0s

Table 5-8: **PMA/PMD Control 1 Register Bit Definitions *(Cont'd)***

Bit(s)	Name	Description	Attributes	Default Value
1.0.1	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	All 0s
1.0.0	Loopback	1 = Enable loopback mode 0 = Disable loopback mode The XAUI block loops the signal in the serial transceivers back into the receiver. In Virtex-4 FPGA implementations it is necessary to enable / disable the TXPOST_TAP_PD bit via the GT11 Dynamic Reconfiguration Port (DRP) interface. For Virtex-5 LXT / SXT FPGA implementation it might be necessary to change GTP transceiver attributes and receiver pins under marginal conditions. See the Near-End PMA Loopback section in the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* (UG196).	R/W	0

MDIO Register 1.1: PMA/PMD Status 1

Figure 5-18 shows the MDIO Register 1.1: PMA/PMD Status 1.

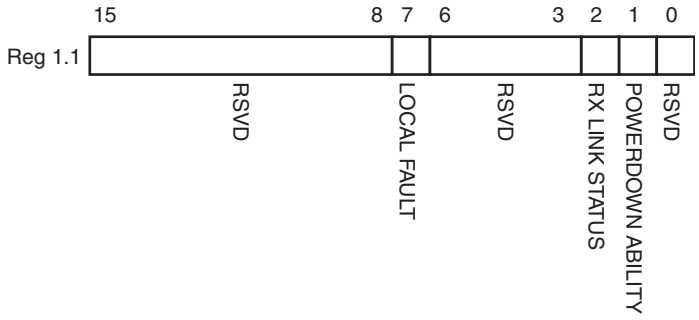

Figure 5-18: **PMA/PMD Status 1 Register**

Table 5-9 shows the PMA/PMD Status 1 register bit definitions.

Table 5-9: **PMA/PMD Status 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.1.15:8	Reserved	The block always returns 0 for this bit.	R/O	0
1.1.7	Local Fault	The block always returns 0 for this bit.	R/O	0
1.1.6:3	Reserved	The block always returns 0 for this bit.	R/O	0
1.1.2	Receive Link Status	The block always returns 1 for this bit.	R/O	1
1.1.1	Power Down Ability	The block always returns 1 for this bit.	R/O	1
1.1.0	Reserved	The block always returns 0 for this bit.	R/O	0

MDIO Registers 1.2 and 1.3: PMA/PMD Device Identifier

Figure 5-19 shows the MDIO Registers 1.2 and 1.3: PMA/PMD Device Identifier.

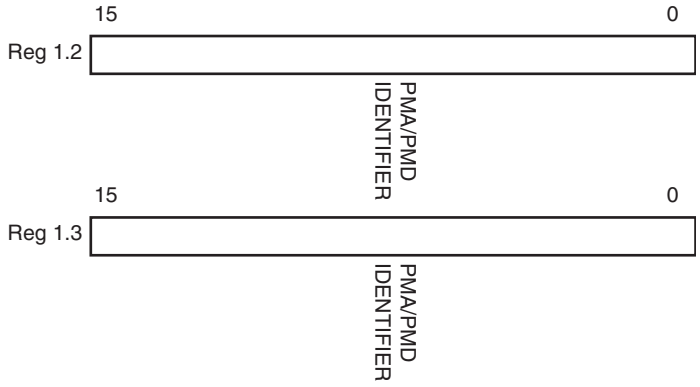

Figure 5-19: **PMA/PMD Device Identifier Registers**

Table 5-10 shows the PMA/PMD Device Identifier registers bit definitions.

Table 5-10: **PMA/PMD Device Identifier Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.2.15:0	PMA/PMD Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
1.3.15:0	PMA/PMD Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s

MDIO Register 1.4: PMA/PMD Speed Ability

Figure 5-20 shows the MDIO Register 1.4: PMA/PMD Speed Ability.

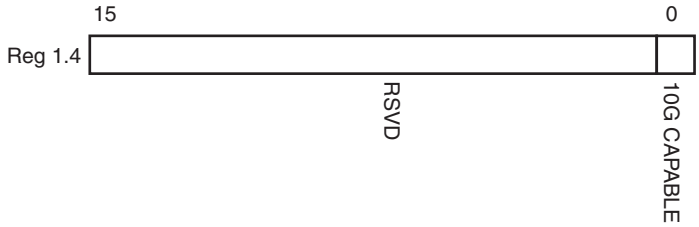

Figure 5-20: **PMA/PMD Speed Ability Register**

Table 5-11 shows the PMA/PMD Speed Ability register bit definitions.

Table 5-11: **PMA/PMD Speed Ability Register Bit Definitions**

Bit(s)	Name	Description	Attribute	Default Value
1.4.15:1	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
1.4.0	10G Capable	The block always returns 1 for this bit and ignores writes.	R/O	1

MDIO Registers 1.5 and 1.6: PMA/PMD Devices in Package

Figure 5-21 shows the MDIO Registers 1.5 and 1.6: PMA/PMD Devices in Package.

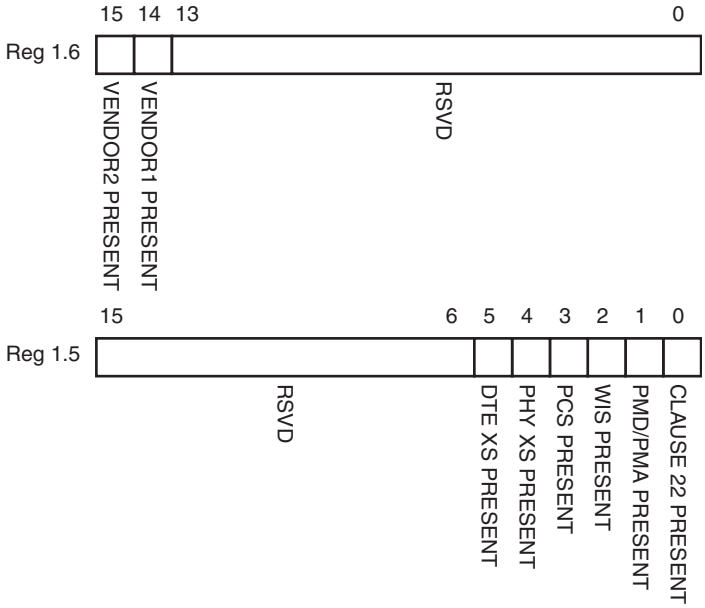

Figure 5-21: **PMA/PMD Devices in Package Registers**

Table 5-12 shows the PMA/PMD Device in Package registers bit definitions.

Table 5-12: **PMA/PMD Devices in Package Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.6.15	Vendor- specific Device 2 Present	The block always returns 0 for this bit.	R/O	0
1.6.14	Vendor-specific Device 1 Present	The block always returns 0 for this bit.	R/O	0
1.6.13:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s
1.5.15:6	Reserved	The block always returns 0 for these bits.	R/O	All 0s
1.5.5	DTE Extender Sublayer (XS) Present	The block always returns 0 for this bit.	R/O	0
1.5.4	PHY XS Present	The block always returns 0 for this bit.	R/O	0
1.5.3	PCS Present	The block always returns 1 for this bit.	R/O	1
1.5.2	WIS Present	The block always returns 0 for this bit.	R/O	0
1.5.1	PMA/PMD Present	The block always returns 1 for this bit.	R/O	1
1.5.0	Clause 22 Device Present	The block always returns 0 for this bit.	R/O	0

MDIO Register 1.7: 10G PMA/PMD Control 2

Figure 5-22 shows the MDIO Register 1.7: 10G PMA/PMD Control 2.

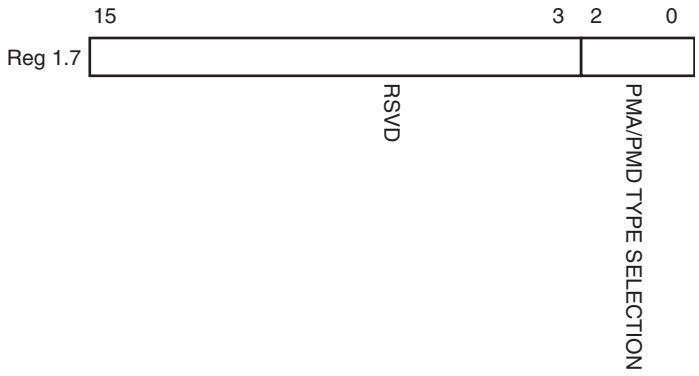

Figure 5-22: **10G PMA/PMD Control 2 Register**

Table 5-13 shows the PMA/PMD Control 2 register bit definitions.

Table 5-13: **10G PMA/PMD Control 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.7.15:3	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
1.7.2:0	PMA/PMD Type Selection	The block always returns 100 for these bits and ignores writes. This corresponds to the 10GBASE-X PMA/PMD.	R/O	100

MDIO Register 1.8: 10G PMA/PMD Status 2

Figure 5-23 shows the MDIO Register 1.8: 10G PMA/PMD Status 2.

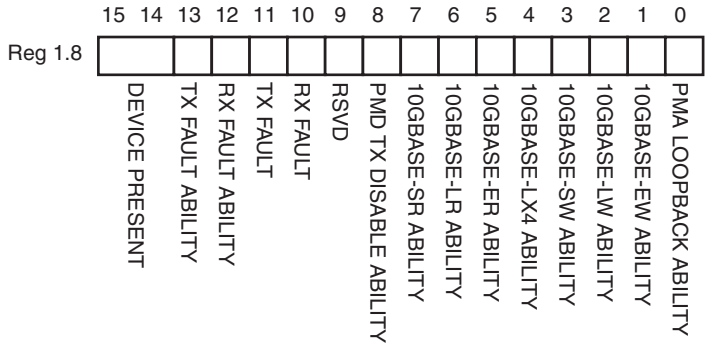

Figure 5-23: **10G PMA/PMD Status 2 Register**

Table 5-14 shows the PMA/PMD Status 2 register bit definitions.

Table 5-14: **10G PMA/PMD Status 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.8.15:14	Device Present	The block always returns 10 for these bits.	R/O	10
1.8.13	Transmit Local Fault Ability	The block always returns 0 for this bit.	R/O	0
1.8.12	Receive Local Fault Ability	The block always returns 0 for this bit.	R/O	0
1.8.11	Transmit Fault	The block always returns 0 for this bit.	R/O	0
1.8.10	Receive Fault	The block always returns 0 for this bit.\	R/O	0
1.8.9	Reserved	The block always returns 0 for this bit.	R/O	0
1.8.8	PMD Transmit Disable Ability	The block always returns 0 for this bit.	R/O	0

Table 5-14: **10G PMA/PMD Status 2 Register Bit Definitions *(Cont'd)***

Bit(s)	Name	Description	Attributes	Default Value
1.8.7	10GBASE-SR Ability	The block always returns 0 for this bit.	R/O	0
1.8.6	10GBASE-LR Ability	The block always returns 0 for this bit.	R/O	0
1.8.5	10GBASE-ER Ability	The block always returns 0 for this bit.	R/O	0
1.8.4	10GBASE-LX4 Ability	The block always returns 1 for this bit.	R/O	1
1.8.3	10GBASE-SW Ability	The block always returns 0 for this bit.	R/O	0
1.8.2	10GBASE-LW Ability	The block always returns 0 for this bit.	R/O	0
1.8.1	10GBASE-EW Ability	The block always returns 0 for this bit.	R/O	0
1.8.0	PMA Loopback Ability	The block always returns 1 for this bit.	R/O	1

MDIO Register 1.10: 10G PMD Signal Receive OK

Figure 5-24 shows the MDIO 1.10 register: 10G PMD Signal Receive OK.

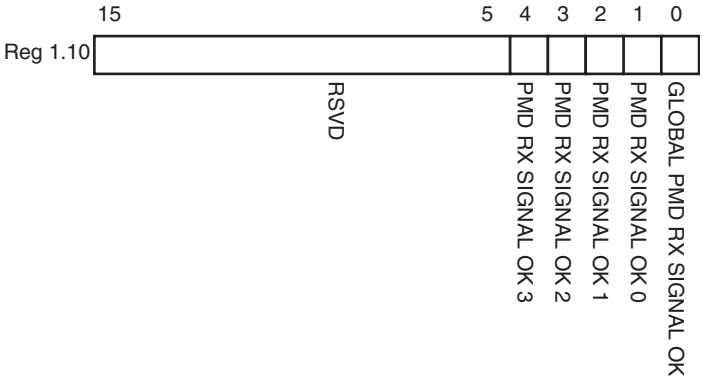

Figure 5-24: **10G PMD Signal Receive OK Register**

Table 5-15 shows the 10G PMD Signal Receive OK register bit definitions.

Table 5-15: **10G PMD Signal Receive OK Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.10.15:5	Reserved	The block always returns 0s for these bits.	R/O	All 0s
1.10.4	PMD Receive Signal OK 3	1 = Signal OK on receive Lane 3 0 = Signal not OK on receive Lane 3 This is the value of the SIGNAL_DETECT[3] port.	R/O	-
1.10.3	PMD Receive Signal OK 2	1 = Signal OK on receive Lane 2 0 = Signal not OK on receive Lane 2 This is the value of the SIGNAL_DETECT[2] port.	R/O	-
1.10.2	PMD Receive Signal OK 1	1 = Signal OK on receive Lane 1 0 = Signal not OK on receive Lane 1 This is the value of the SIGNAL_DETECT[1] port.	R/O	-
1.10.1	PMD Receive Signal OK 0	1 = Signal OK on receive Lane 0 0 = Signal not OK on receive Lane 0 This is the value of the SIGNAL_DETECT[0] port.	R/O	-
1.10.0	Global PMD Receive Signal OK	1 = Signal OK on all receive lanes 0 = Signal not OK on all receive lanes	R/O	-

MDIO Registers 1.14 and 1.15: PMA/PMD Package Identifier

Figure 5-25 shows the MDIO registers 1.14 and 1.15: PMA/PMD Package Identifier register.

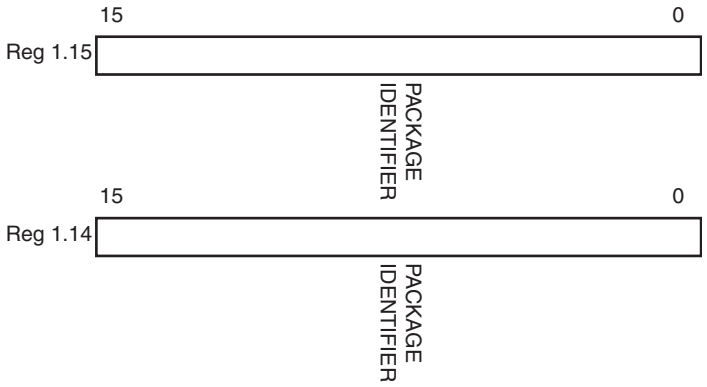

Figure 5-25: **PMA/PMD Package Identifier Registers**

Table 5-16 shows the PMA/PMD Package Identifier registers bit definitions.

Table 5-16: **PMA/PMD Package Identifier Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.15.15:0	PMA/PMD Package Identifier	The block always returns 0 for these bits.	R/O	All 0s
1.14.15:0	PMA/PMD Package Identifier	The block always returns 0 for these bits.	R/O	All 0s

MDIO Register 3.0: PCS Control 1

Figure 5-26 shows the MDIO Register 3.0: PCS Control 1.

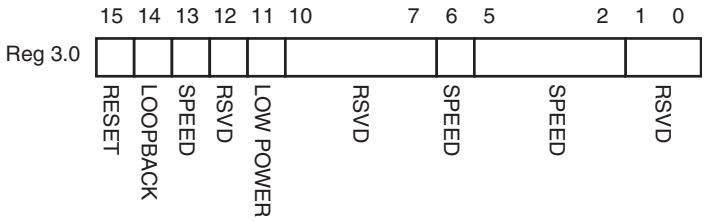

Figure 5-26: **PCS Control 1 Register**

Table 5-17 shows the PCS Control 1 register bit definitions.

Table 5-17: **PCS Control 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.0.15	Reset	1 = Block reset 0 = Normal operation The XAUI block is reset when this bit is set to 1. It returns to 0 when the reset is complete.	R/W Self-clearing	0
3.0.14	10GBASE-R Loopback	The block always returns 0 for this bit and ignores writes.	R/O	0
3.0.13	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
3.0.12	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0
3.0.11	Power down	1 = Power down mode 0 = Normal operation When set to 1, the serial transceivers are placed in a low-power state. Set to 0 to return to normal operation.	R/W	0
3.0.10:7	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s

Table 5-17: **PCS Control 1 Register Bit Definitions *(Cont'd)***

Bit(s)	Name	Description	Attributes	Default Value
3.0.6	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
3.0.5:2	Speed Selection	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
3.0.1:0	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	All 0s

MDIO Register 3.1: PCS Status 1

Figure 5-27 shows the MDIO Register 3.1: PCS Status 1.

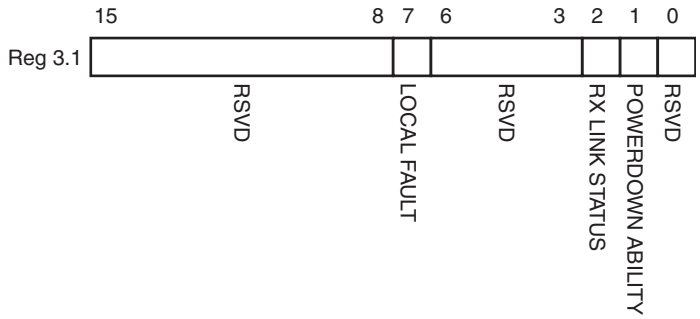

Figure 5-27: **PCS Status 1 Register**

Table 5-18 show the PCS 1 register bit definitions.

Table 5-18: **PCS Status 1 Register Bit Definition**

Bit(s)	Name	Description	Attributes	Default Value
3.1.15:8	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
3.1.7	Local Fault	1 = Local fault detected 0 = No local fault detected This bit is set to 1 whenever either of the bits 3.8.11, 3.8.10 are set to 1.	R/O	-
3.1.6:3	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
3.1.2	PCS Receive Link Status	1 = The PCS receive link is up 0 = The PCS receive link is down This is a latching Low version of bit 3.24.12.	R/O Self-setting	-

Table 5-18: **PCS Status 1 Register Bit Definition *(Cont'd)***

Bit(s)	Name	Description	Attributes	Default Value
3.1.1	Power Down Ability	The block always returns 1 for this bit.	R/O	1
3.1.0	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0

MDIO Registers 3.2 and 3.3: PCS Device Identifier

Figure 5-28 shows the MDIO Registers 3.2 and 3.3: PCS Device Identifier.

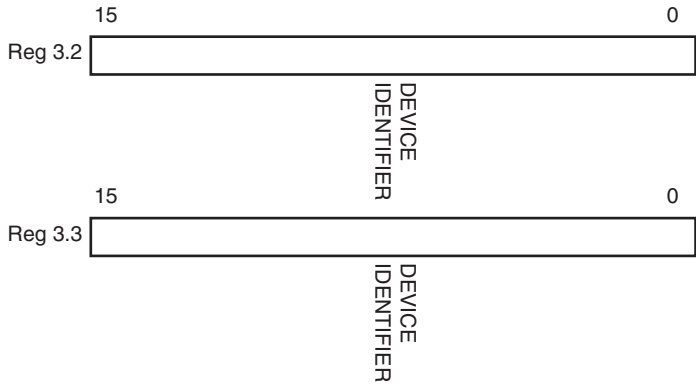

Figure 5-28: **PCS Device Identifier Registers**

Table 5-19 shows the PCS Device Identifier registers bit definitions.

Table 5-19: **PCS Device Identifier Registers Bit Definition**

Bit(s)	Name	Description	Attributes	Default Value
3.2.15:0	PCS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
3.3.15:0	PCS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s

MDIO Register 3.4: PCS Speed Ability

Figure 5-29 shows the MDIO Register 3.4: PCS Speed Ability.

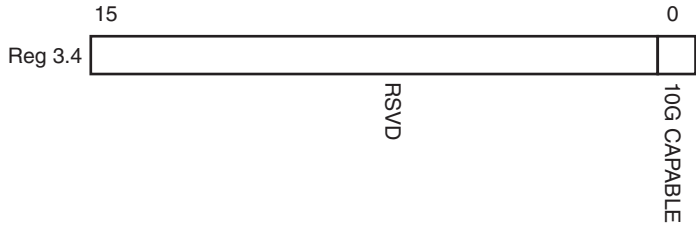

Figure 5-29: **PCS Speed Ability Register**

Table 5-20 shows the PCS Speed Ability register bit definitions.

Table 5-20: **PCS Speed Ability Register Bit Definition**

Bit(s)	Name	Description	Attribute	Default Value
3.4.15:1	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
3.4.0	10 G Capable	The block always returns 1 for this bit and ignores writes.	R/O	1

MDIO Registers 3.5 and 3.6: PCS Devices in Package

Figure 5-30 shows the MDIO Registers 3.5 and 3.6: PCS Devices in Package.

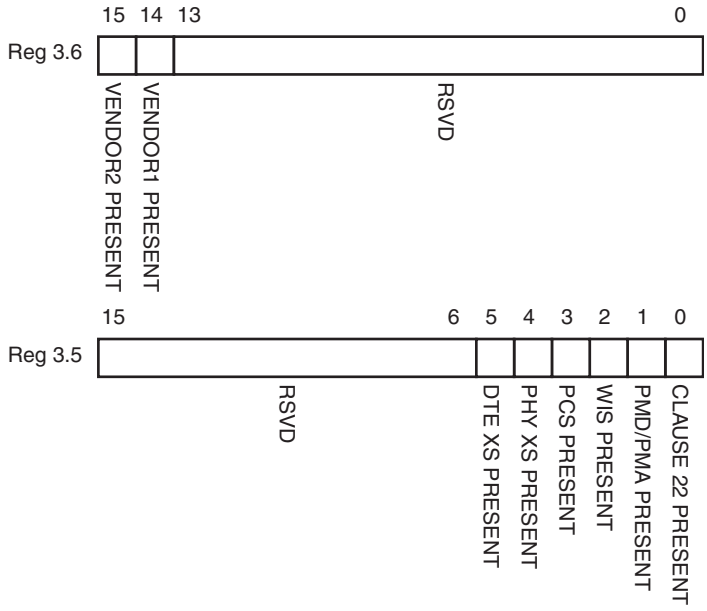

Figure 5-30: **PCS Devices in Package Registers**

Table 5-21 shows the PCS Devices in Package registers bit definitions.

Table 5-21: **PCS Devices in Package Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.6.15	Vendor-specific Device 2 Present	The block always returns 0 for this bit.	R/O	0
3.6.14	Vendor-specific Device 1 Present	The block always returns 0 for this bit.	R/O	0
3.6.13:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.5.15:6	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.5.5	PHY XS Present	The block always returns 0 for this bit.	R/O	0
3.5.4	PHY XS Present	The block always returns 0 for this bit.	R/O	0
3.5.3	PCS Present	The block always returns 1 for this bit.	R/O	1
3.5.2	WIS Present	The block always returns 0 for this bit.	R/O	0
3.5.1	PMA/PMD Present	The block always returns 1 for this bit.	R/O	1
3.5.0	Clause 22 device present	The block always returns 0 for this bit.	R/O	0

MDIO Register 3.7: 10G PCS Control 2

Figure 5-31 shows the MDIO Register 3.7: 10G PCS Control 2.

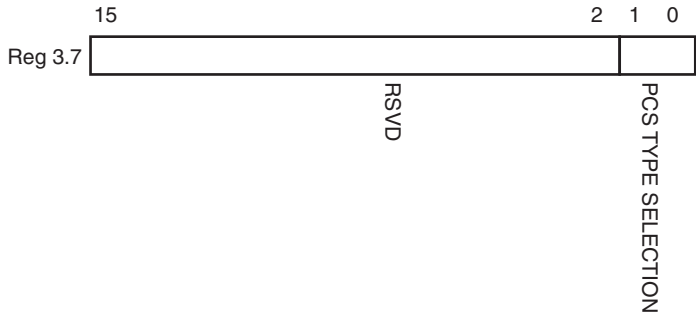

Figure 5-31: **10G PCS Control 2 Register**

Table 5-22 shows the 10 G PCS Control 2 register bit definitions.

Table 5-22: **10G PCS Control 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.7.15:2	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
3.7.1:0	PCS Type Selection	The block always returns 01 for these bits and ignores writes.	R/O	01

MDIO Register 3.8: 10G PCS Status 2

Figure 5-32 shows the MDIO Register 3.8: 10G PCS Status 2.

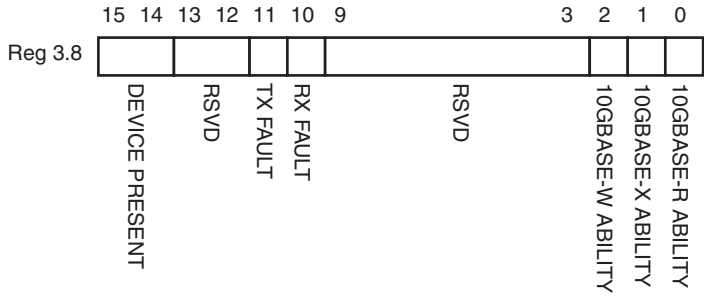

Figure 5-32: **10G PCS Status 2 Register**

Table 5-23 shows the 10G PCS Status 2 register bit definitions.

Table 5-23: **10G PCS Status 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.8.15:14	Device present	The block always returns 10.	R/O	10
3.8.13:12	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.8.11	Transmit local fault	1 = Fault condition on transmit path 0 = No fault condition on transmit path	R/O Latching High	-
3.8.10	Receive local fault	1 = Fault condition on receive path 0 = No fault condition on receive path	R/O Latching High	-
3.8.9:3	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.8.2	10GBASE-W Capable	The block always returns 0 for this bit.	R/O	0
3.8.1	10GBASE-X Capable	The block always returns 1 for this bit.	R/O	1
3.8.0	10GBASE-R Capable	The block always returns 0 for this bit.	R/O	0

MDIO Registers 3.14 and 3.15: PCS Package Identifier

Figure 5-33 shows the MDIO Registers 3.14 and 3.15: PCS Package Identifier.

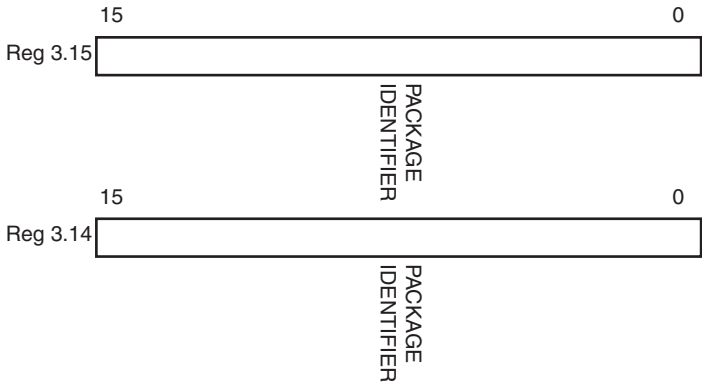

Figure 5-33: **Package Identifier Registers**

Table 5-24 shows the PCS Package Identifier registers bit definitions.

Table 5-24: **PCS Package Identifier Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.14.15:0	Package Identifier	The block always returns 0 for these bits.	R/O	All 0s
3.15.15:0	Package Identifier	The block always returns 0 for these bits.	R/O	All 0s

MDIO Register 3.24: 10GBASE-X Status

Figure 5-34 shows the MDIO Register 3.24: 10GBase-X Status.

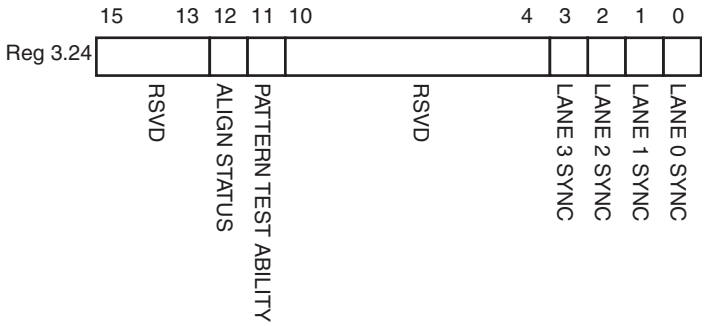

Figure 5-34: **10GBASE-X Status Register**

Table 5-25 shows the 10GBase-X Status register bit definitions.

Table 5-25: **10GBASE-X Status Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.24.15:13	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.24.12	10GBASE-X Lane Alignment Status	1 = 10GBASE-X receive lanes aligned; 0 = 10GBASE-X receive lanes not aligned.	RO	-
3.24.11	Pattern Testing Ability	The block always returns 1 for this bit.	R/O	1
3.24.10:4	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.24.3	Lane 3 Sync	1 = Lane 3 is synchronized; 0 = Lane 3 is not synchronized.	R/O	-
3.24.2	Lane 2 Sync	1 =Lane 2 is synchronized; 0 =Lane 2 is not synchronized.	R/O	-
3.24.1	Lane 1 Sync	1 = Lane 1 is synchronized; 0 = Lane 1 is not synchronized.	R/O	-
3.24.0	Lane 0 Sync	1 = Lane 0 is synchronized; 0 = Lane 0 is not synchronized.	R/O	-

MDIO Register 3.25: 10GBASE-X Test Control

Figure 5-35 shows the MDIO Register 3.25: 10GBase-X Test Control.

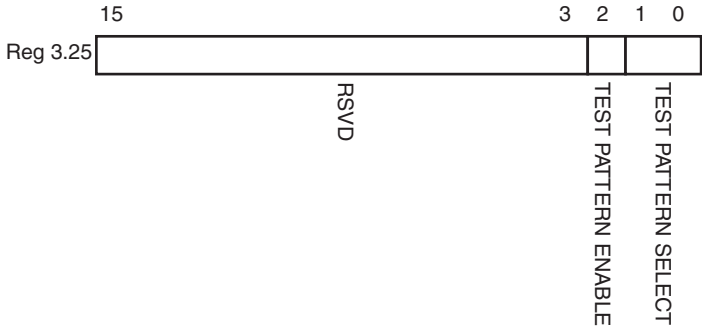

Figure 5-35: **Test Control Register**

Table 5-26 shows the 10GBase-X Test Control register bit definitions.

Table 5-26: **10GBASE-X Test Control Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.25.15:3	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.25.2	Transmit Test Pattern Enable	1 = Transmit test pattern enable 0 = Transmit test pattern disabled	R/W	0
3.25.1:0	Test Pattern Select	11 = Reserved 10 = Mixed frequency test pattern 01 = Low frequency test pattern 00 = High frequency test pattern	R/W	00

DTE XS MDIO Register Map

When the core is configured as a DTE XGXS, it occupies MDIO Device Address 5 in the MDIO register address map (Table 5-27).

Table 5-27: **DTE XS MDIO Registers**

Register Address	Register Name
5.0	DTE XS Control 1
5.1	DTE XS Status 1
5.2, 5.3	DTE XS Device Identifier
5.4	DTE XS Speed Ability
5.5, 5.6	DTE XS Devices in Package
5.7	Reserved
5.8	DTE XS Status 2
5.9 to 5.13	Reserved
5.14, 5.15	DTE XS Package Identifier
5.16 to 5.23	Reserved
5.24	10G DTE XGXS Lane Status
5.25	10G DTE XGXS Test Control

MDIO Register 5.0:DTE XS Control 1

Figure 5-36 shows the MDIO Register 5.0: DTE XS Control 1.

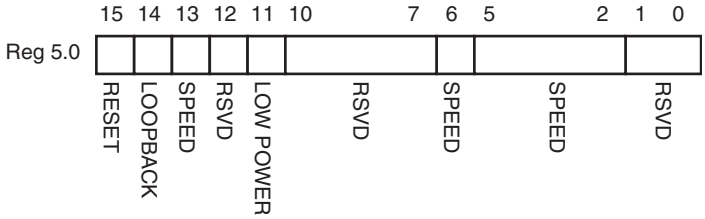

Figure 5-36: **DTE XS Control 1 Register**

Table 5-28 shows the DTE XS Control 1 register bit definitions.

Table 5-28: **DTE XS Control 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.0.15	Reset	1 = Block reset 0 = Normal operation The XAUI block is reset when this bit is set to 1. It returns to 0 when the reset is complete.	R/W Self-clearing	0
5.0.14	Loopback	1 = Enable loopback mode 0 = Disable loopback mode The XAUI block loops the signal in the serial transceivers back into the receiver. In Virtex-4 FPGA implementations it is necessary to enable/disable the TXPOST_TAP_PD bit via the GT11 DRP interface. For Virtex-5 LXT / SXT FPGA implementation it might be necessary to change GTP transceiver attributes and receiver pins under marginal conditions. See the Near-End PMA Loopback section in the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* (UG196).	R/W	0
5.0.13	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
5.0.12	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0
5.0.11	Power down	1 = Power down mode 0 = Normal operation When set to 1, the serial transceivers are placed in a low power state. Set to 0 to return to normal operation	R/W	0
5.0.10:7	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
5.0.6	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1

Table 5-28: **DTE XS Control 1 Register Bit Definitions *(Cont'd)***

Bit(s)	Name	Description	Attributes	Default Value
5.0.5:2	Speed Selection	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
5.0.1:0	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s

MDIO Register 5.1: DTE XS Status 1

Figure 5-37 shows the MDIO Register 5.1: DTE XS Status 1.

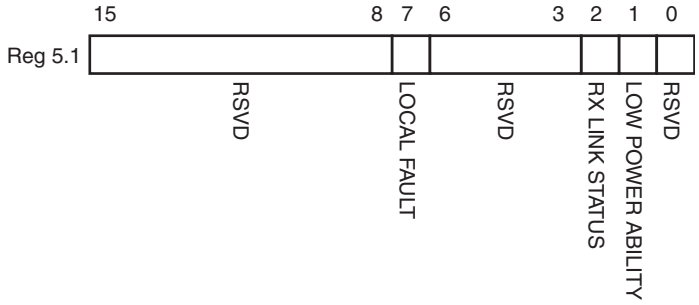

Figure 5-37: **DTE XS Status 1 Register**

Table 5-29 shows the DET XS Status 1 register bit definitions.

Table 5-29: **DTE XS Status 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.1.15:8	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
5.1.7	Local Fault	1 = Local fault detected 0 = No Local Fault detected This bit is set to 1 whenever either of the bits 5.8.11, 5.8.10 are set to 1.	R/O	-
5.1.6:3	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
5.1.2	DTE XS Receive Link Status	1 = The DTE XS receive link is up. 0 = The DTE XS receive link is down. This is a latching Low version of bit 5.24.12.	R/O Self-setting	-
5.1.1	Power Down Ability	The block always returns 1 for this bit.	R/O	1
5.1.0	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0

MDIO Registers 5.2 and 5.3: DTE XS Device Identifier

Figure 5-38 shows the MDIO Registers 5.2 and 5.3: DTE XS Device Identifier.

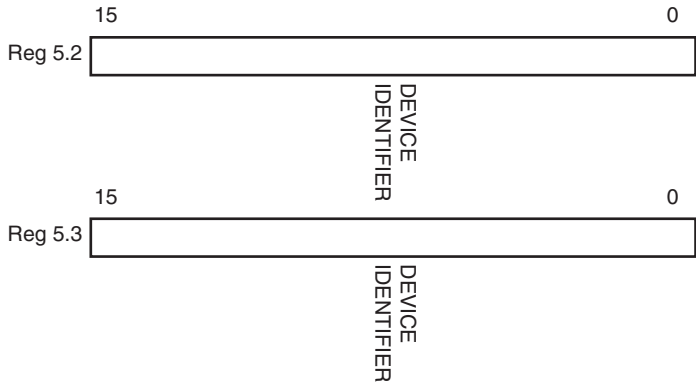

Figure 5-38: **DTE XS Device Identifier Registers**

Table 5-30 shows the DTE XS Device Identifier registers bit definitions.

Table 5-30: **DTE XS Device Identifier Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.2.15:0	DTE XS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
5.3.15:0	DTE XS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s

MDIO Register 5.4: DTE XS Speed Ability

Figure 5-39 shows the MDIO Register 5.4: DTE Speed Ability.

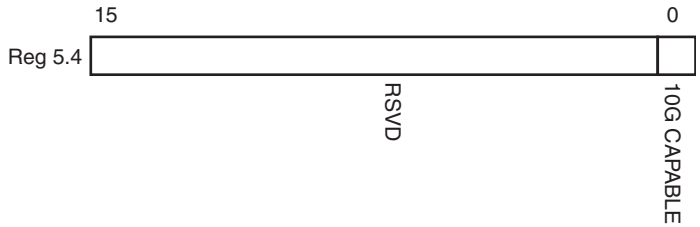

Figure 5-39: **DTE XS Speed Ability Register**

Table 5-31 shows the DTE XS Speed Ability register bit definitions.

Table 5-31: **DTE XS Speed Ability Register Bit Definitions**

Bit(s)	Name	Description	Attribute	Default Value
5.4.15:1	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
5.4.0	10G Capable	The block always returns 1 for this bit and ignores writes.	R/O	1

MDIO Registers 5.5 and 5.6: DTE XS Devices in Package

Figure 5-39 shows the MDIO Registers 5.5 and 5.6: DTE XS Devices in Package.

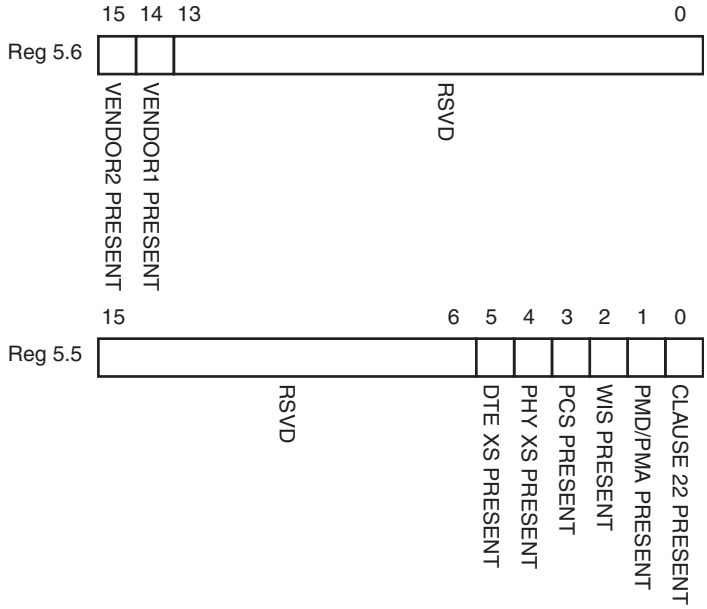

Figure 5-40: **DTE XS Devices in Package Register**

Table 5-32 shows the DTE XS Devices in Package registers bit definitions.

Table 5-32: **DTE XS Devices in Package Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.6.15	Vendor-specific Device 2 Present	The block always returns 0 for this bit.	R/O	0
5.6.14	Vendor-specific Device 1 Present	The block always returns 0 for this bit.	R/O	0
5.6.13:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.6.15:6	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.5.5	DTE XS Present	The block always returns 1 for this bit.	R/O	1
5.5.4	PHY XS Present	The block always returns 0 for this bit.	R/O	0
5.5.3	PCS Present	The block always returns 0 for this bit.	R/O	0
5.5.2	WIS Present	The block always returns 0 for this bit.	R/O	0
5.5.1	PMA/PMD Present	The block always returns 0 for this bit.	R/O	0
5.5.0	Clause 22 Device Present	The block always returns 0 for this bit.	R/O	0

MDIO Register 5.8: DTE XS Status 2

Figure 5-41 shows the MDIO Register 5.8: DTE XS Status 2.

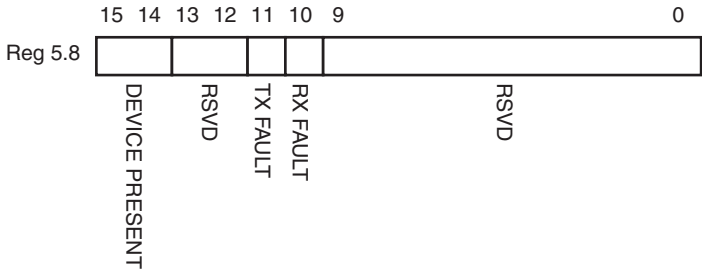

Figure 5-41: **DTE XS Status 2 Register**

Table 5-33 show the DTE XS Status 2 register bits definitions.

Table 5-33: **DTE XS Status 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.8.15:14	Device Present	The block always returns 10.	R/O	10
5.8.13:12	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.8.11	Transmit Local Fault	1 = Fault condition on transmit path 0 = No fault condition on transmit path	R/O Latching High	-
5.8.10	Receive Local Fault	1 = Fault condition on receive path 0 = No fault condition on receive path	R/O Latching High	-
5.8.9:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s

MDIO Registers 5.14 and 5.15: DTE XS Package Identifier

Figure 5-41 shows the MDIO Registers 5.14 and 5.15: DTE XS Package Identifier.

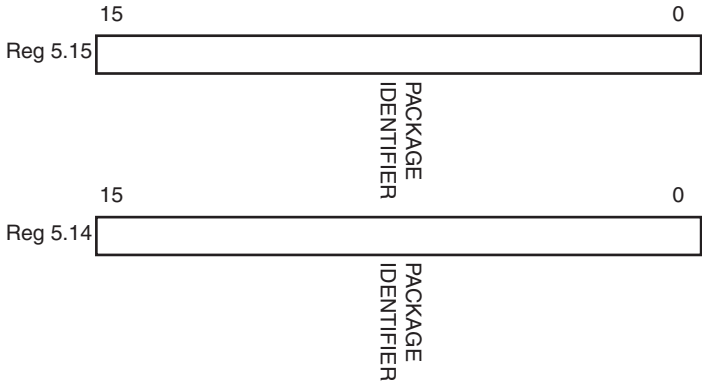

Figure 5-42: **DTE XS Package Identifier Registers**

Table 5-34 shows the DTE XS Package Identifier registers bit definitions.

Table 5-34: **DTE XS Package Identifier Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.14.15:0	DTE XS Package Identifier	The block always returns 0 for these bits.	R/O	All 0s
5.15.15:0	DTE XS Package Identifier	The block always returns 0 for these bits.	R/O	All 0s

Test Patterns

The XAUI core is capable of sending test patterns for system debug. These patterns are defined in Annex 48A of *IEEE Std. 802.3-2008* and transmission of these patterns is controlled by the MDIO Test Control Registers.

There are three types of pattern available:

- High frequency test pattern of "1010101010...." at each device-specific transceiver output
- Low frequency test pattern of "11111000001111100000111110000...." at each device-specific transceiver output
- mixed frequency test pattern of "11111010110000010100111110101100000101010..." at each device-specific transceiver output.

MDIO Register 5.24: DTE XS Lane Status

Figure 5-43 shows the MDIO Register 5.24: DTE XS Lane Status.

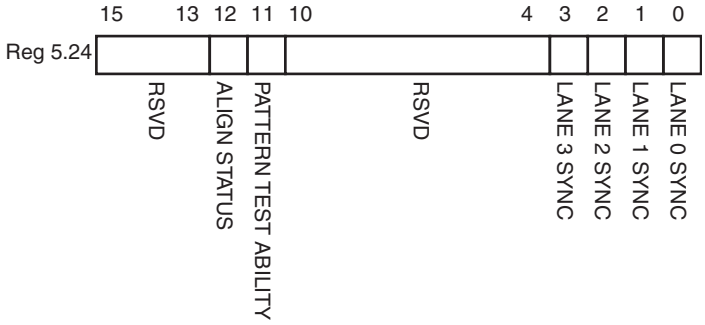

Figure 5-43: **DTE XS Lane Status Register**

Table 5-35 shows the DTE XS Lane Status register bit definitions.

Table 5-35: **DTE XS Lane Status Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.24.15:13	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.24.12	DTE XGXS Lane Alignment Status	1 = DTE XGXS receive lanes aligned 0 = DTE XGXS receive lanes not aligned	R/O	-
5.24.11	Pattern testing ability	The block always returns 1 for this bit.	R/O	1
5.24.10:4	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.24.3	Lane 3 Sync	1 = Lane 3 is synchronized; 0 = Lane 3 is not synchronized.	R/O	-
5.24.2	Lane 2 Sync	1 = Lane 2 is synchronized; 0 = Lane 2 is not synchronized.	R/O	-
5.24.1	Lane 1 Sync	1 = Lane 1 is synchronized; 0 = Lane 1 is not synchronized.	R/O	-
5.24.0	Lane 0 Sync	1 = Lane 0 is synchronized; 0 = Lane 0 is not synchronized.	R/O	-

MDIO Register 5.25: 10G DTE XGXS Test Control

Figure 5-44 shows the MDIO Register 5.25: 10G DTE XGXS Test Control.

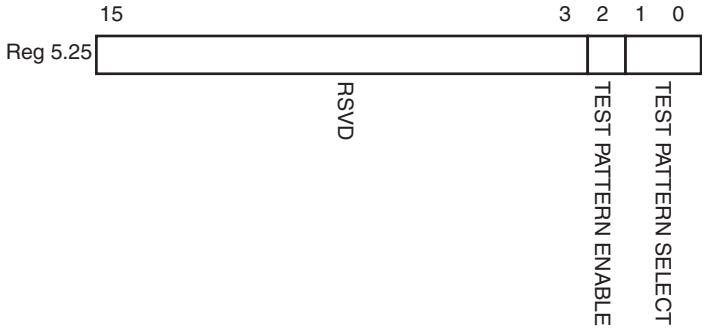

Figure 5-44: **10G DTE XGXS Test Control Register**

Table 5-36 shows the 10G DTE XGXS Test Control register bit definitions.

Table 5-36: **10G DTE XGXS Test Control Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.25.15:3	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.25.2	Transmit Test Pattern Enable	1 = Transmit test pattern enable 0 = Transmit test pattern disabled	R/W	0
5.25.1:0	Test Pattern Select	11 = Reserved 10 = Mixed frequency test pattern 01 = Low frequency test pattern 00 = High frequency test pattern	R/W	00

PHY XS MDIO Register Map

When the core is configured as a PHY XGXS, it occupies MDIO Device Address 4 in the MDIO register address map (Table 5-37).

Table 5-37: **PHY XS MDIO Registers**

Register Address	Register Name
4.0	PHY XS Control 1
4.1	PHY XS Status 1
4.2, 4.3	Device Identifier
4.4	PHY XS Speed Ability
4.5, 4.6	Devices in Package
4.7	Reserved
4.8	PHY XS Status 2
4.9 to 4.13	Reserved
4.14, 4.15	Package Identifier
4.16 to 4.23	Reserved
4.24	10G PHY XGXS Lane Status
4.25	10G PHY XGXS Test Control

MDIO Register 4.0: PHY XS Control 1

Figure 5-45 shows the MDIO Register 4.0: PHY XS Control 1.

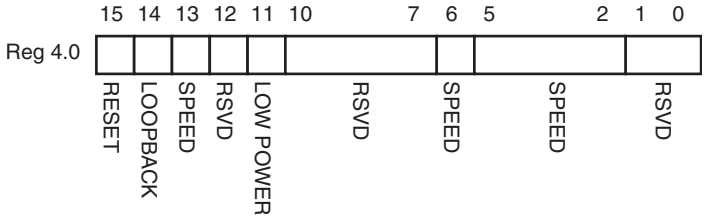

Figure 5-45: **PHY XS Control 1 Register**

Table 5-38 shows the PHY XS Control 1 register bit definitions.

Table 5-38: **PHY XS Control 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.0.15	Reset	1 = Block reset 0 = Normal operation The XAUI block is reset when this bit is set to 1. It returns to 0 when the reset is complete.	R/W Self-clearing	0
4.0.14	Loopback	1 = Enable loopback mode 0 = Disable loopback mode The XAUI block loops the signal in the serial transceivers back into the receiver. In Virtex-4 FPGA implementations it is necessary to enable/disable the TXPOST_TAP_PD bit via the GT11 DRP interface. For Virtex-5 LXT / SXT FPGA implementation it might be necessary to change GTP transceiver attributes and receiver pins under marginal conditions. See the Near-End PMA Loopback section in the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* (UG196).	R/W	0
4.0.13	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
4.0.12	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0
4.0.11	Power down	1 = Power down mode 0 = Normal operation When set to 1, the serial transceivers are placed in a low power state. Set to 0 to return to normal operation	R/W	0
4.0.10:7	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s

Table 5-38: **PHY XS Control 1 Register Bit Definitions *(Cont'd)***

Bit(s)	Name	Description	Attributes	Default Value
4.0.6	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
4.0.5:2	Speed Selection	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
4.0.1:0	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s

MDIO Register 4.1: PHY XS Status 1

Figure 5-46 shows the MDIO Register 4.1: PHY XS Status 1.

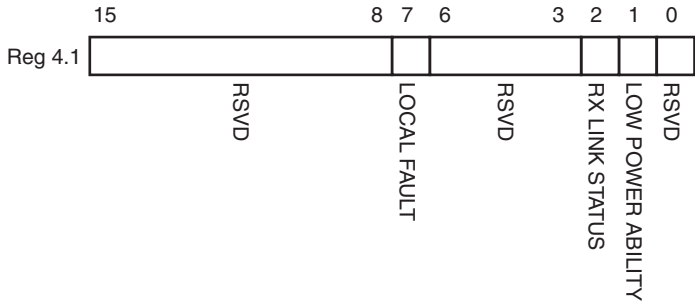

Figure 5-46: **PHY XS Status 1 Register**

Table 5-39 shows the PHY XS Status 1 register bit definitions.

Table 5-39: **PHY XS Status 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.1.15:8	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
4.1.7	Local Fault	1 = Local fault detected 0 = No Local Fault detected This bit is set to 1 whenever either of the bits 4.8.11, 4.8.10 are set to 1.	R/O	-
4.1.6:3	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
4.1.2	PHY XS Receive Link Status	1 = The PHY XS receive link is up. 0 =The PHY XS receive link is down. This is a latching Low version of bit 4.24.12.	R/O Self-setting	-
4.1.1	Power Down Ability	The block always returns 1 for this bit.	R/O	1
4.1.0	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0

MDIO Registers 4.2 and 4.3: PHY XS Device Identifier

Figure 5-47 shows the MDIO Registers 4.2 and 4.3: PHY XS Device Identifier.

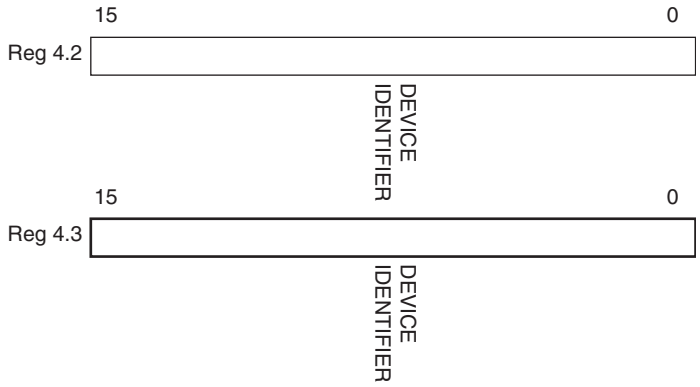

Figure 5-47: **PHY XS Device Identifier Registers**

Table 5-40 shows the PHY XS Devices Identifier registers bit definitions.

Table 5-40: **PHY XS Device Identifier Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.2.15:0	PHY XS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
4.3.15:0	PHY XS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s

MDIO Register 4.4: PHY XS Speed Ability

Figure 5-48 shows the MDIO Register 4.4: PHY XS Speed Ability.

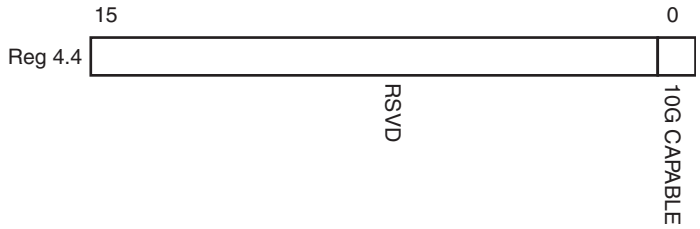

Figure 5-48: **PHY XS Speed Ability Register**

Table 5-41 shows the PHY XS Speed Ability register bit definitions.

Table 5-41: **PHY XS Speed Ability Register Bit Definitions**

Bit(s)	Name	Description	Attribute	Default Value
4.4.15:1	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
4.4.0	10G Capable	The block always returns 1 for this bit and ignores writes.	R/O	1

MDIO Registers 4.5 and 4.6: PHY XS Devices in Package

Figure 5-49 shows the MDIO Registers 4.5 and 4.6: PHY XS Devices in Package.

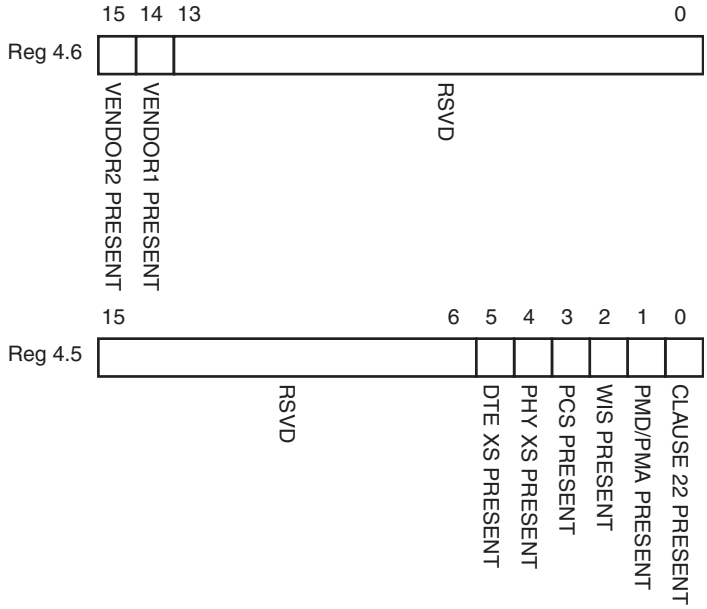

Figure 5-49: **PHY XS Devices in Package Registers**

Table 5-42 shows the PHY XS Devices in Package registers bit definitions.

Table 5-42: **PHY XS Devices in Package Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.6.15	Vendor-specific Device 2 present	The block always returns 0 for this bit.	R/O	0
4.6.14	Vendor-specific Device 1 present	The block always returns 0 for this bit.	R/O	0
4.6.13:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.5.15:6	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.5.5	DTE XS Present	The block always returns 0 for this bit.	R/O	0
4.5.4	PHY XS Present	The block always returns 1 for this bit.	R/O	1
4.5.3	PCS Present	The block always returns 0 for this bit.	R/O	0
4.5.2	WIS Present	The block always returns 0 for this bit.	R/O	0
4.5.1	PMA/PMD Present	The block always returns 0 for this bit.	R/O	0
4.5.0	Clause 22 device present	The block always returns 0 for this bit.	R/O	0

MDIO Register 4.8: PHY XS Status 2

Figure 5-50 shows the MDIO Register 4.8: PHY XS Status 2.

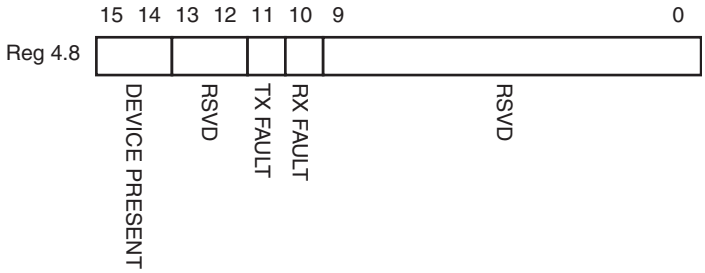

Figure 5-50: **PHY XS Status 2 Register**

Table 5-43 shows the PHY XS Status 2 register bit definitions.

Table 5-43: **PHY XS Status 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.8.15:14	Device Present	The block always returns 10.	R/O	10
4.8.13:12	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.8.11	Transmit Local Fault	1 = Fault condition on transmit path 0 = No fault condition on transmit path	R/O Latching High	-
4.8.10	Receive local fault	1 = Fault condition on receive path 0 = No fault condition on receive path	R/O Latching High	-
4.8.9:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s

MDIO Registers 4.14 and 4.15: PHY XS Package Identifier

Figure 5-51 shows the MDIO 4.14 and 4.15 Registers: PHY XS Package Identifier.

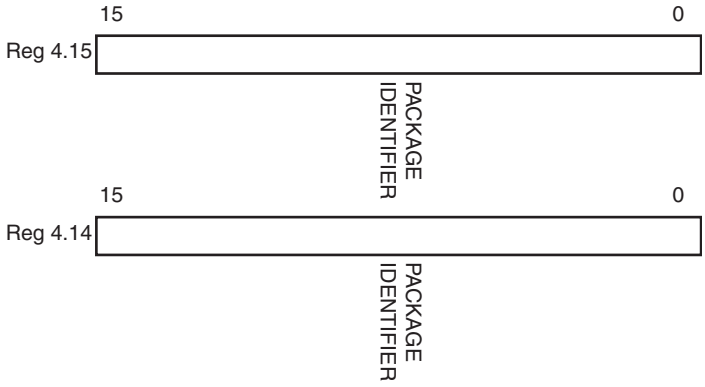

Figure 5-51: **PHY XS Package Identifier Registers**

Table 5-44 shows the Package Identifier registers bit definitions.

Table 5-44: **Package Identifier Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.15.15:0	PHY XS Package Identifier	The block always returns 0 for these bits.	R/O	All 0s
4.14.15:0	PHY XS Package Identifier	The block always returns 0 for these bits.	R/O	All 0s

MDIO Register 4.24: 10G PHY XGXS Lane Status

Figure 5-52 shows the MDIO Register 4.24: 10G XGXS Lane Status.

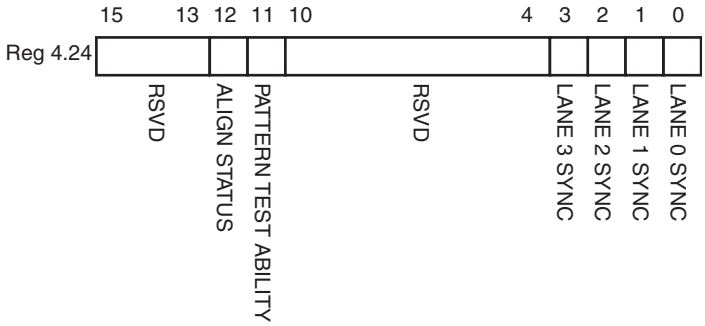

Figure 5-52: **10G PHY XGXS Lane Status Register**

Table 5-45 shows the 10G PHY XGXS Lane register bit definitions.

Table 5-45: **10G PHY XGXS Lane Status Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.24.15:13	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.24.12	PHY XGXS Lane Alignment Status	1 = PHY XGXS receive lanes aligned; 0 = PHY XGXS receive lanes not aligned.	RO	-
4.24.11	Pattern Testing Ability	The block always returns 1 for this bit.	R/O	1
4.24.10:4	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.24.3	Lane 3 Sync	1 = Lane 3 is synchronized; 0 = Lane 3 is not synchronized.	R/O	-
4.24.2	Lane 2 Sync	1 = Lane 2 is synchronized; 0 = Lane 2 is not synchronized.	R/O	-
4.24.1	Lane 1 Sync	1 = Lane 1 is synchronized; 0 = Lane 1 is not synchronized.	R/O	-
4.24.0	Lane 0 Sync	1 = Lane 0 is synchronized; 0 = Lane 0 is not synchronized.	R/O	-

MDIO Register 4.25: 10G PHY XGXS Test Control

Figure 5-53 shows the MDIO Register 4.25: 10G XGXS Test Control.

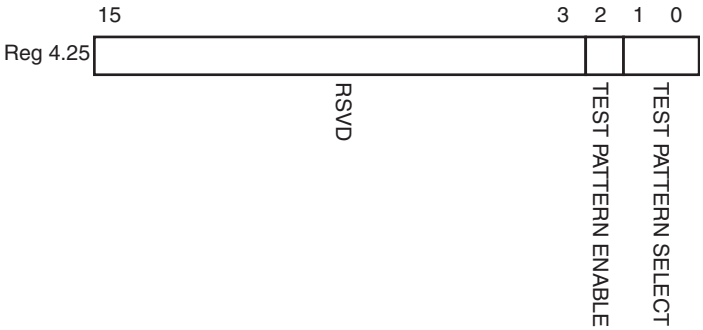

Figure 5-53: **10G PHY XGXS Test Control Register**

Table 5-46 shows the 10G PHY XGXS Test Control register bit definitions.

Table 5-46: **10G PHY XGXS Test Control Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.25.15:3	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.25.2	Transmit Test Pattern Enable	1 = Transmit test pattern enable 0 = Transmit test pattern disabled	R/W	0
4.25.1:0	Test Pattern Select	11 = Reserved 10 = Mixed frequency test pattern 01 = Low frequency test pattern 00 = High frequency test pattern	R/W	00

Configuration and Status Vectors

If the XAUI core is generated without an MDIO interface, the key configuration and status information is carried on simple bit vectors, which are:

- configuration_vector[6:0]
- status_vector[7:0]

Table 5-47 shows the Configuration Vector bit definitions.

Table 5-47: **Configuration Vector Bit Definitions**

Bit(s)	Name	Description
0	Loopback	Sets serial loopback in the device-specific transceivers. See bit 5.0.14 in Table 5-28.
1	Power Down	Sets the device-specific transceivers into power down mode. See bit 5.0.11 in Table 5-28.
2	Reset Local Fault	Clears both TX Local Fault and RX Local Fault bits (status_vector[0] and status_vector[1]). See Table 5-48. This bit should be driven by a register on the same clock domain as the XAUI core.
3	Reset Rx Link Status	Sets the RX Link Status bit (status_vector[7]). See Table 5-48. This bit should be driven by a register on the same clock domain as the XAUI core.
4	Test Enable	Enables transmit test pattern generation. See bit 5.25.2 in Table 5-36.
6:5	Test Select(1:0)	Selects the test pattern. See bits 5.25.1:0 in Table 5-36.

Table 5-48 shows the Status Vector bit definitions.

Table 5-48: **Status Vector Bit Definitions**

Bit(s)	Name	Description
0	Tx Local Fault	1 if there is a fault in the transmit path, otherwise 0; see bit 5.8.11 in Table 5-33. Latches High. Cleared by rising edge on configuration_vector[2].
1	Rx Local Fault	1 if there is a fault in the receive path, otherwise 0; see bit 5.8.10 in Table 5-33. Latches High. Cleared by rising edge on configuration_vector[2].
5:2	Synchronization	Each bit is 1 if the corresponding XAUI lane is synchronized on receive, otherwise 0; see bits 5.24.3:0 in Table 5-34. These four bits are also used to generate the sync_status[3:0] signal described in Table 5-49.
6	Alignment	1 if the XAUI receiver is aligned over all four lanes, otherwise 0; see bit 5.24.12 in Table 5-34. This is also used to generate the align_status signal described in Table 5-49.
7	Rx Link Status	1 if the Receiver link is up, otherwise 0; see bit 5.1.2 in Table 5-29. Latches Low. Cleared by rising edge on configuration_vector[3].

Bits 0 and 1 of the status_vector port, the "Local Fault" bits, are latching-high and cleared low by bit 2 of the `configuration_vector` port. Figure 5-54 shows how the status bits are cleared.

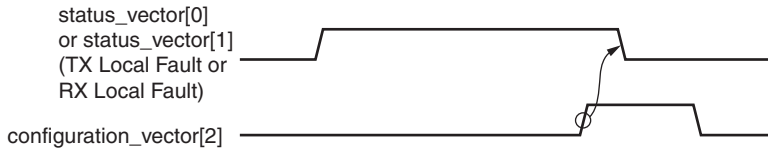

Figure 5-54: **Clearing the Local Fault Status Bits**

Bit 7 of the `status_vector` port, the "RX Link Status" bit, is latching-low and set high by bit 3 of the configuration vector. Figure 5-55 shows how the status bit is set.

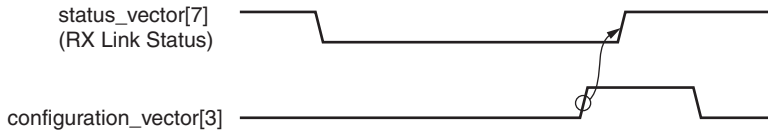

Figure 5-55: **Setting the RX Link Status Bit**

Alignment and Synchronization Status Ports

In addition to the configuration and status interfaces described in the previous section, there are always available two output ports signalling the alignment and synchronization status of the receiver. (Table 5-49.)

Table 5-49: **Alignment Status and Synchronization Status Ports**

Port Name	Description
align_status	1 when the XAUI receiver is aligned across all four lanes, 0 otherwise.
sync_status[3:0]	Each pin is 1 when the respective XAUI lane receiver is synchronized to byte boundaries, 0 otherwise.

XILINX®

Chapter 6

Constraining the Core

This chapter describes how to constrain a design containing the XAUI core. This is illustrated by the UCF delivered with the core at generation time. See Chapter 10, Detailed Example Design for a complete description of the Xilinx® CORE Generator™ tool output files.

> ***Caution!*** Not all constraints are relevant to specific implementations of the core; consult the UCF created with the core instance to see exactly what constraints are relevant.

Device, Package, and Speed Grade Selection

This line selects the part to be used in the implementation run. Change this line so that it matches the part intended for the final application.

```
# Select the part to be used in the implementation run
CONFIG PART = xc5vlx110t-ff1136-1;
```

The XAUI core can be implemented in the following Xilinx devices:

- Virtex®-7 devices
- Kintex™-7 devices
- All Virtex-6 CXT/LXT/SXT/HXT devices
- All Virtex-5 LXT/SXT/FXT/TXT devices
- Spartan®-6 LXT FPGA Family, XC6SLX45T and larger, with a speed grade of -3 or higher
- Virtex-4 FX FPGA Family, XC4VFX20 and larger, with speed grade of -10 or higher

The 20G-XAUI core can be implemented in the following Xilinx devices:

- Virtex-6 devices with a speed grade of -3.
- Virtex-7 and Kintex-7 devices with a speed grade of -2 or higher

Clock Frequencies, Clock Management, and Placement

The XAUI core can have one or two clock domains:

- The `refclk` domain derived from the `TXOUTCLK1` output of the Virtex-4 FX FPGA, serial transceiver derived or from the `REFCLK_OUT` output of the Virtex-5 FPGA GTP or GTX transceiver or from the `TXOUTCLK` output of the Virtex-6 FPGA GTX, Virtex-7 FPGA GTX or Kintex-7 FPGA GTX transceiver or from the `GTPCLKOUT` output of the Spartan-6 FPGA GTP transceiver.
- Optionally, the `xgmii_tx_clk` domain from an inbound XGMII clock

This section specifies the main clock frequencies for the design and sets the attributes for any Digital Clock Manager (DCM) or Mixed-Mode Clock Manager (MMCM) primitives included in the design.

Virtex-7, Kintex-7, Virtex-6, Spartan-6, Virtex-5 and Virtex-4 FX FPGAs

```
##################################################################
# Clock frequencies and clock management #
##################################################################
```

For 10G-XAUI:

```
NET "*txoutclk*" TNM_NET="clk156_top"; TIMESPEC "TS_clk156_top" =
PERIOD "clk156_top" 156.25MHz;
```

For 20G-XAUI:

```
NET "*txoutclk*" TNM_NET="clk156_top"; TIMESPEC "TS_clk156_top" =
PERIOD "clk156_top" 312.5MHz;
```

The clock frequency by default is set for 10-Gigabit Ethernet; increasing the frequency to 159.375 MHz as directed in the UCF comment raises the maximum clock frequency to that needed for 10-Gigabit Fibre Channel and equates to a device-specific transceiver serial rate of 3.1875 Gb/s per lane.

General Clocking

These constraints set up the basic operating attributes of the system DCMs.

```
#########################################################
#                              #
# ****** Please CHECK these constraints. *****   #
#                             #
# Please check this constraint with reference to a#
# static timing report which should be generated  #
# after Place and Route. If the setup and hold   #
# times for the XGMII inputs are not met then   #
# please adjust the PHASE_SHIFT value as explained#
# in the XAUI User Guide. #
#########################################################
INST "dcm_txclk" DUTY_CYCLE_CORRECTION = TRUE;
INST "dcm_txclk" DESKEW_ADJUST = SOURCE_SYNCHRONOUS;
INST "dcm_txclk" PHASE_SHIFT = "0";
```

For Virtex-7, Kintex-7 and Virtex-6 FPGA MMCMs:

```
INST "mmcm_clk156" CLKOUT0_PHASE = "0";
```

This constraint sets a phase shift on the main output of the system DCM/MMCM with respect to the input clock; this is used to obtain clock/data alignment on the inbound XGMII interface. See Chapter 7, Design Considerations for a description of the clock scheme and Appendix C, Calculating the DCM/MMCM Phase Shift for information about setting the phase-shift value. These constraints are only present in the example if the external XGMII is used.

For 10G-XAUI:

```
NET "xgmii_tx_clk" TNM_NET="xgmii_tx_clk";
TIMESPEC "TS_xgmii_tx_clk" = PERIOD "xgmii_tx_clk" 156.25 MHz;
```

For 20G-XAUI:

```
NET "xgmii_tx_clk" TNM_NET="xgmii_tx_clk";
TIMESPEC "TS_xgmii_tx_clk" = PERIOD "xgmii_tx_clk" 312.5 MHz;
```

If the external XGMII interface is used, the inbound XGMII clock frequency must also be constrained. The previous discussion of the Fibre Channel clock frequency increase also applies to this clock.

Transceiver Placement

Virtex-7 and Kintex-7 FPGAs

```
INST xaui_block/gtx_wrapper_i/gtxe2_common_0_i LOC=GTXE2_COMMON_X0Y0

INST xaui_block/gtx_wrapper_i/gt0_gt_wrapper_i/gtxe2_i LOC=GTXE2_CHANNEL_X0Y0
INST xaui_block/gtx_wrapper_i/gt1_gt_wrapper_i/gtxe2_i LOC=GTXE2_CHANNEL_X0Y1
INST xaui_block/gtx_wrapper_i/gt2_gt_wrapper_i/gtxe2_i LOC=GTXE2_CHANNEL_X0Y2
INST xaui_block/gtx_wrapper_i/gt3_gt_wrapper_i/gtxe2_i LOC=GTXE2_CHANNEL_X0Y3
```

Virtex-6 FPGAs

```
INST xaui_block/gtx_wrapper_i/gtx0_gtx_wrapper_i/gtxe1_i LOC=GTXE1_X0Y0
INST xaui_block/gtx_wrapper_i/gtx1_gtx_wrapper_i/gtxe1_i LOC=GTXE1_X0Y1
INST xaui_block/gtx_wrapper_i/gtx2_gtx_wrapper_i/gtxe1_i LOC=GTXE1_X0Y2
INST xaui_block/gtx_wrapper_i/gtx3_gtx_wrapper_i/gtxe1_i LOC=GTXE1_X0Y3
```

These constraints lock down the placement of the device-specific transceivers.

Spartan-6 LXT FPGAs

```
INST xaui_block/gtp_wrapper_i/tile0_gtp_wrapper_i/gtpa1_dual_i LOC = GTPA1_DUAL_X0Y0
INST xaui_block/gtp_wrapper_i/tile1_gtp_wrapper_i/gtpa1_dual_i LOC = GTPA1_DUAL_X1Y0
```

Virtex-5 FXT/TXT FPGAs

```
INST xaui_block/rocketio_wrapper_i/tile0_rocketio_wrapper_i/gtx_dual_i LOC=GTX_DUAL_X0Y0;
INST xaui_block/rocketio_wrapper_i/tile1_rocketio_wrapper_i/gtx_dual_i LOC=GTX_DUAL_X0Y1;
```

These constraints lock down the placement of the device-specific RocketIO™ transceivers. There are two tiles constrained, giving a total of four transceivers.

Virtex-5 LXT/SXT FPGAs

```
INST xaui_block/rocketio_wrapper_i/tile0_rocketio_wrapper_i/gtp_dual_i LOC=GTP_DUAL_X0Y0;
INST xaui_block/rocketio_wrapper_i/tile1_rocketio_wrapper_i/gtp_dual_i LOC=GTP_DUAL_X0Y1;
```

These constraints lock down the placement of the device-specific RocketIO transceivers. There are two tiles constrained, giving a total of four transceivers.

Virtex-4 FPGAs

```
INST xaui_block/rocketio_wrapper_i/MGT0 LOC=GT11_X0Y4;
INST xaui_block/rocketio_wrapper_i/MGT1 LOC=GT11_X0Y5;
INST xaui_block/rocketio_wrapper_i/MGT2 LOC=GT11_X0Y6;
INST xaui_block/rocketio_wrapper_i/MGT3 LOC=GT11_X0Y7;
```

These constraints lock down the placement of the device-specific RocketIO transceivers.

XGMII

```
# XGMII input/output buffer attributes
NET xgmii_txc<?>* IOSTANDARD = HSTL_I;
NET xgmii_txd<?>* IOSTANDARD = HSTL_I;
NET xgmii_txd<??>* IOSTANDARD = HSTL_I;
NET xgmii_rxc<?>* IOSTANDARD = HSTL_I;
NET xgmii_rxd<?>* IOSTANDARD = HSTL_I;
NET xgmii_rxd<??>* IOSTANDARD = HSTL_I;
```

These lines set the Input/Output (I/O) attributes for the XGMII Input/Output buffers (IOBs).

Transmit Elastic Buffer

```
NET "*xaui_core/BU2/U0/*elastic_buffer_i/asynch_fifo_i/rd_truegray<?>" MAXDELAY = 6.0 ns;
NET "*xaui_core/BU2/U0/*elastic_buffer_i/can_insert_wra" TIG;
NET "*xaui_core/BU2/U0/*elastic_buffer_i/asynch_fifo_i/wr_gray<?>" MAXDELAY = 6.0 ns;
NET "*xaui_core/BU2/U0/*elastic_buffer_i/asynch_fifo_i/rd_lastgray<?>" MAXDELAY = 6.0 ns;
```

If the Transmit Elastic Buffer is used, these constraints are required to cross the clock domain cleanly.

MDIO

```
##################################################################
# MDIO-related constraints #
##################################################################
NET "*xaui_core/BU2/U0/*management_1/mdc_reg1" IOB=TRUE;
NET "*xaui_core/BU2/U0/*management_1/mdio_in_reg1" IOB=TRUE;
NET "mdio_out" IOB=TRUE;
NET "mdio_tri" IOB=TRUE;
```

(The preceding four constraints are for Virtex-4 devices only)

```
NET "*xaui_core/BU2/U0/*management_1/mdc_rising*" TNM_NET =
"xaui_mdc_grp";
INST "*xaui_core/BU2/U0/type_sel_reg_1" TNM = FFS "xaui_mdc_grp";
TIMESPEC "TS_XAUI_mdc" = FROM "xaui_mdc_grp" to "xaui_mdc_grp" 400 ns;
```

These constraints set the correct attributes for the registers at the edge of the MDIO block. The TIMESPEC constraints the MDIO interface to 2.5 MHz. If you wish to overclock the MDIO interface, you must alter this constraint.

XILINX®

Chapter 7

Design Considerations

This chapter describes considerations that might apply in particular design cases.

Clocking: Virtex-7 and Kintex-7 FPGAs

The clocking schemes in this section are illustrative only and might require customization for a specific application.

Reference Clock

10G-XAUI

For Virtex®-7 and Kintex™-7 FPGA GTX transceivers, the transceivers typically use a reference clock of 156.25 MHz to operate at a line rate of 3.125 Gb/s. To use a reference clock of 312.5 MHz:

1. Run the GTX Transceiver Wizard. Select the XAUI protocol and a reference clock of 312.5 MHz and set `TXOUTCLK` source to `TXPLLREFCLK_DIV2`. The GTX transceiver is configured to divide by 2 on the `TXOUTCLK` output.
2. Copy the output file files gt_wrapper_gt.v[hd] and gt_wrapper.v[hd] to the XAUI example_design directory.

20G-XAUI

For Virtex®-7 and Kintex-7 FPGA GTX transceivers, the GTX transceivers typically use a reference clock of 312.5 MHz to operate at a line rate of 6.25 Gb/s.

It is also possible to use a reference clock of 156.25 MHz to operate at a line rate of 6.25 Gb/s. To use a reference clock of 156.25 MHz:

1. Run the GTX Transceiver Wizard. Select the XAUI protocol and a reference clock of 156.25 MHz. Use the generated wrapper files in your design.
2. Implement the necessary clocking circuitry to convert the 156.25 MHz `TXOUTCLK` to 312.5 MHz for use by the core.

Transceiver Placement

Common to all schemes shown is that a single IBUFDS_GTXE2 block is used to feed the reference clock to GTXE2_COMMON transceiver Quad PLL (QPLL).

For more information about 7 series FPGA transceiver clock distribution, see the section on Clocking in the *7 Series FPGAs Transceivers User Guide* (UG476).

Internal Client-Side Interface for 10G - XAUI (Virtex-7 and Kintex-7 FPGAs)

The simplest clocking scheme is for the internal client interface, as shown in Figure 7-1.

A 156.25 MHz clock derived from the GTX transceiver TXOUTCLK port is used as the clock for the netlist part of the XAUI core and is typically also used for your logic.

A dedicated clock DCLK is used by the GTX transceiver tiles. The example design uses a 50 MHz clock. Choosing a different frequency allows sharing of clock resources. See the *7 Series FPGAs Transceivers User Guide* for more information about this clock.

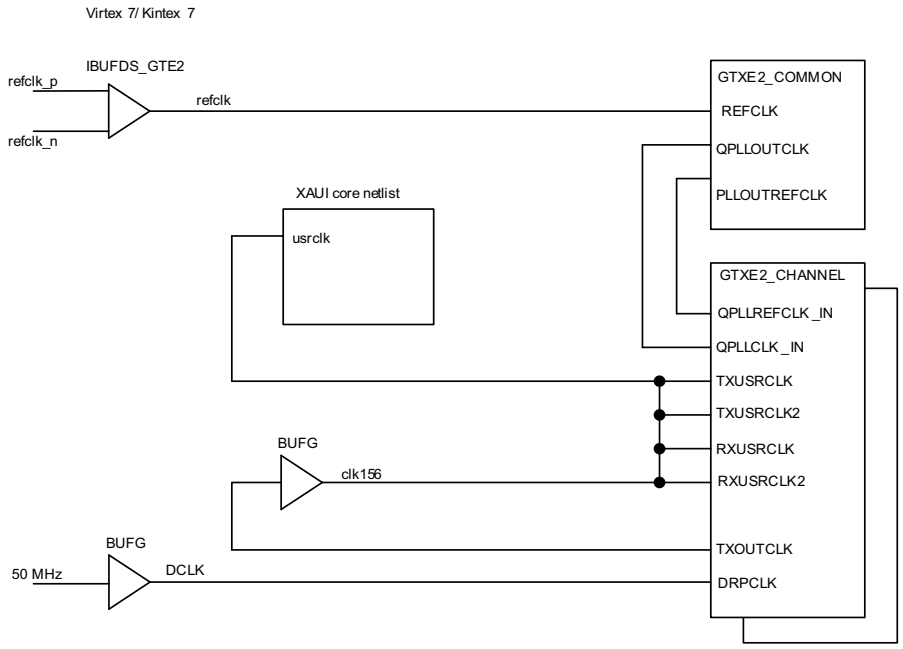

Figure 7-1: **Clock Scheme for Internal Client-Side Interface: Virtex-7 and Kintex-7 FPGAs**

Internal Client-Side Interface for 20G - XAUI (Virtex-7 and Kintex-7 FPGAs)

The simplest clocking scheme is for the internal client interface, as shown in Figure 7-1.

A 312.5 MHz clock derived from the GTX transceiver TXOUTCLK port is used as the clock for the netlist part of the XAUI core and is typically also used for your logic.

A dedicated clock is used by the GTX transceiver tiles. The example design uses a 50 MHz clock. Choosing a different frequency allows sharing of clock resources. See the *7-Series FPGAs GTX Transceiver User Guide* for more information about this clock.

External XGMII Interface: Transmit Elastic Buffer (Virtex-7 and Kintex-7 FPGAs)

Often, it cannot be arranged that the inbound transmit data clock domain is derived from the reference clock. In these circumstances, the optional transmit elastic buffer in the core must be used. Figure 7-2 shows a clocking scheme for this case.

The `tx_clk` pin of the XAUI core is used to clock the input side of an elastic buffer in the transmit path; `usrclk` is used to clock the output side.

The `clk156` and `clk156_90` signals are used to clock DDR output registers in the correct phasing for XGMII signaling. The `tx_clk0` signal is used to clock DDR input registers.

A dedicated clock `DCLK` is used by the GTX transceiver tiles. The example design uses a 50 MHz clock. You might choose to use a different frequency to allow sharing of clock resources. See the *7 Series FPGAs Transceivers User Guide* for more information about this clock.

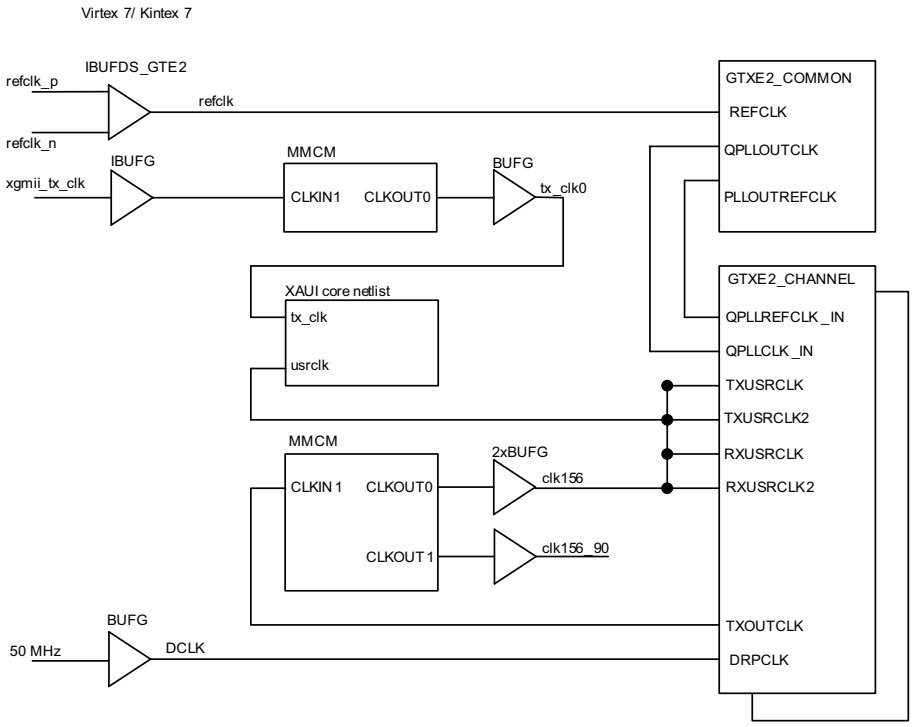

Figure 7-2: **Clock Scheme for External XGMII Client-Side Interface with Transmit Elastic Buffer: Virtex-7 and Kintex-7 FPGAs**

Clocking: Virtex-6 FPGAs

The clocking schemes in this section are illustrative only and might require customization for a specific application.

Reference Clock

10G-XAUI

For Virtex-6 FPGAs, the GTX transceivers typically use a reference clock of 156.25 MHz to operate at a line rate of 3.125 Gb/s. To use a reference clock of 312.5 MHz:

1. Run the Virtex-6 FPGA GTX Transceiver Wizard. Select the XAUI protocol and a reference clock of 312.5 MHz. The GTXE1 is configured to divide by 2 on the `TXOUTCLK` output.
2. Copy the output file gtx_wrapper_gtx.v[hd] to the XAUI example_design directory.

20G-XAUI

For Virtex-6 FPGAs, the GTX transceivers typically use a reference clock of 312.5 MHz to operate at a line rate of 6.25 Gb/s.

It is also possible to use a reference clock of 156.25 MHz to operate at a line rate of 6.25 Gb/s. To use a reference clock of 312.5 MHz:

1. Run the Virtex-6 FPGA GTX Transceiver Wizard. Select the XAUI protocol and a reference clock of 156.25 MHz. Use the generated wrapper files in your design.
2. Implement the necessary clocking circuitry to convert the 156.25 MHz `TXOUTCLK` to 312.5 MHz for use by the core.

Transceiver Placement

Common to all schemes shown is that a single IBUFDS_GTXE1 block is used to feed the reference clocks for all GTXE1 transceivers. In addition, timing requirements are more easily met if all four transceivers are placed next to each other within the column.

For more information about Virtex-6 FPGA transceiver clock distribution, see the section on Clocking in the *Virtex-6 FPGA GTX Transceiver User Guide (UG366)*.

Internal Client-Side Interface for 10G-XAUI (Virtex-6 FPGAs)

The simplest clocking scheme is for the internal client interface, as shown in Figure 7-3.

The GTX transceiver primitives require a 156.25 MHz clock. The 156.25 MHz clock from the GTX transceiver `TXOUTCLK` port is used as the clock for the netlist part of the XAUI core and is typically also used for your logic.

A dedicated clock is used by the GTX transceiver tiles. The example design uses a 50 MHz clock. Choosing a different frequency allows sharing of clock resources. See the *Virtex-6*

FPGA GTX Transceiver User Guide for more information about this clock. This clock can be omitted if the GTX transceiver DRP bus is not used in your design.

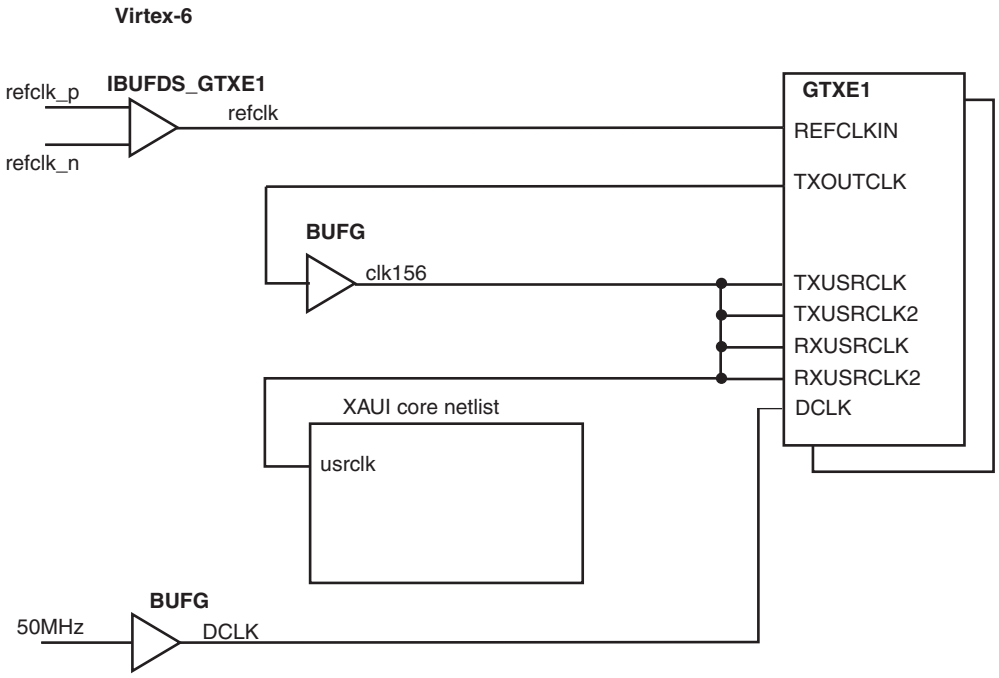

Figure 7-3: **Clock Scheme for Internal Client-Side Interface: Virtex-6 FPGAs**

Internal Client-Side Interface for 20G-XAUI (Virtex-6 FPGAs)

The simplest clocking scheme is for the internal client interface, as shown in Figure 7-3.

The GTX transceiver primitives require a 312.5 MHz clock. The 312.5 MHz clock derived from the GTX transceiver `TXOUTCLK` port is used as the clock for the netlist part of the XAUI core and is typically also used for your logic.

A dedicated clock is used by the GTX transceiver tiles. The example design uses a 50 MHz clock. Choosing a different frequency allows sharing of clock resources. See the *Virtex-6 FPGA GTX Transceiver User Guide* for more information about this clock.

External XGMII Interface: Transmit Elastic Buffer (Virtex-6 FPGAs)

Often, it cannot be arranged that the inbound transmit data clock domain is derived from the reference clock. In these circumstances, the optional transmit elastic buffer in the core must be used. Figure 7-4 shows a clocking scheme for this case.

The `tx_clk` pin of the XAUI core is used to clock the input side of an elastic buffer in the transmit path; `usrclk` is used to clock the output side.

The `clk156` and `clk156_90` signals are used to clock DDR output registers in the correct phasing for XGMII signaling. The `tx_clk0` signal is used to clock DDR input registers.

A dedicated clock is used by the GTX transceiver tiles. The example design uses a 50 MHz clock. You might choose to use a different frequency to allow sharing of clock resources. See the *Virtex-6 FPGA GTX Transceiver User Guide* for more information about this clock.

DCLK can be omitted if the design does not use the GTX transceiver DRP bus.

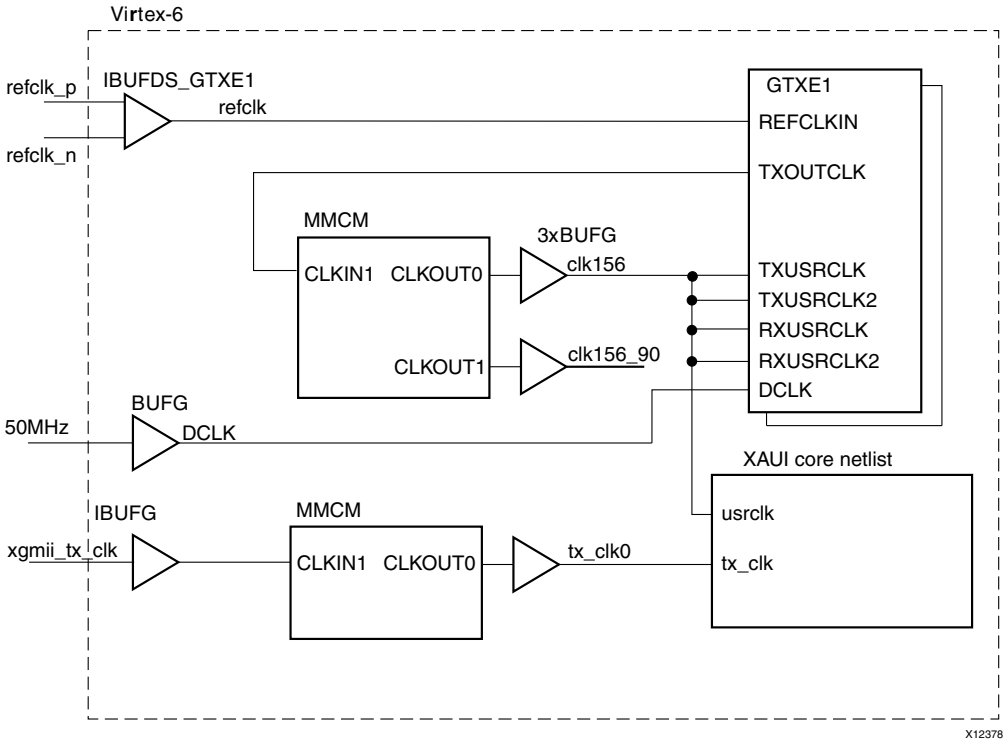

Figure 7-4: **Clock Scheme for External XGMII Client-Side Interface with Transmit Elastic Buffer: Virtex-6 FPGAs**

Clocking: Spartan-6 LXT FPGAs

The clocking schemes in this section are illustrative only and might require customization for a specific application.

Reference Clock

The GTP transceivers require a reference clock of 156.25 MHz to operate at a line rate of 3.125 Gb/s.

Transceiver Placement

A single IBUFDS is used to feed the reference clocks for both GTP transceiver tiles; as a result of this and additional limitations, both tiles *must* be in a single row.

For more information about Spartan®-6 FPGA transceiver clock distribution, see the section on Clocking in the *Spartan-6 FPGA GTP Transceiver User Guide*.

Internal Client-Side Interface

The simplest clocking scheme for the internal client interface, as shown in Figure 7-5. The GTPA1_DUAL primitives require a 156.25 MHz clock and a 312.5 MHz clock; these are generated by a DCM. The 156.25 MHz clock from the DCM is used as the clock for the netlist part of the XAUI core and is typically also used for your logic.

A dedicated clock is used by the GTP transceiver tiles. The example design uses a 50 MHz clock. You might choose to use a different frequency to allow sharing of clock resources.

DCLK can be omitted if the design does not use the GTP transceiver DRP bus. See the *Spartan-6 FPGA GTP Transceiver User Guide* for more information about this clock.

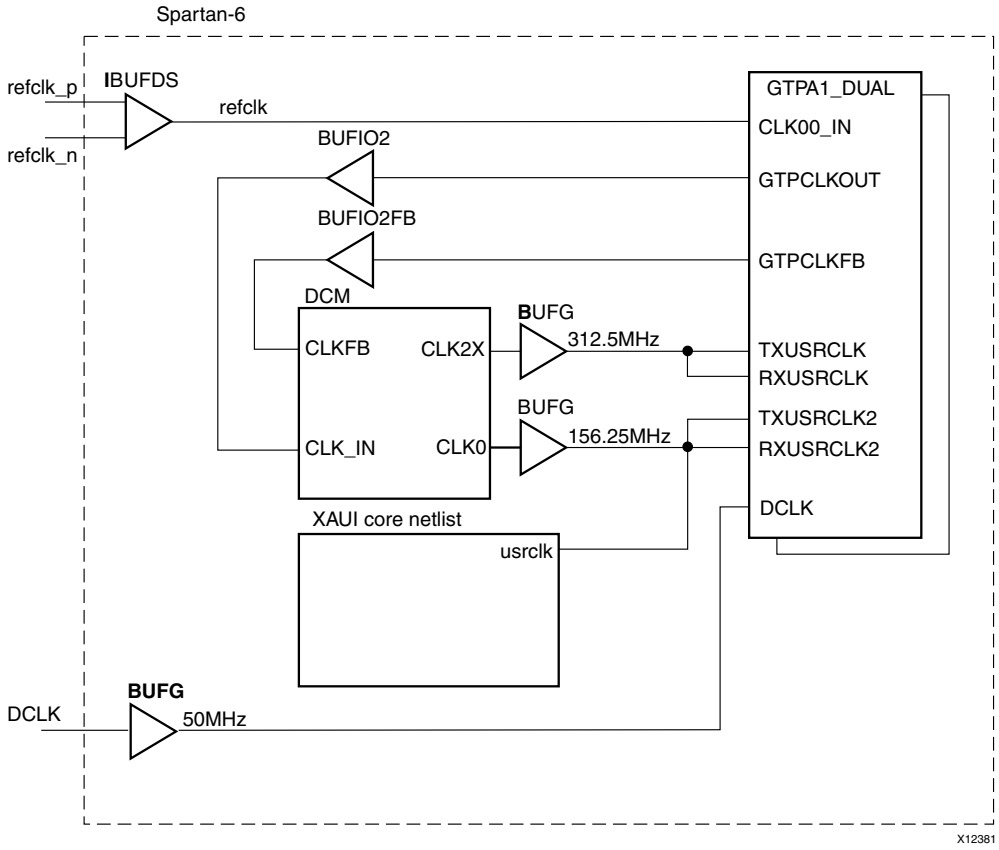

Figure 7-5: **Clock Scheme for Internal Client-Side Interface: Spartan-6 LXT FPGAs**

Clocking: Virtex-5 FPGAs

The clocking schemes in this section are illustrative only and may require customization for a specific application.

Reference Clock

The GTP transceivers require a reference clock of 156.25 MHz to operate at a line rate of 3.125 Gb/s.

The GTX transceivers require a reference clock of 156.25 MHz or 312.5 MHz to operate at a line rate of 3.125 Gb/s.

The XAUI core uses a default of 156.25 MHz. To change to a 312.5 MHz reference clock for Virtex-5 FPGA GTX transceivers the following is required:

1. Regenerate the RocketIO transceiver wrappers using the Virtex-5 FPGA RocketIO GTX Transceiver Wizard. Select the XAUI protocol template and change the reference clock to 312.5 MHz. Enable the Loss of Sync State machine if required.
2. Copy the rocketio_wrapper_tile.v[hd] file from the wizard outputs to XAUI example design directory

3. Use a Digital Clock Manager/Phase-Locked Loop (DCM/PLL) to generate a 156.25 MHz clock to the XAUI block level input 'clk156' from the 312.5 MHz XAUI block level output 'txoutclk'

Transceiver Placement

Common to all schemes shown is that a single IBUFDS block is used to feed the reference clocks for both GTP/GTX transceiver tiles; as a result, both tiles *must* be in a single column. In addition, timing requirements benefit if the two tiles are placed next to each other within the column.

For more information about Virtex-5 FPGA RocketIO transceiver clock distribution, see the section on Clocking in the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* or *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*.

Internal Client-Side Interface (Virtex-5 LXT/SXT FPGAs)

The simplest clocking scheme is for the internal client interface, as shown in Figure 7-6. The GTP transceiver primitives require a 156.25 MHz clock and a 312.5 MHz clock; these are generated by a DCM. The 156.25 MHz clock from the DCM is used as the clock for the netlist part of the XAUI core and is typically also used for your logic.

A dedicated clock is used by the GTP transceiver tiles. The example design uses a 50 MHz clock. You might choose to use a different frequency to allow sharing of clock resources. See the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* for more information about this clock.

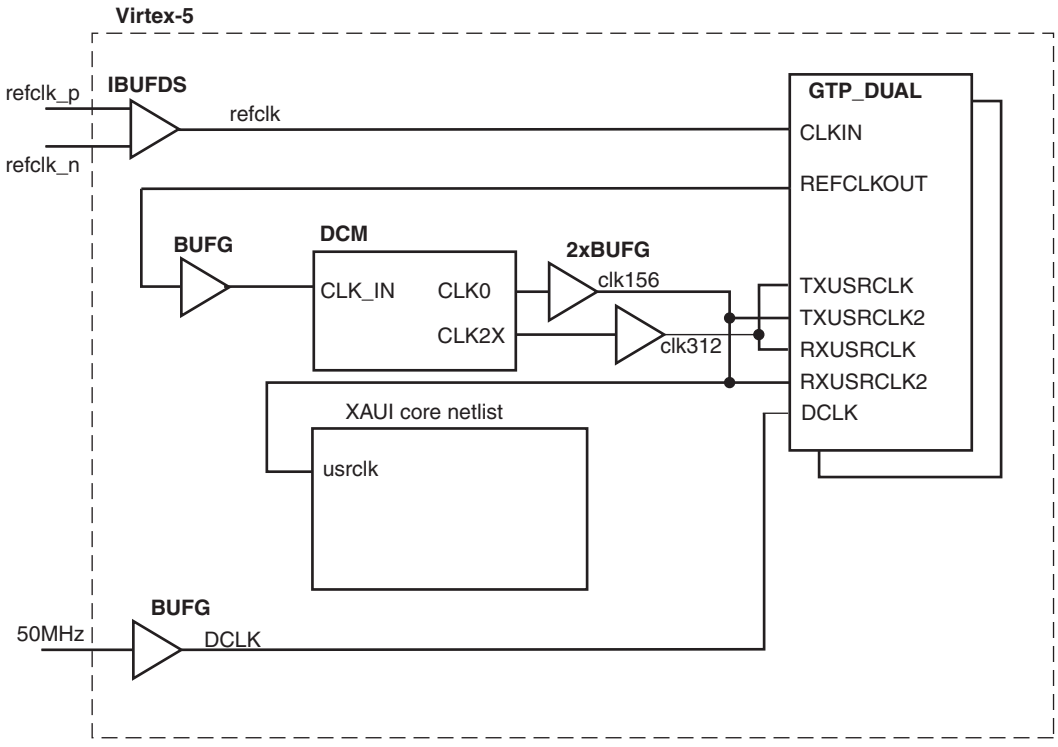

Figure 7-6: **Clock Scheme for Internal Client-Side Interface: Virtex-5 LXT/SXT FPGAs**

Internal Client-Side Interface (Virtex-5 FXT/TXT FPGAs)

The simplest clocking scheme is for the internal client interface, as shown in Figure 7-7.

The GTX transceiver primitives require a 156.25 MHz clock. The 156.25 MHz clock from the GTX transceiver `REFCLKOUT` port is used as the clock for the netlist part of the XAUI core and is typically also used for your logic.

A dedicated clock is used by the GTX transceiver tiles. The example design uses a 50 MHz clock. Choosing a different frequency allows sharing of clock resources. See the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* for more information about this clock.

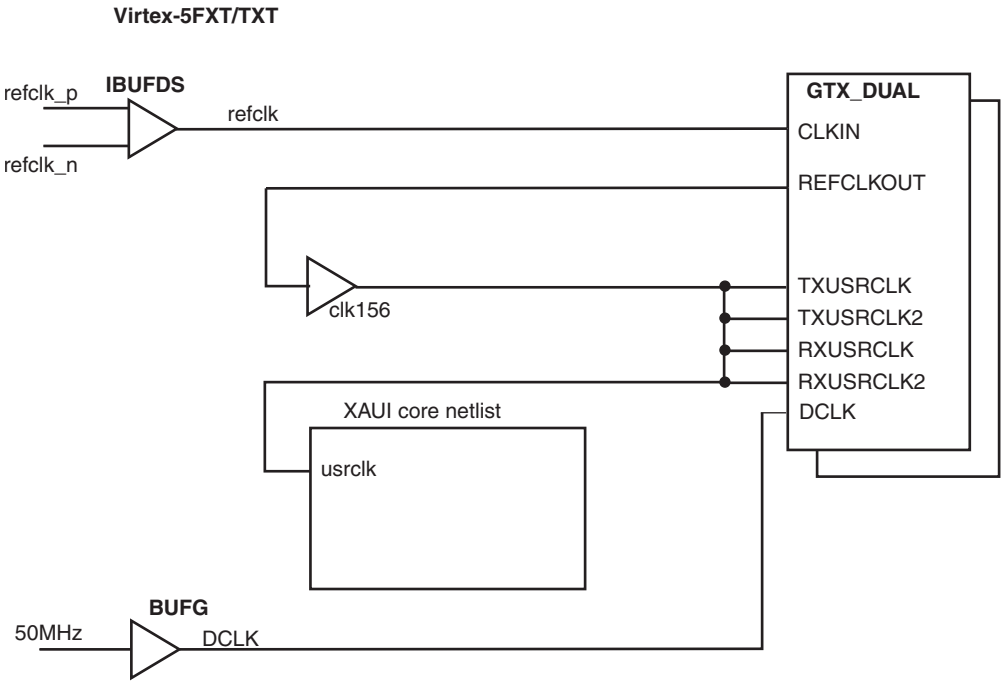

Figure 7-7: **Clock Scheme for Internal Client-Side Interface: Virtex-5 FXT/TXT FPGAs**

External XGMII Interface: No Transmit Elastic Buffer (Virtex-5 LXT/SXT FPGA)

The clock scheme in Figure 7-8 shows the clocking arrangement for the external XGMII client-side interface with no transmit elastic buffer on Virtex-5 LXT/SXT FPGA devices.

The device-specific transceiver clocking requirements are that the `TXUSRCLK`, `TXUSRCLK2`, `RXUSRCLK`, and `RXUSRCLK2` inputs are derived from the same clock that drives the `MGTCLK` inputs; as a result, the `xgmii_tx_clk` input *must* be derived from the same source as the `refclkp/refclk_n` differential pair in the system.

The `clk156` and `clk156_90` signals are used to clock DDR input and output registers in the correct phasing for XGMII signaling.

A dedicated clock is used by the GTP transceiver tiles. The example design uses a 50 MHz clock. You might choose to use a different frequency to allow sharing of clock resources. See the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* for more information about this clock.

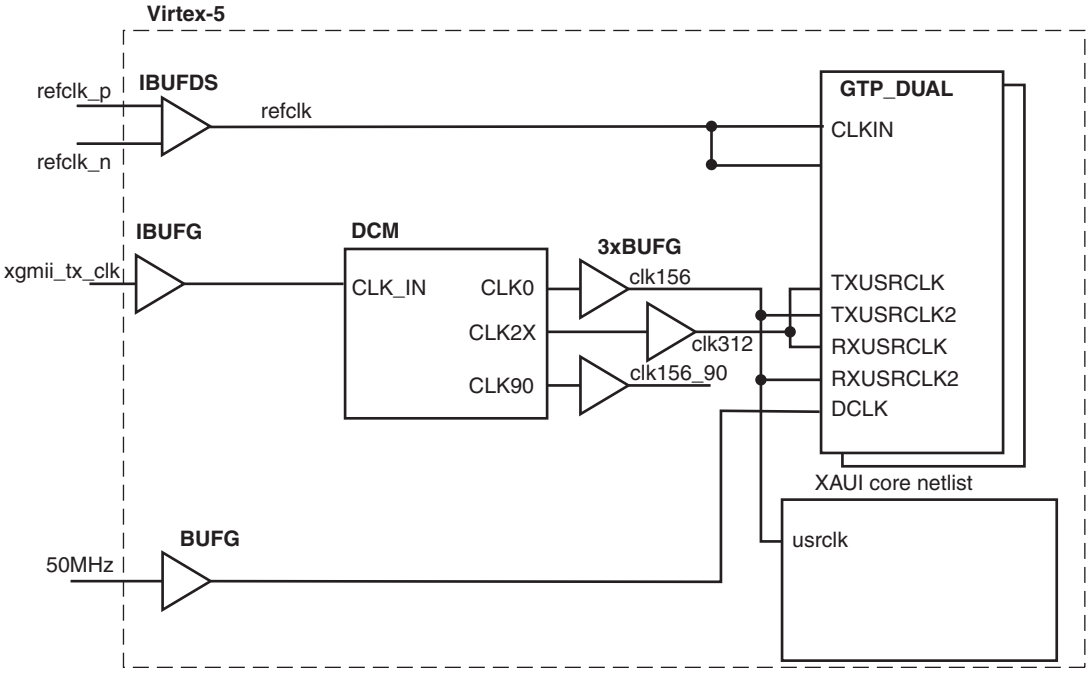

Figure 7-8: **Clock Scheme for External XGMII Client-Side Interface without Transmit Elastic Buffer: Virtex-5 LXT/SXT FPGAs**

External XGMII Interface: No Transmit Elastic Buffer (Virtex-5 FXT/TXT FPGAs)

The clock scheme in Figure 7-9 shows the clocking arrangement for the external XGMII client-side interface with no transmit elastic buffer on Virtex-5 FXT/TXT FPGA devices.

The device-specific RocketIO transceiver clocking requirements are that the `TXUSRCLK`, `TXUSRCLK2`, `RXUSRCLK`, and `RXUSRCLK2` inputs are derived from the same clock that drives the `MGTCLK` inputs; as a result, the `xgmii_tx_clk` input *must* be derived from the same source as the `refclkp/refclk_n` differential pair in the system.

The `clk156` and `clk156_90` signals are used to clock DDR input and output registers in the correct phasing for XGMII signaling.

A dedicated clock is used by the GTX transceiver tiles. The example design uses a 50 MHz clock. You might choose to use a different frequency to allow sharing of clock resources. See the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* for more information about this clock.

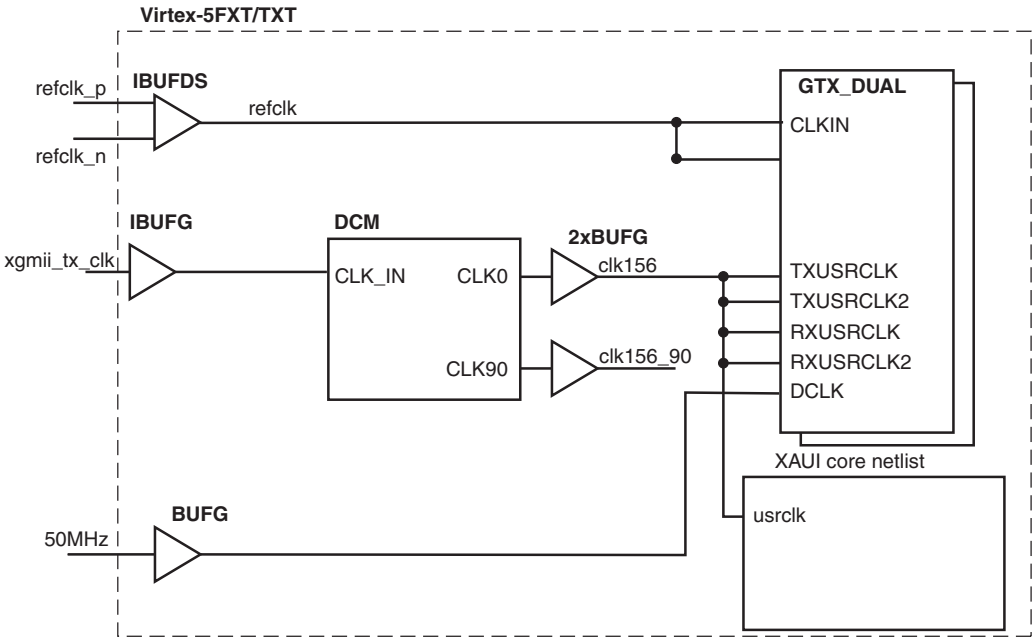

Figure 7-9: **Clock Scheme for External XGMII Client-Side Interface without Transmit Elastic Buffer: Virtex-5 FXT/TXT FPGAs**

External XGMII Interface: Transmit Elastic Buffer (Virtex-5 LXT/SXT FPGA)

Often, it cannot be arranged that the inbound transmit data clock domain is derived from the reference clock. In these circumstances, the optional transmit elastic buffer in the core must be used. Figure 7-10 shows a clocking scheme for this case.

The `tx_clk` pin of the XAUI core is used to clock the input side of an elastic buffer in the transmit path; `usrclk` is used to clock the output side.

The `clk156` and `clk156_90` signals are used to clock DDR output registers in the correct phasing for XGMII signaling. The `tx_clk0` signal is used to clock DDR input registers.

A dedicated clock is used by the GTP transceiver tiles. The example design uses a 50 MHz clock. You might choose to use a different frequency to allow sharing of clock resources. See the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* for more information about this clock.

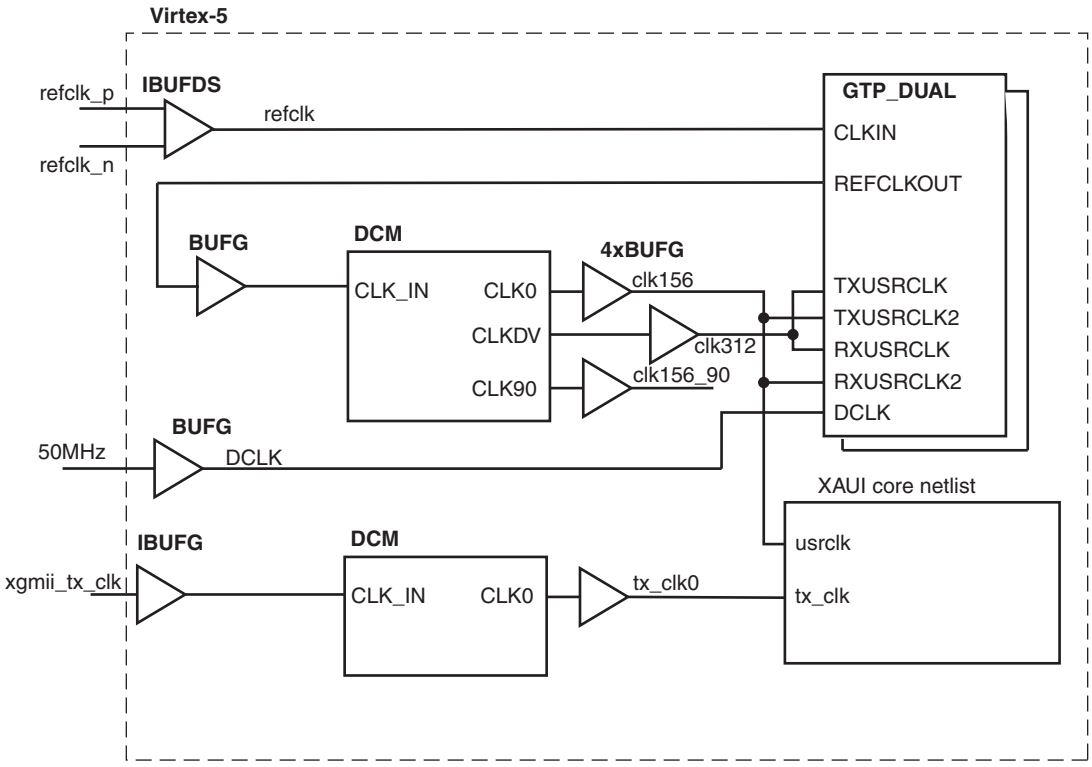

Figure 7-10: **Clock Scheme for External XGMII Client-Side Interface with Transmit Elastic Buffer: Virtex-5 LXT/SXT FPGAs**

External XGMII Interface: Transmit Elastic Buffer (Virtex-5 FXT/TXT FPGA)

Often, it cannot be arranged that the inbound transmit data clock domain is derived from the reference clock. In these circumstances, the optional transmit elastic buffer in the core must be used. Figure 7-11 shows a clocking scheme for this case.

The `tx_clk` pin of the XAUI core is used to clock the input side of an elastic buffer in the transmit path; `usrclk` is used to clock the output side.

The `clk156` and `clk156_90` signals are used to clock DDR output registers in the correct phasing for XGMII signaling. The `tx_clk0` signal is used to clock DDR input registers.

A dedicated clock is used by the GTX transceiver tiles. The example design uses a 50 MHz clock. You might choose to use a different frequency to allow sharing of clock resources. See the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* for more information about this clock.

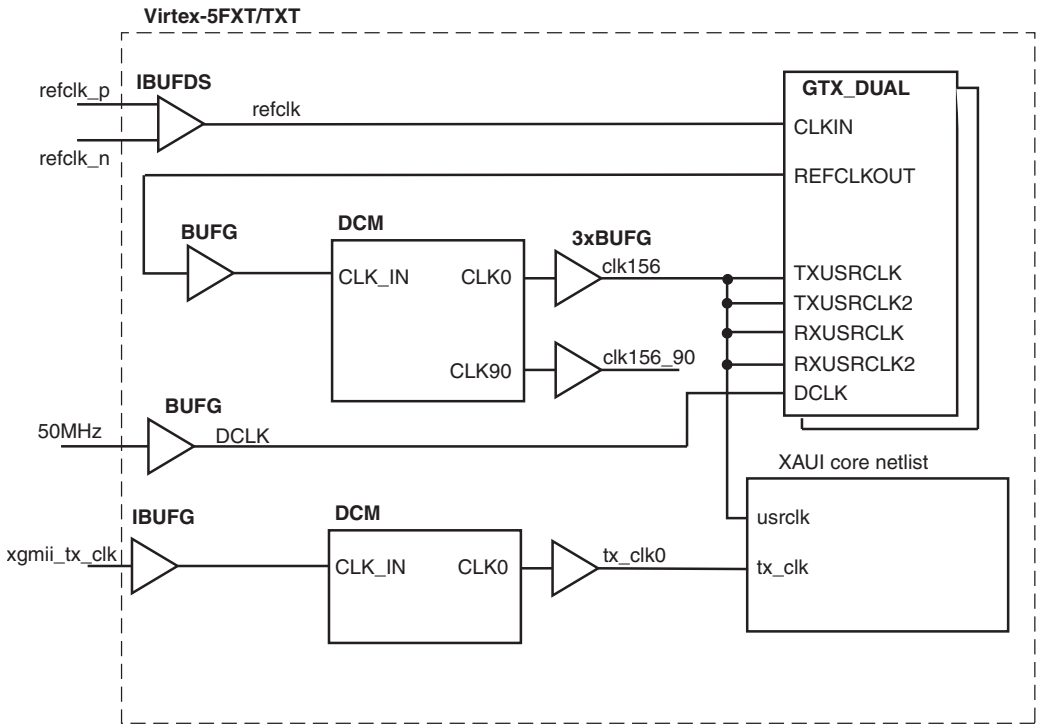

Figure 7-11: **Clock Scheme for External XGMII Client-Side Interface with Transmit Elastic Buffer: Virtex-5 FXT/TXT FPGAs**

Clocking: Virtex-4 FPGAs

The clocking schemes in this section are illustrative only and may require customization for a specific application.

Reference Clock

The GT11 transceivers require a reference clock of 312.5 MHz to operate at a line rate of 3.125 Gb/s.

Transceiver Placement

Common to all schemes shown is that a single GT11CLK_MGT block is used to feed the MGT clock tree for all four transceivers; as a result, all four transceivers in an instance of the core *must* be in a single column of MGT transceiver tiles. In addition, timing requirements can be met if the four transceivers are placed on neighboring tiles within the column.

See Chapter 2, "Clocking and Timing Considerations" in the *Virtex-4 RocketIO MGT User Guide* for more information on Virtex-4 FPGA RocketIO™ MGT transceiver clock distribution.

Internal Client-Side Interface

The simplest clocking scheme is that for the internal client interface, as shown in Figure 7-12. The GT11 transceiver primitives require a 78.125 MHz clock and this is generated by a DCM. The 156.25 MHz clock from the DCM is used as the clock for the netlist part of the XAUI core and is typically also used for your logic.

A dedicated clock is used for the calibration blocks. The example design uses a 50 MHz clock. You might choose to use a different frequency to allow sharing of clock resources.

See the *Calibration Block User Guide* before changing this clock. Also, see Answer Record 22477 for updates to the *Calibration Block User Guide*.

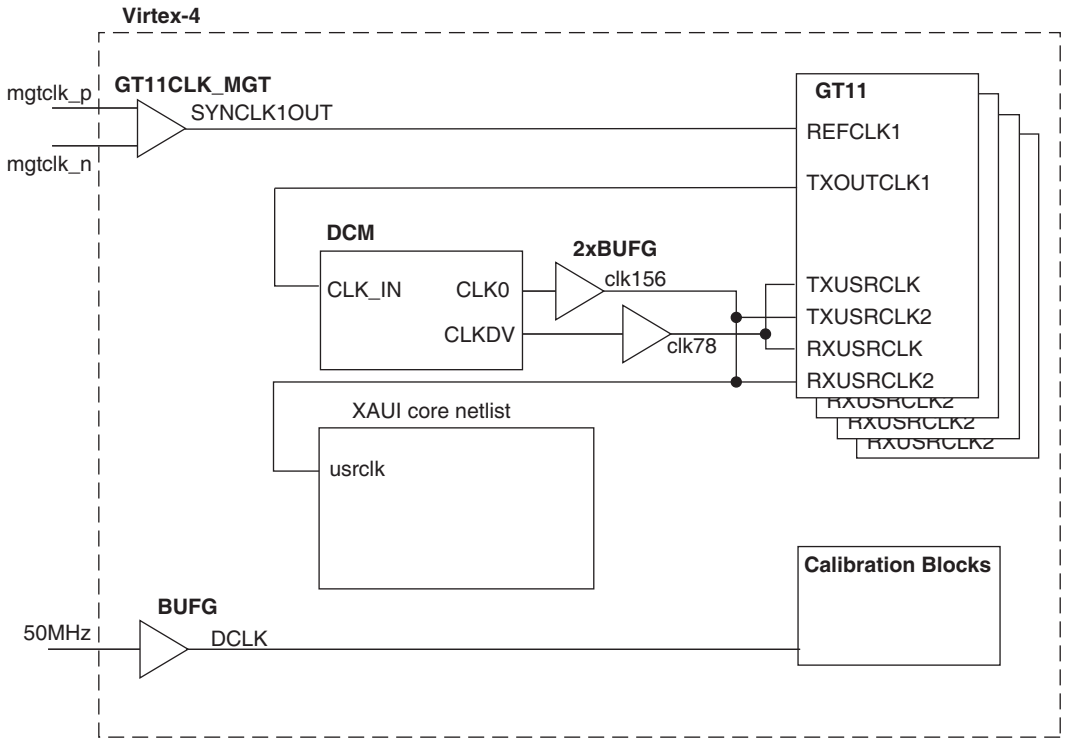

Figure 7-12: **Clock Scheme for Internal Client-Side Interface: Virtex-4 FPGAs**

External XGMII Interface: No Transmit Elastic Buffer

The clock scheme in Figure 7-13 shows the clocking arrangement for the external XGMII client-side interface with no transmit elastic buffer on Virtex-4 FPGA devices.

The device-specific transceiver clocking requirements are that the `TXUSRCLK`, `TXUSRCLK2`, `RXUSRCLK`, and `RXUSRCLK2` inputs are derived from the same clock that drives the `MGTCLK` inputs; as a result, the `xgmii_tx_clk` input *must* be derived from the same source as the `mgtclkp/mgtclk_n` differential pair in the system.

The `clk156` and `clk156_90` signals are used to clock DDR input and output registers in the correct phasing for XGMII signaling.

A dedicated clock is used for the calibration blocks. The example design uses a 50 MHz clock. You can choose to use a different frequency to allow sharing of clock resources.

See the *Calibration Block User Guide* before changing this clock. Also see Answer Record 22477 for updates to the *Calibration Block User Guide*.

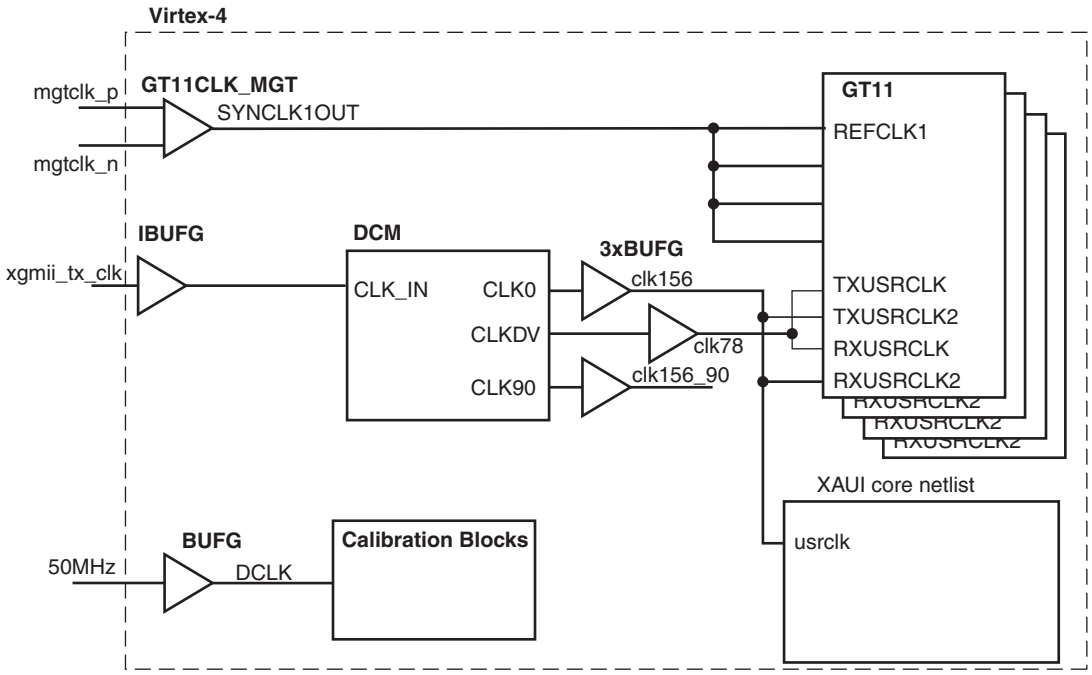

Figure 7-13: **Clock Scheme for External XGMII Client-Side Interface without Transmit Elastic Buffer: Virtex-4 FPGAs**

External XGMII Interface: Transmit Elastic Buffer

Often, it cannot be arranged that the inbound transmit data clock domain is derived from the `MGTCLK` reference clock. In these circumstances, the optional transmit elastic buffer in the core must be used. Figure 7-14 shows a clocking scheme for this case.

The `tx_clk` pin of the XAUI core is used to clock the input side of an elastic buffer in the transmit path; `usrclk` is used to clock the output side.

The `clk156` and `clk156_90` signals are used to clock DDR output registers in the correct phasing for XGMII signaling. The `tx_clk0` signal is used to clock DDR input registers.

A dedicated clock is used for the calibration blocks. The example design uses a 50 MHz clock. You can choose to use a different frequency to allow sharing of clock resources.

See the *Calibration Block User Guide* before changing this clock. Also, see Answer Record 22477 for updates to the *Calibration Block User Guide*.

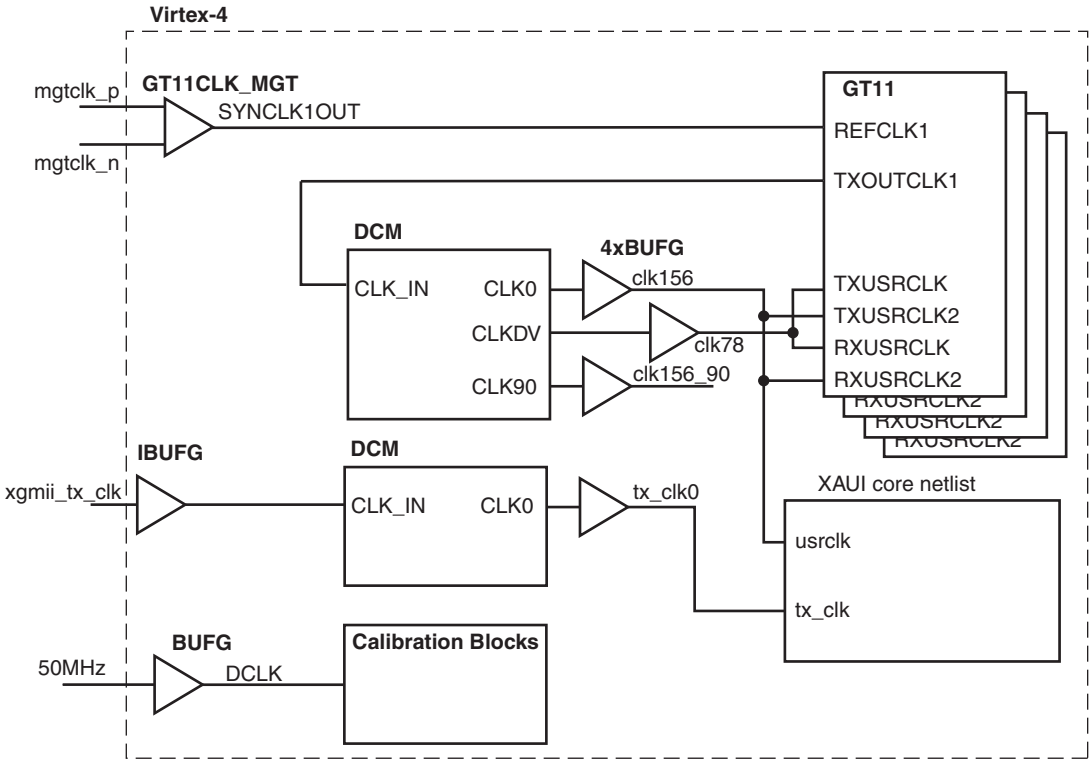

Figure 7-14: **Clock Scheme for External XGMII Client-Side Interface with Transmit Elastic Buffer: Virtex-4 FPGAs**

Using Both Transceiver Columns in Virtex-4 FX FPGAs

Where possible, it is recommended that device-specific RocketIO transceivers be placed in adjacent tiles in the same column. However sometimes this is not possible, and it is necessary to divide the transceivers between the two columns. This implementation of the XAUI core requires particular care.

The primary issue is that each column of transceivers must be fed a low-jitter reference clock from a separate GT11CLK_MGT block (two reference clocks must be supplied to the device, derived from the same clock source).

As shown in Figure 7-15, the `usrclk` derived from one `TXOUTCLK` has been used to provide the system clock for the entire XAUI core as well as the FPGA logic ports of all four device-specific RocketIO transceivers. The Virtex-4 FPGA example design instantiates the necessary circuitry to ensure that the transceivers are all phase-aligned.

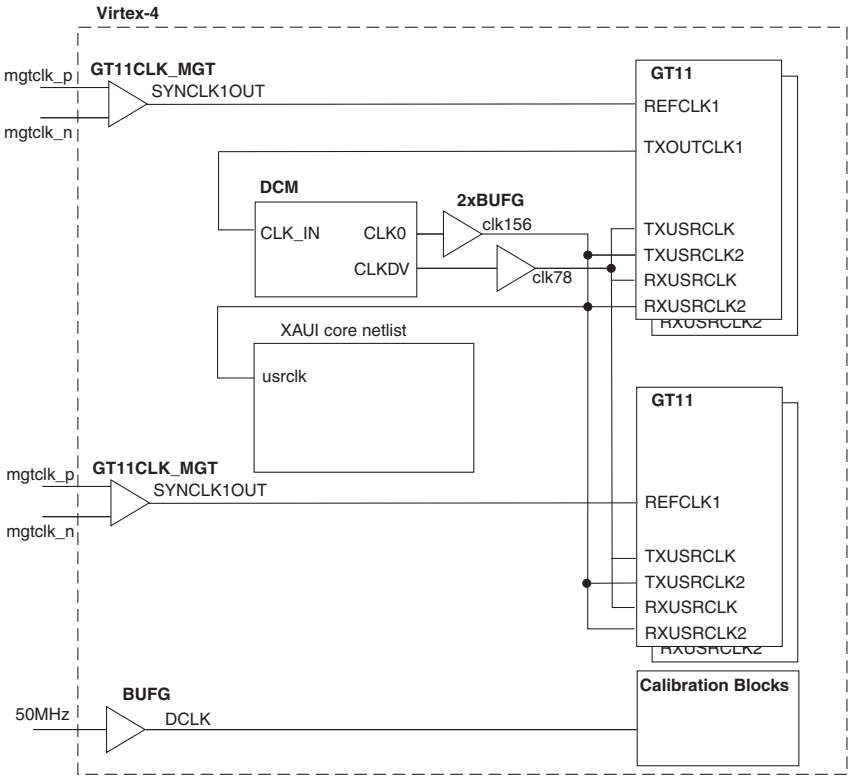

Figure 7-15: **Clocking Using Two Columns**

Multiple Core Instances

If more than one instance of the XAUI core is implemented in a Virtex-4, Virtex-5 or Virtex-6 FPGA, transceivers and cores sharing a column may also share reference and logic clocks. If instances are not implemented in the same column, each core must be treated as an independent clock domain.

In Virtex-7 and Kintex-7 devices, the reference clock can be shared from a neighboring quad. Logic clocks cannot be shared between core instances with the supplied design. The `USRCLKs` on each core and quad of transceivers must be sourced from the `TXOUTCLK` port of that quad.

See the "Clocking and Timing Considerations" chapter in the *Virtex-4 RocketIO MGT User Guide* for more information about Virtex-4 FPGA RocketIO MGT transceiver clock distribution.

See the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide*, *Virtex-5 FPGA RocketIO GTX Transceiver User Guide, Virtex-6 FPGA GTX Transceiver User Guide* and *7 Series FPGAs Transceivers User Guide*.

Reset Circuits

All register resets within the XAUI core netlist are synchronous to the `usrclk` port, apart from the registers on the input side of the transmit elastic buffer which are synchronous to the `tx_clk` port.

Receiver Termination: Virtex-7, Kintex-7, Virtex-6, Virtex-5 and Spartan-6 FPGAs

The receiver termination must be set correctly. The default setting is 2/3 VTTRX.

- See the Receiver chapter in the *7 Series FPGAs Transceivers User Guide (UG476)*.
- See "Chapter 4, Receiver" *Virtex-6 FPGA GTX Transceiver User Guide* (UG366)
- See "Chapter 4, Receiver" in the *Spartan-6 FPGA GTP Transceiver User Guide* (UG386).
- See "Chapter 7, GTP Receiver," in the *Virtex-5 FPGA RocketIO GTP Transceiver User* Guide (UG196) or "Chapter 7, GTX Receiver" in the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* (UG198) for information about receiver termination.

Transmit Skew

The transceivers are configured to operate in a mode that minimizes the amount of transmit skew that can be introduced between lanes. Full details on that maximum amount of transmit skew can be found by looking at T_{LLSKEW} in the appropriate device data sheet.

Under some circumstances it is possible that T_{LLSKEW} can exceed the PMA Tx Skew budget defined in 802.3-2008. If it is necessary to keep within this skew budget, then the appropriate amount must be borrowed from the PCB and medium sections of the budget to keep the total amount of skew within range.

XILINX®

Chapter 8

Implementing the Core

This chapter describes how to simulate and implement your design containing the XAUI core.

Pre-implementation Simulation

A unit delay gate-level model of the XAUI core netlist is provided as a Xilinx® CORE Generator™ tool output file. This can be used for simulation of the block in the design phase of a project.

For information about setting up your simulator to use the pre-implemented model, consult the Xilinx *Synthesis and Verification Design Guide*, included in your Xilinx software installation.

The unit delay gate-level model of the XAUI core can be found in the CORE Generator tool project directory. Details of the CORE Generator tool outputs can be found in Chapter 10, Detailed Example Design.

VHDL

component_name.vhd

Verilog

component_name.v

Synthesis

XST: VHDL

In the CORE Generator tool project directory, there is a xaui_component_name.vho file that is a component and instantiation template for the core. Use this to help instance the XAUI core into your VHDL source.

When your entire design is complete, create:

- An XST project file top_level_module_name.prj listing all your source code files
- An XST script file top_level_module_name.scr containing your required synthesis options

To synthesize the design, run:

```
$ xst -ifn top_level_module_name.scr
```

See the *XST User Guide* for more information on creating project and synthesis script files and running the `xst` program.

XST: Verilog

In the CORE Generator tool project directory, there is a module declaration for the XAUI core at:

```
<project_directory>/<component_name>/implement/component_name_mod.v
```

Use this module to help instance the XAUI core into your Verilog source.

When your entire design is complete, create:

- An XST project file top_level_module_name.prj listing all your source code files. Make sure you include:

  ```
  %XILINX%/verilog/src/iSE/unisim_comp.v
  ```

  and

  ```
  <project_directory>/<component_name>/implement/component_name_mod.
  v
  ```

  as the first two files in the project list.
- An XST script file top_level_module_name.scr containing your required synthesis options.

To synthesize the design, run:

```
$ xst -ifn top_level_module_name.scr
```

See the *XST User Guide* for more information about creating project and synthesis script files, and running the `xst` program.

Implementation

Generating the Xilinx Netlist

To generate the Xilinx netlist, the ngdbuild tool is used to translate and merge the individual design netlists into a single design database, the Native Generic Database (NGD) file. Also merged at this stage is the User Constraints File (UCF) for the design. An example of the `ngdbuild` command is:

```
$ ngdbuild -sd path_to_xaui_netlist -sd
path_to_user_synth_results \
    -uc top_level_module_name.ucf top_level_module_name
```

Mapping the Design

To map the logic gates of your design netlist into the Configurable Logic Blocks (CLBs) and IOBs of the FPGA, run the `map` command. The `map` command writes out a physical design to an NCD file. An example of the `map` command is:

```
$ map -o top_level_module_name_map.ncd top_level_module_name.ngd \
top_level_module_name.pcf
```

Placing and Routing the Design

To place and route your design logic components (mapped physical logic cells) contained within an Native Circuit Description (NCD) file in accordance with the layout and timing requirements specified in the PCF file, the `par` command must be executed. The `par` command outputs the placed and routed physical design to an NCD file. An example of the `par` command is:

```
$ par -ol high top_level_module_name_map.ncd top_level_module_name.ncd
\ top_level_module_name.pcf
```

Static Timing Analysis

To evaluate timing closure on a design and create a Timing Report file (TWR) derived from static timing analysis of the Physical Design file (NCD), the `trce` command must be executed. The analysis is typically based on constraints included in the optional PCF file. An example of the `trce` command is:

```
$ trce -o top_level_module_name.twr top_level_module_name.ncd \
    top_level_module_name.pcf
```

Generating a Bitstream

To create the configuration bitstream (BIT) file based on the contents of a physical implementation file (NCD), the `bitgen` command must be executed. The BIT file defines the behavior of the programmed FPGA. An example of the `bitgen` command is:

```
$ bitgen top_level_module_name.ncd top_level_module_name.bit
top_level_module_name.pcf
```

Other Implementation Information

For more information about using the Xilinx implementation tool flow including command line switches and options, consult the software manuals that came with your Xilinx ISE® design suite.

Chapter 9

Quick Start Example Design

This chapter provides instructions for generating a core using the default configuration, implementing the example design, and simulating your design using Mentor Graphics ModelSim v6.6d, Cadence Incisive Enterprise Simulator (IES) v10.2, and Synopsys Verilog Compiled Simulator (VCS) and VCS MX 2010.06.

Introduction

Figure 9-1 illustrates the default configuration of the example design.

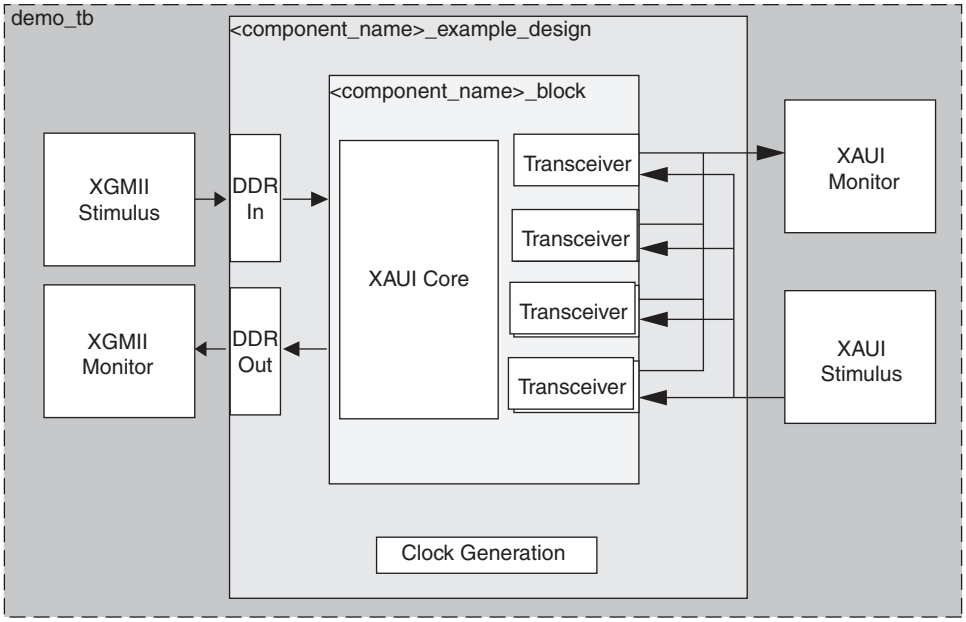

Figure 9-1: **XAUI Example Design and Test Bench: Default Configuration**

The XAUI example design consists of the following:

- A XAUI core netlist
- Transceiver wrappers
- An example HDL wrapper
- A demonstration test bench to exercise the example design

The XAUI Design Example has been tested with Xilinx® ISE® tools v13.4, Mentor Graphics ModelSim v6.6d, Cadence Incisive Enterprise Simulator (IES) v10.2 and Synopsys VCS and VCS MX 2010.06.

Generating the Core

To generate a XAUI core with default values using the CORE Generator™ tool do the following:

1. Start the CORE Generator tool.

   For help starting and using the CORE Generator tool, see the documentation supplied with ISE design suite.

2. Choose **File** --> **New Project**.
3. Type a directory name.
4. Do the following to set project options:
   - From the Part tab, select a silicon family, part, speed grade, and package that supports the XAUI core, for example, Virtex®-4 FPGAs.
   - If an unsupported silicon family is selected, the XAUI core does not appear in the taxonomy tree. For a list of supported architectures, see the *XAUI Data Sheet*.
   - From the Generation tab, select VHDL or Verilog; for Vendor, select Other.
   - On the Advanced tab, accept the default values.
5. After creating the project, locate the core in the taxonomy tree at the left side of the CORE Generator tool window. The XAUI core appears under the following categories:
   - Communications & Networking/Ethernet
   - Communications & Networking/Networking
   - Communications & Networking/Telecommunications
6. Double-click the core to open it. A message might appear warning you about the limitations of the Simulation Only license, and then the XAUI customization screen appears.
7. In the Component Name field, enter a name for the core instance.
8. Accept the remaining default options and click Finish to generate the core.

The core and its supporting files, including the example design, are generated in the project directory. For a detailed description of the directory structure and files, see Directory and File Contents in Chapter 10

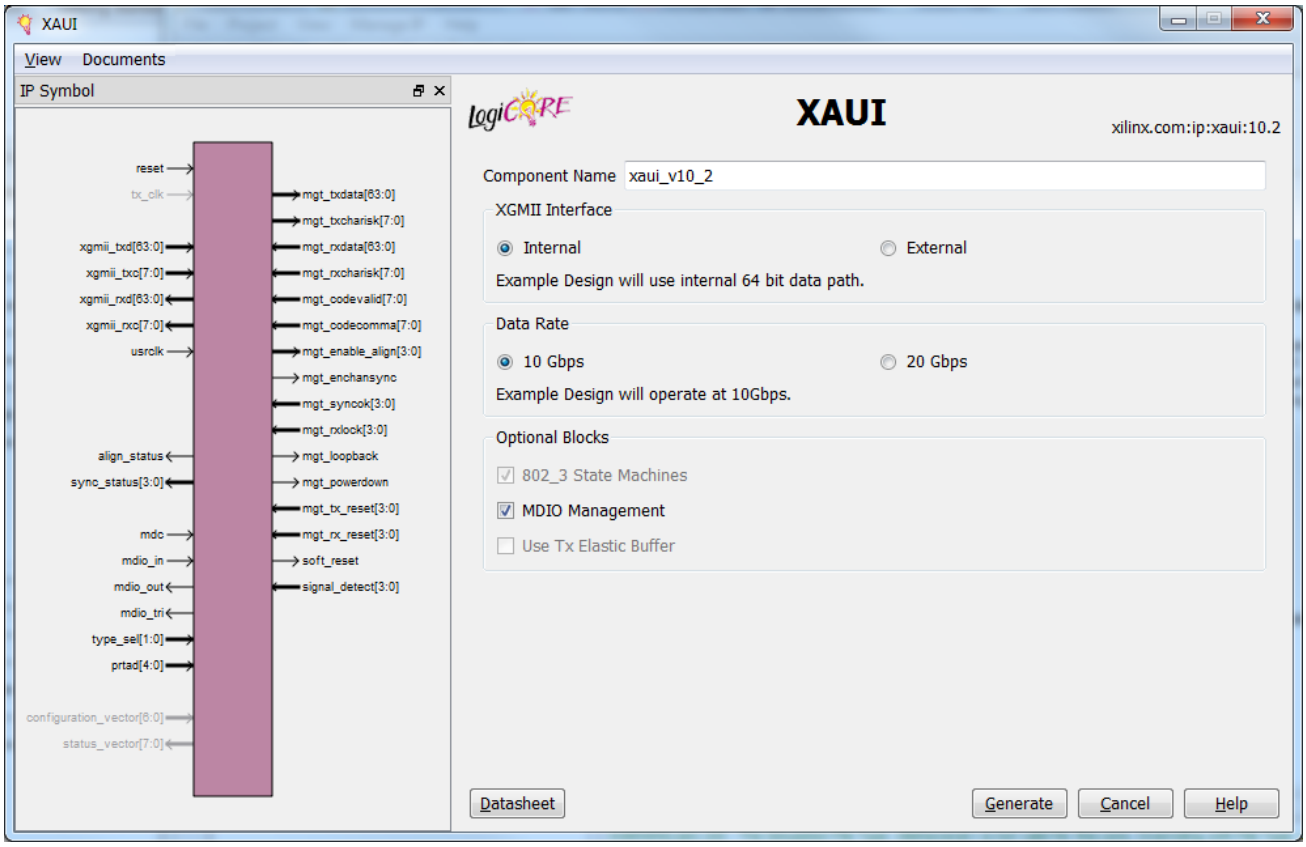

Figure 9-2: **XAUI Main Screen**

Implementing the XAUI Example Design

If the core is generated with a Simulation Only license, the implementation feature of the example design is not available; in this case, go directly to Chapter 9, Simulating the XAUI Example Design.

After the core is successfully generated, the netlist and example design HDL wrapper can be processed through the Xilinx implementation tools. The generated outputs include several scripts to assist in processing.

Open a command prompt or shell in your project directory and enter the following commands:

Linux

```
% cd <component_name>/implement
% ./implement.sh
```

Windows

```
> cd <component_name>\implement
> implement.bat
```

The implement command accomplishes the following:

- Starts a script to synthesize the example design HDL wrapper
- Builds, maps, and place-and-routes the example design (Full license only)
- Creates gate-level netlist HDL files in both VHDL and Verilog with associated timing information (SDF files)

The created files are placed in the results directory that is created by the implement script at run time.

Simulating the XAUI Example Design

The example design provided with the XAUI core provides a complete environment which allows you to simulate the core and view the outputs. Scripts are provided for pre- and post-layout simulation. The simulation model is either in VHDL or Verilog depending on the CORE Generator tool Design Entry project option.

Setting up for Simulation

To run the gate-level simulation you must have the Xilinx® Simulation Libraries compiled for your system. See the Compiling Xilinx Simulation Libraries (COMPXLIB) in the *Xilinx ISE Synthesis and Verification Design Guide*, and the *Xilinx ISE Software Manuals and Help*. You can download these documents from: www.xilinx.com/support/software_manuals.htm.

The Xilinx simulation libraries must be mapped into the simulator. If the libraries are not set for your environment, go to Answer Record 15338 on www.xilinx.com/support for assistance compiling Xilinx simulation models and setting up the simulator environment.

All Virtex FPGA designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. For a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator, the following simulators are supported.

- Mentor Graphics ModelSim 6.6d
- Cadence Incisive Enterprise Simulator (IES) 10.2
- Synopsys VCS and VCS MX 2010.06

Pre-Implementation Simulation

To run a functional simulation of the example design:

1. Open a command prompt or shell in your project directory and set the current directory to

   ```
   <component_name>/simulation/functional
   ```

2. Launch the simulation script:

   ```
   ModelSim: vsim -do simulate_mti.do
   Cadence sim: ./simulate_ncsim.sh
   vcs: ./simulate_vcs.sh
   ```

The simulation script compiles the functional model and the demonstration test bench, adds some relevant signals to a wave window, and then runs the simulation to completion. You can then inspect the simulation transcript and waveform to observe the operation of the core.

Post-Implementation Simulation

To run a timing simulation of the example design:

1. Open a command prompt or shell in your project directory, then set the current directory to:

   ```
   <component_name>/simulation/timing
   ```

2. Launch the simulation script:

   ```
   modelSim: vsim -do simulate_mti.do
   ncsim: ./simulate_ncsim.sh
   vcs: ./simulate_vcs.sh
   ```

The simulation script compiles the gate-level model and the demonstration test bench, adds some relevant signals to a wave window, and then runs the simulation to completion. You can then inspect the simulation transcript and waveform to observe the operation of the core.

Additional Information

For more information about the example design, including guidelines for modifying the design and extending the test bench, see Chapter 10, Detailed Example Design. To start using the XAUI core in your own design, see the *XAUI User Guide*.

Chapter 10

Detailed Example Design

This chapter provides detailed information about the example design, including a description of the files and the directory structure generated by the Xilinx® CORE Generator™ tool, the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

- <project directory>
  Top-level project directory; name is user-defined.
  - <project directory>/<component name>
    Core release notes file
    - <component name>/doc
      Product documentation
    - <component name>/example_design
      Verilog and VHDL design files
    - <component name>/implement
      Implementation script files
      - implement/results
        Results directory, created after implementation scripts are run, and contains implement script results
    - <component name>/simulation
      Simulation scripts
      - simulation/functional
        Functional simulation files
      - Implementation and Test Scripts
        Timing simulation files

Directory and File Contents

The core directories and their associated files are defined in the following sections.

<project directory>

The project directory contains all the CORE Generator tool project files.

Table 10-1: **Project Directory**

Name	Description
<project_dir>	
<component_name>.ngc	A binary Xilinx implementation netlist. Describes how the core is to be implemented. Used as an input to the Xilinx implementation tools.
<component_name>.v[hd]	VHDL or Verilog structural simulation model. File used to support functional simulation of a core.
<component_name>.xco	As an output file, the XCO file is a log file which records the settings used to generate a particular core. An XCO file is generated by the CORE Generator tool for each core that it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator tool.
<component_name>_flist.txt	List of files delivered with the core
<component_name>.{veo|vho}	A VHDL or Verilog template for the core. This can be copied into your design.

Back to Top

<project directory>/<component name>

The <component name> directory contains the release notes file provided with the core, which might include last-minute changes and updates.

Table 10-2: **Component Name Directory**

Name	Description
<project_dir>/<component_name>	
xaui_readme.txt	Core release notes file

Back to Top

<component name>/doc

The doc directory contains the PDF documentation provided with the core.

Table 10-3: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
xaui_ds266.pdf	XAUI Data Sheet
xaui_ug150.pdf	XAUI User Guide

Back to Top

<component name>/example_design

The example design directory contains the example design files provided with the core.

Table 10-4: **Example Design Directory**

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_block.v[hd]	Block entity containing the XAUI core and transceiver wrappers
<component_name>_example_design.v[hd]	Top-level entity for the example design containing the block level design and clocking circuitry
<component_name>_example_design.ucf	User constraints file for the core and example design
<component_name>_mod.v	Wrapper file for the XAUI core
cal_block_v1_4_1.v[hd] (Virtex®-4 FPGAs only)	Virtex-4 FX FPGA Calibration Block
chanbond_monitor.v[hd]	Transceiver Channel Bonding Monitor
cc_2b_1skp.v[hd] (Virtex-5 FXT/TXT FPGAs only)	Transceiver Clock Correction
rocketio_init_rx.v[hd] (Virtex-4 FPGAs only) rocketio_init_tx.v[hd] (Virtex-4 FPGAs only) <component_name>_tx_sync.v[hd] (Spartan®-6, Virtex-5 and Virtex-6 FPGAs)	Transceiver initialization circuitry

Table 10-4: **Example Design Directory *(Cont'd)***

Name	Description
rocketio_wrapper.v[hd] (Virtex-4, Virtex-5 FPGAs only) rocketio_wrapper_tile.v[hd] (Virtex-5 FPGAs only) gtp_wrapper.v[hd] (Spartan-6 FPGAs only) gtp_wrapper_tile.v[hd] (Spartan-6 FPGAs only) gtx_wrapper_gtx.v[hd] (Virtex-6 FPGAs only) gtx_wrapper.v[hd] (Virtex-6 FPGAs only) gt_wrapper.v[hd] (Virtex-7 and Kintex™-7 FPGAs only) gt_wrapper_gt.v[hd] (Virtex-7 and Kintex-7 FPGAs only)	Wrappers for the transceivers

Back to Top

<component name>/implement

This directory contains the support files necessary for implementation of the example design with the Xilinx tools. Execution of an implement script creates a results directory and an `xst` project directory.

Table 10-5: **Implement Directory**

Name	Description
<project_dir>/<component_name>/implement	
implement.bat	Windows batch file that process the example design through the Xilinx tool flow
implement.sh	Linux shell script that processes the example design through the Xilinx tool flow
xst.scr	XST script file for the example design
xst.prj	XST project file for the example design
xst.xcf	XCF constraint file for the example design

Back to Top

implement/results

This directory is created by the implement scripts and is used to run the example design files and the <component_name>.ngc file through the Xilinx implementation tools. On completion of an implement script, this directory contains the following files for timing simulation. Output files from the Xilinx implementation tools can also be found in this directory.

Table 10-6: **Results Directory**

Name	Description
<project_dir>/<component_name>/implement/results	
routed.v[hd]	The back-annotated SIMPRIM-based VHDL or Verilog design. Used for timing simulation.
routed.sdf	Timing information for simulation

Back to Top

<component name>/simulation

The simulation directory and the subdirectories below it contain the files necessary to test a VHDL or Verilog implementation of the example design.

Table 10-7: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
demo_tb.v[hd]	The VHDL or Verilog demonstration test bench for the XAUI core

Back to Top

simulation/functional

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

Table 10-8: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_mti.do	ModelSim macro file that compiles the example design sources, the structural simulation model and the demonstration test bench then runs the functional simulation to completion.
simulate_ncsim.sh	Linux shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using the Cadence Incisive Enterprise Simulator (IES) simulator.
simulate_vcs.sh (verilog only)	Linux shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using VCS.
ucli_commands.key (verilog only)	VCS command file. This file is called by the simulate_vcs.sh script.
vcs_session.tcl (verilog only)	VCS DVE tcl script that opens wave windows and adds interesting signals to it. This macro is used by the simulate_vcs.sh script.
wave_mti.do	ModelSim macro file that opens a wave window and adds interesting signals to it. This macro is called by the simulate_mti.do macro file.
wave_ncsim.sv	The Cadence IES simulator macro file that opens a wave windows and adds interesting signals to it. This macro is called by the simulate_ncsim.sh script.

Back to Top

Implementation and Test Scripts

Implementation Script

The implementation script is either a shell script or batch file that processes the example design through the Xilinx tool flow. The script is located at:

Linux

```
<project_dir>/<component_name>/implement/implement.sh
```

Windows

```
<project_dir>/<component_name>/implement/implement.bat
```

The implement script performs the following steps:

1. The example HDL wrapper is synthesized using XST.
2. ngdbuild is run to consolidate the core netlist and the wrapper netlist into the NGD file containing the entire design.
3. The design is mapped to the target technology.
4. The design is place-and-routed on the target device.
5. Static timing analysis is performed on the routed design using trce.
6. A bitstream is generated.
7. netgen runs on the routed design to generate VHDL and Verilog netlists and timing information in the form of SDF files.

Setting up for Simulation

The Xilinx UNISIM library must be mapped into the simulator. If the library is not set up for your environment, go to Answer Record 15338 for assistance compiling Xilinx simulation models and for setting up the simulator environment.

All Virtex FPGA designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. For a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator, the following simulators are supported.

- Mentor Graphics ModelSim 6.6d
- Cadence Incisive Enterprise Simulator (IES) 10.2
- Synopsys VCS and VCS MX 2010.06

Simulation Scripts

Simulation macro files are provided for ModelSim and shell scripts are provided for the Cadence IES simulator and Synopsys VCS simulator. The scripts automate the simulation of the test bench and can be found in the following location:

Functional

<project_dir>/*<component_name>*/simulation/functional/simulate_mti.do

<project_dir>/*<component_name>*/simulation/functional/simulate_ncsim.sh

<project_dir>/*<component_name>*/simulation/functional/simulate_vcs.sh

The scripts perform the following tasks:

- Compiles the gate level netlist
- Compiles the demonstration test bench
- Starts a simulation of the test bench
- Opens a Wave window and adds some interesting signals (wave_mti.do/wave_ncsim.sv/vcs_session.tcl)
- Runs the simulation to completion

XAUI Core with External XGMII Client-Side Interface

Example HDL Wrapper

In Figure 10-1, the example generated HDL wrapper contains the following:

- The RocketIO™ transceiver instances
- Virtex-4 FPGA RocketIO transceiver Calibration Blocks (see Answer Record 22477 for information about the *Calibration Block User Guide*)
- Initialization blocks for the transmit and receive RocketIO transceivers
- Clock management logic, including DCM and Global Clock Buffer instances
- DDR logic for the XGMII interface

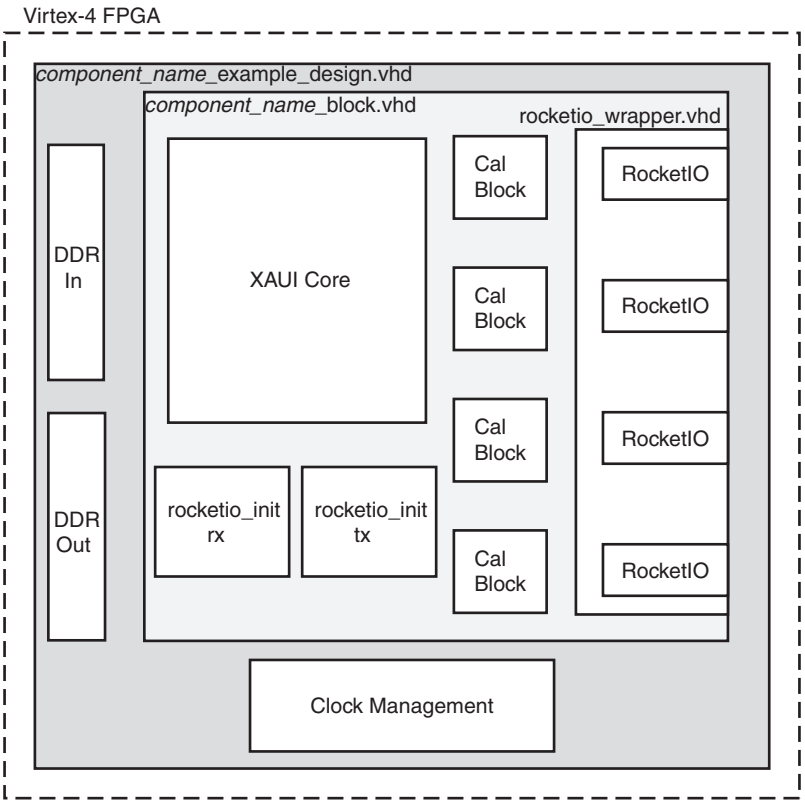

Figure 10-1: **Example HDL Wrapper for XAUI with XGMII (Virtex-4 FPGAs)**

In Figure 10-2, the example generated HDL wrapper contains the following:

- The RocketIO transceiver tile instances
- The wrapper for the two RocketIO transceiver tiles
- A RocketIO transceiver transmit initialization block
- Clock management logic, including DCM and Global Clock Buffer instances
- DDR logic for the XGMII interface

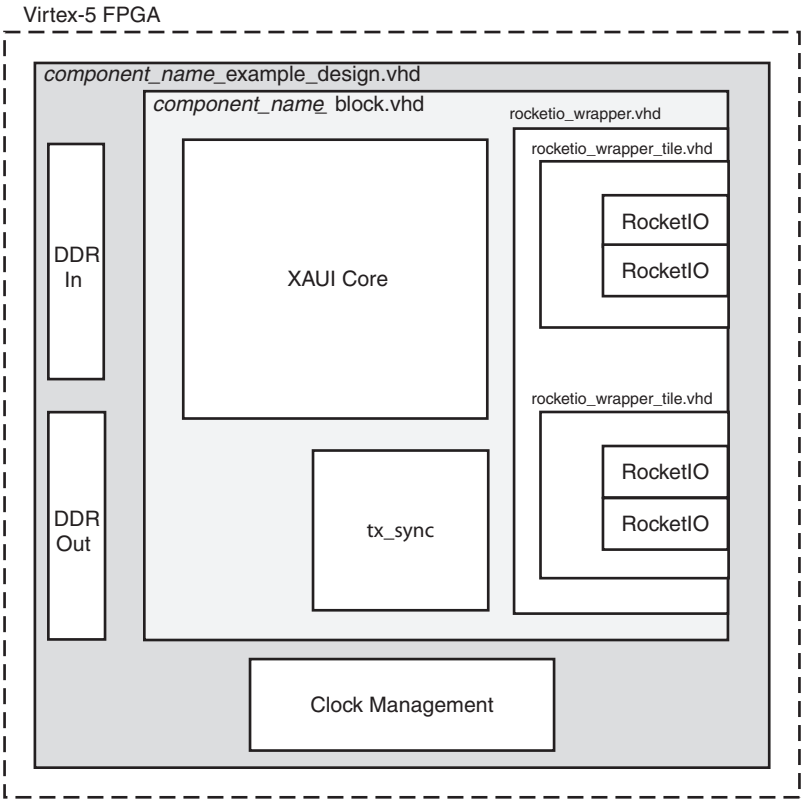

Figure 10-2: **Example HDL Wrapper for XAUI with XGMII (Virtex-5 FPGAs)**

In Figure 10-3, the example generated HDL wrapper contains the following:

- The Transceiver Instances
- A wrapper for the four transceivers
- A transceiver transmit initialization block
- Clock management logic, including MMCM and Global Clock Buffer instances
- DDR logic for the XGMII interface

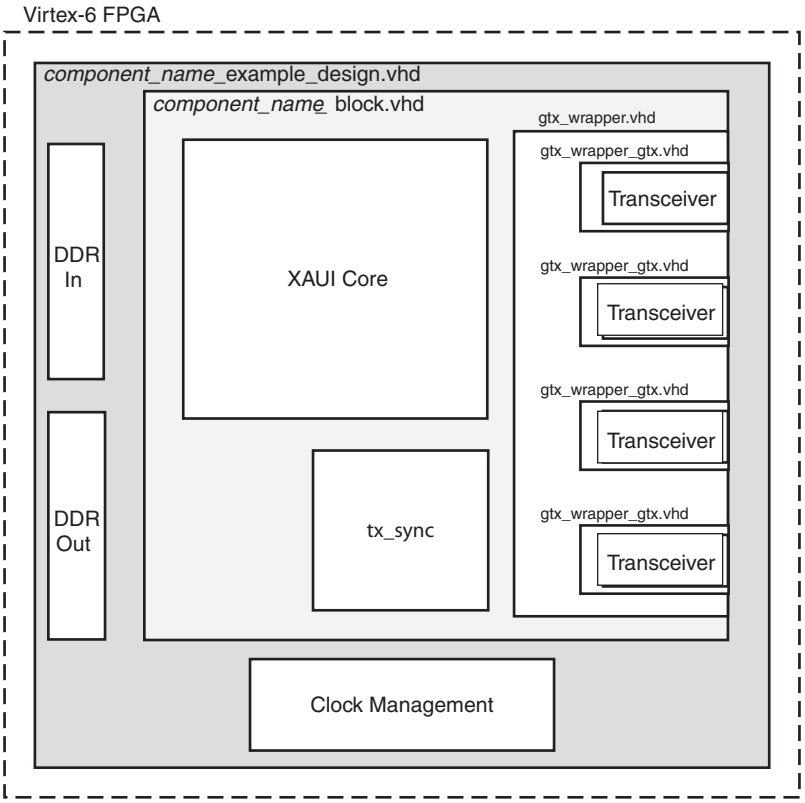

Figure 10-3: **Example HDL Wrapper for XAUI with XGMII (Virtex-6 FPGAs)**

In Figure 10-4, the example generated HDL wrapper contains the following:

- The transceiver and transceiver QPLL Instances
- A wrapper for the four transceivers and transceiver QPLL
- Clock management logic, including MMCM and Global Clock Buffer instances
- DDR logic for the XGMII interface

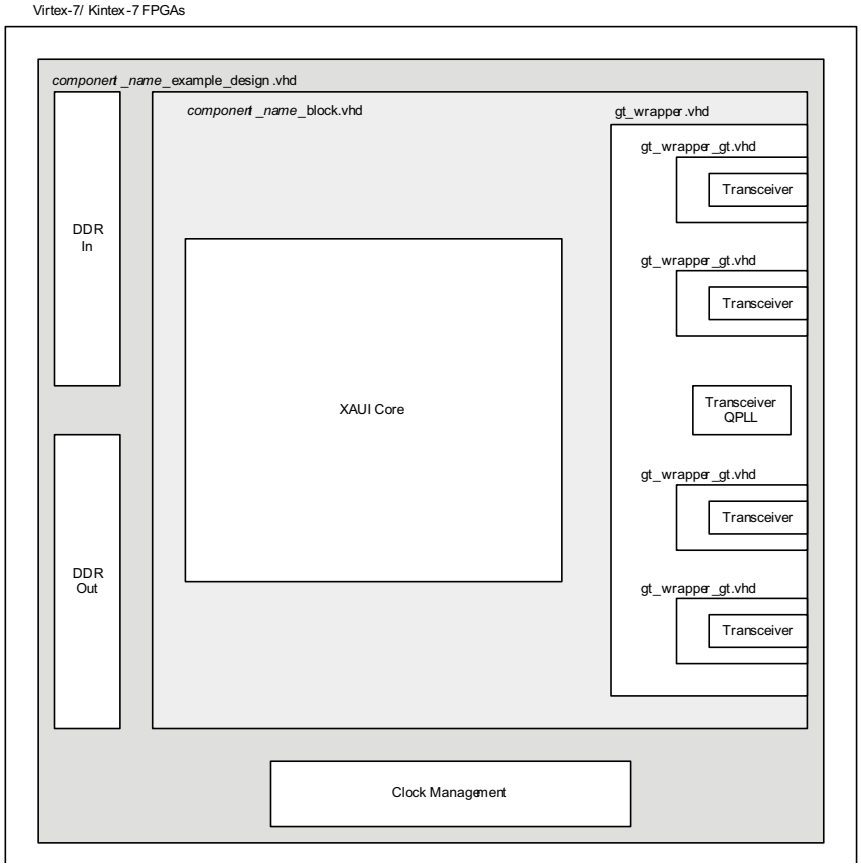

Figure 10-4: **Example HDL Wrapper for XAUI with XGMII (Virtex-7 and Kintex-7 FPGAs)**

Demonstration Test Bench

The demonstration test bench in Figure 10-5 is a simple VHDL or Verilog program to exercise the example design and the core itself. This test bench consists of transactor procedures or tasks which connect to the major ports of the example design, and a control program that pushes frames of varying length and content through the design and checks the values as they exit the core.

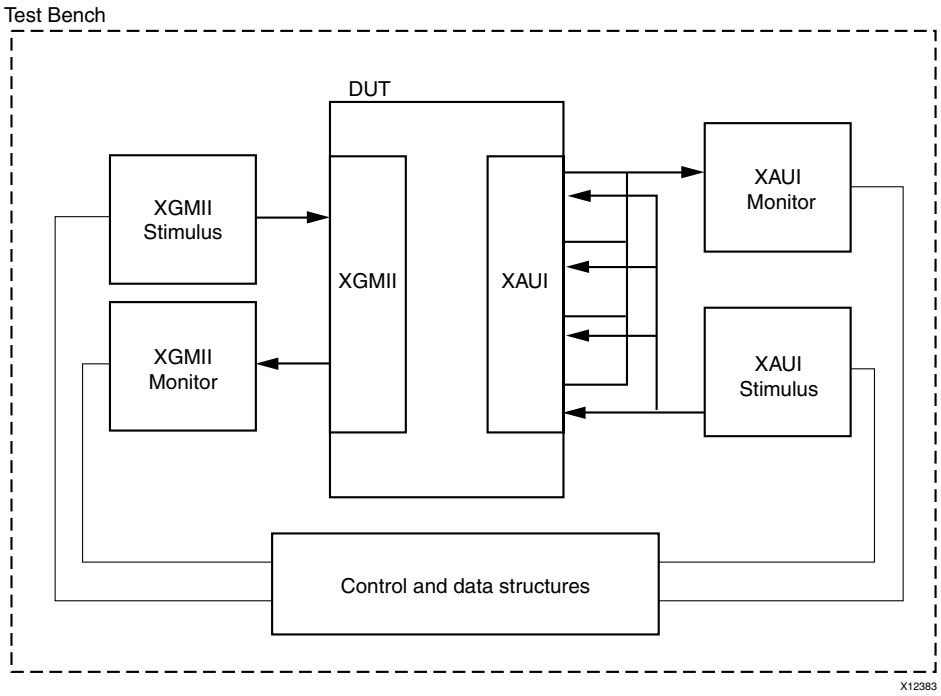

Figure 10-5: **Demonstration Test Bench for XAUI with XGMII Interface**

XAUI Core with Internal Client-Side Interface

Example HDL Wrapper

In Figure 10-6, the example HDL wrapper generated when the internal client-side interface is selected contains the following:

- The RocketIO transceiver instances
- Virtex-4 device RocketIO transceiver Calibration Blocks (see Answer Record 22477 for information about the *Calibration Block User Guide*)
- Initialization blocks for the transmit and receive RocketIO transceivers
- Clock management logic, including DCM and Global Clock Buffer instances
- Re-timing registers on the parallel data interface, both on input and output

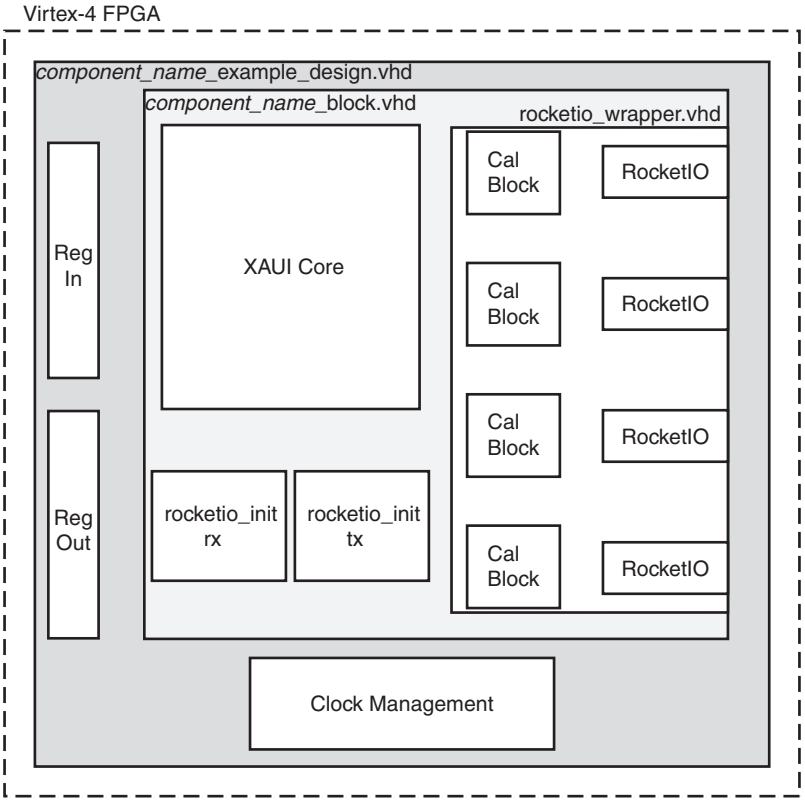

Figure 10-6: **Example HDL Wrapper for XAUI without XGMII (Virtex-4 FPGAs)**

In Figure 10-7, the example HDL wrapper generated when the internal client-side interface is selected contains the following:

- The RocketIO transceiver tile instances
- The wrapper for the two RocketIO transceiver tiles
- A RocketIO transceiver transmit initialization block
- Clock management logic, including DCM (if required) and Global and Clock Buffer instances
- Re-timing registers on the parallel data interface, both on input and output

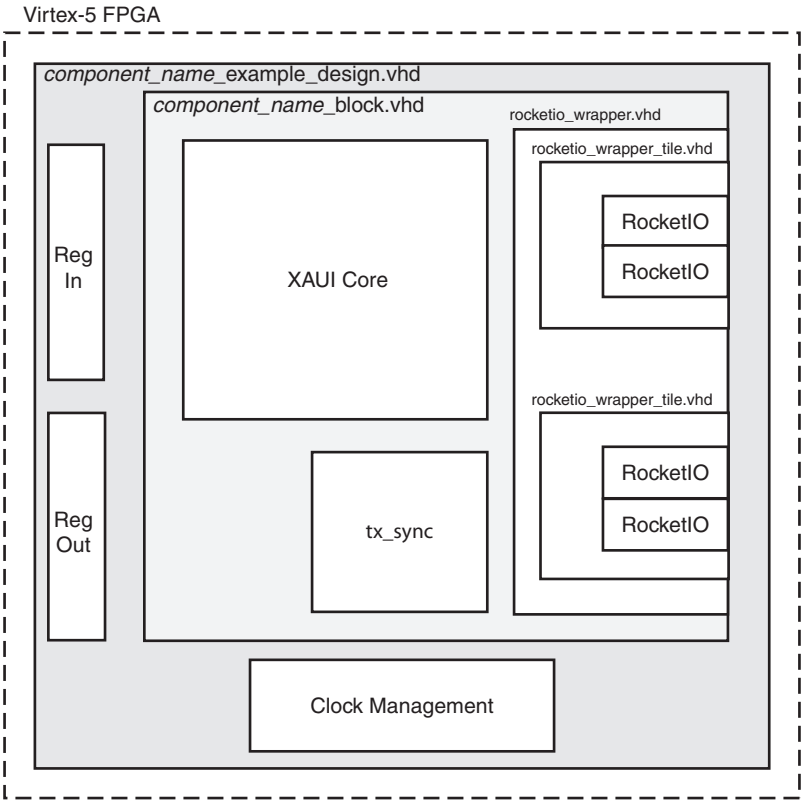

Figure 10-7: **Example HDL Wrapper for XAUI without XGMII (Virtex-5 FPGAs)**

In Figure 10-8, the example HDL wrapper generated when the internal client-side interface is selected contains the following:

- The Transceiver Instances
- The wrapper for the four transceivers
- A transceiver transmit initialization block
- Clock management logic and Clock Buffer instances
- Re-timing registers on the parallel data interface, both on inputs and outputs

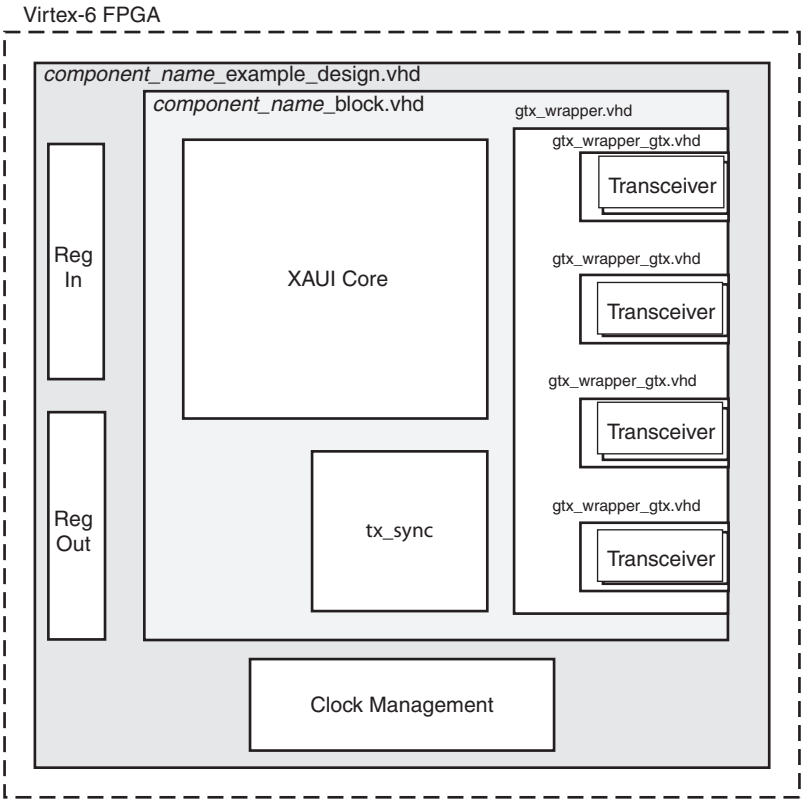

Figure 10-8: **Example HDL Wrapper for XAUI without XGMII (Virtex-6 FPGAs)**

In Figure 10-9, the example HDL wrapper generated when the internal client-side interface is selected contains the following:

- The transceiver and transceiver QPLL Instances
- A wrapper for the four transceivers and transceiver QPLL
- Clock management logic and Clock Buffer instances
- Re-timing registers on the parallel data interface, both on inputs and outputs

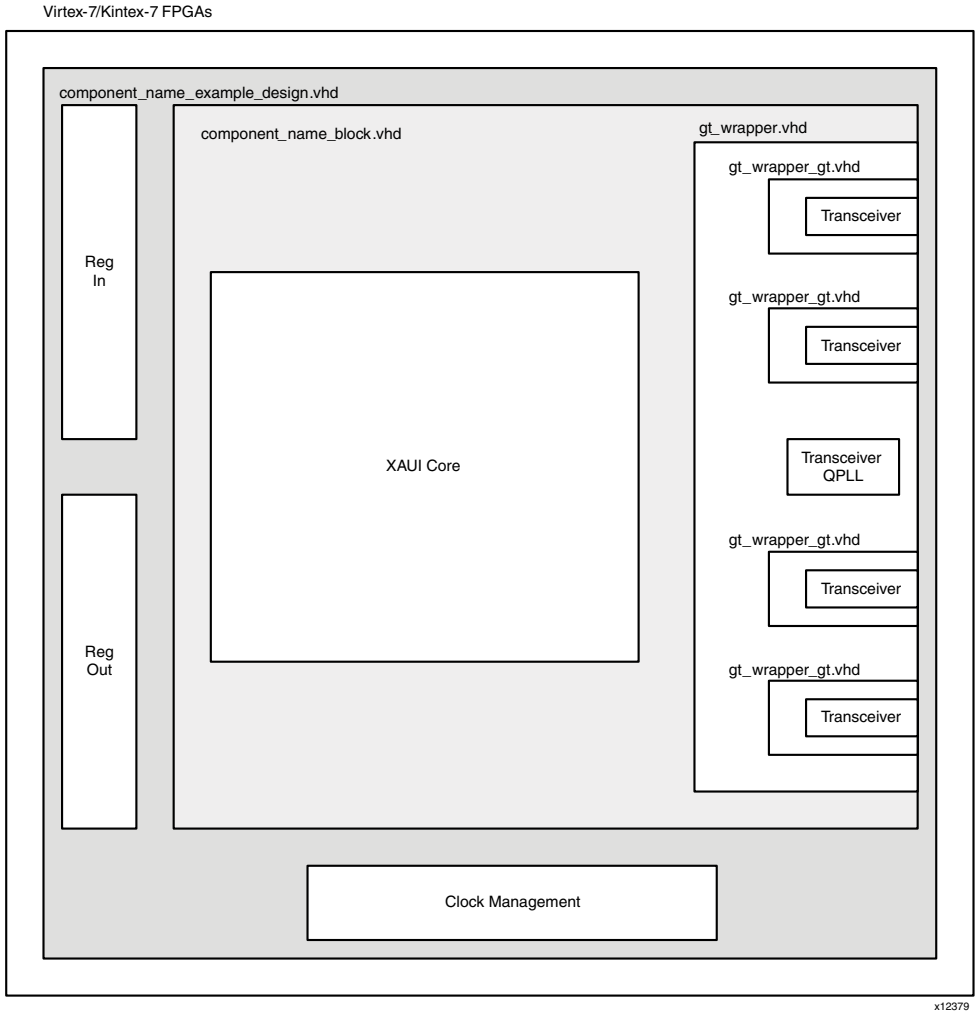

Figure 10-9: **Example HDL Wrapper for XAUI without XGMII (Virtex-7 and Kintex-7 FPGAs)**

In Figure 10-10, the example HDL wrapper generated when the internal client-side interface is selected contains the following:

- The Transceiver Instances
- The wrapper for the four transceivers
- A transceiver synchronization block
- Clock management logic and Clock Buffer instances
- Re-timing registers on the parallel data interface, both on inputs and outputs

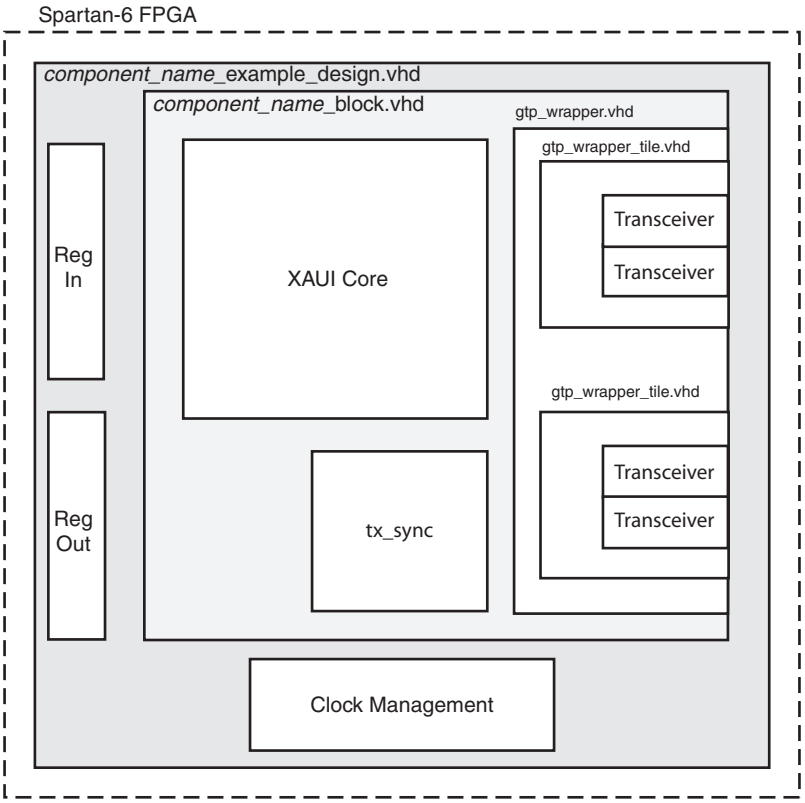

Figure 10-10: **Example HDL Wrapper for XAUI without XGMII (Spartan-6 FPGAs)**

Demonstration Test Bench

In Figure 10-11, the demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself. This test bench consists of transactor procedures or tasks that connect to the major ports of the example design, and a control program that pushes frames of varying length and content through the design and checks the values as they exit the core.

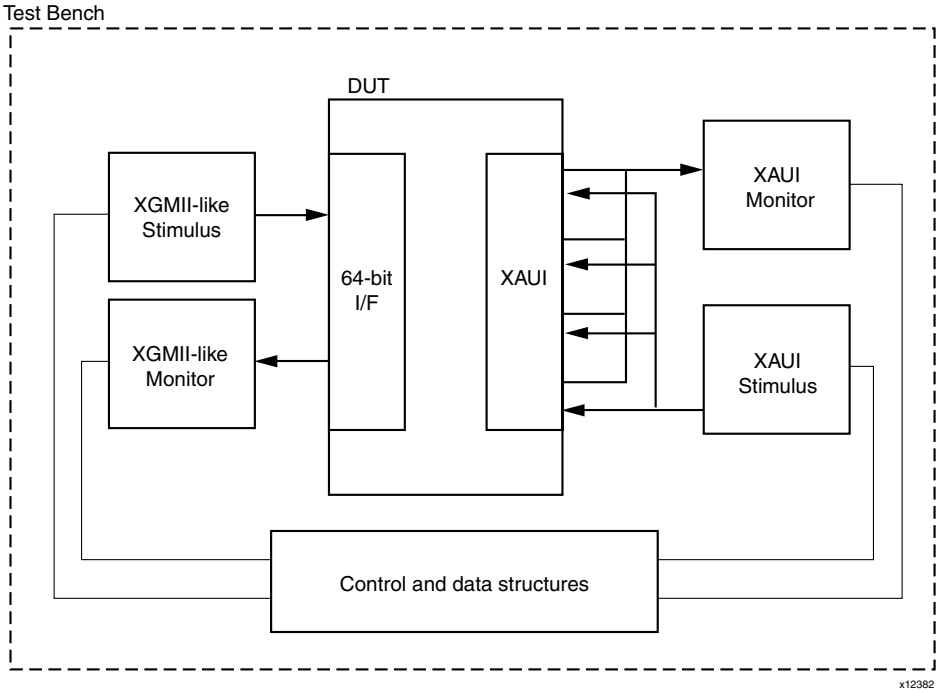

Figure 10-11: **Demonstration Test Bench for XAUI without XGMII Interface**

XILINX®

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/support/documentation/sw_manuals/glossary.pdf.

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.

Appendix B

Verification and Interoperability

The XAUI core has been verified using both simulation and hardware testing.

Simulation

A highly parameterizable transaction-based simulation test suite has been used to verify the core. Tests included:

- Register access over MDIO
- Loss and re-gain of synchronization
- Loss and re-gain of alignment
- Frame transmission
- Frame reception
- Clock compensation
- Recovery from error conditions

Hardware Testing

The core has been used in a number of hardware test platforms within Xilinx. In particular, the core has been used in a test platform design with the Xilinx® 10-Gigabit Ethernet MAC core. This design comprises the MAC, XAUI, a "ping" loopback FIFO, and a test pattern generator all under embedded PowerPC® processor control. This design has been used for conformance and interoperability testing at the University of New Hampshire Interoperability Lab. PCS reports are available from the factory on request.

XILINX®

Appendix C

Calculating the DCM/MMCM Phase Shift

DCM/MMCM Phase Shifting Requirement

A DCM/MMCM is used in the transmitter clock path to meet the input setup and hold requirements when using the core with an XGMII.

In these cases, a fixed phase shift offset is applied to the transmit clock DCM/MMCM to skew the clock; this performs static alignment by using the transmit clock DCM/MMCM to shift the internal version of the transmit clock such that its edges are centered on the data eye at the IOB DDR flip-flops. The ability to shift the internal clock in small increments is critical for sampling high-speed source synchronous signals such as XGMII. For statically aligned systems, the DCM/MMCM output clock phase offset (as set by the phase shift value) is a critical part of the system, as is the requirement that the PCB is designed with precise delay and impedance-matching for all the XGMII data bus and control signals.

You must determine the best DCM/MMCM setting (phase shift) to ensure that the target system has the maximum system margin to perform across voltage, temperature, and process (multiple chips) variations. Testing the system to determine the best DCM/MMCM phase shift setting has the added advantage of providing a benchmark of the system margin based on the UI (unit interval or bit time). System margin is defined as the following:

System Margin (ps) = UI(ps) * (working phase shift range/128)

Finding the Ideal Phase Shift Value for Your System

Xilinx cannot recommend a singular phase shift value that is effective across all hardware families. Xilinx does not recommend attempting to determine the phase shift setting empirically. In addition to the clock-to-data phase relationship, other factors such as package flight time (package skew) and clock routing delays (internal to the device) affect the clock to data relationship at the sample point (in the IOB) and are difficult to characterize.

Xilinx recommends extensive investigation of the phase shift setting during hardware integration and debugging. The phase shift settings provided in the example design constraint file is a placeholder, and works successfully in back-annotated simulation of the example design.

DCM Phase Shift Settings

Perform a complete sweep of phase shift settings during your initial system test. Use only positive (0 to 255) phase shift settings, and use a test range that covers a range of no less than 128, corresponding to a total 180 degrees of clock offset. This does not imply that 128 phase shift values must be tested; increments of 4 (52, 56, 60, and so forth) correspond to roughly one DCM tap, and consequently provide an appropriate step size. Additionally, it is not necessary to characterize areas outside the working phase shift range.

MMCM Phase Shift Settings

To change the MMCM phase shift value alter the CLKOUT0_PHASE attribute. This attribute is the number of degrees of clock offset required. Use only positive phase shift settings and use a test range that covers no less than 180 degrees.

At the edge of the operating phase shift range, system behavior changes dramatically. In eight phase shift settings or less, the system can transition from no errors to exhibiting errors. Checking the operational edge at a step size of two (on more than one board) refines the typical operational phase shift range. After the range is determined, choose the average of the high and low working phase shift values as the default. During the production test, Xilinx recommends that you re-examine the working range at corner case operating conditions to determine whether any final adjustments to the final phase shift setting are needed.

You can use the FPGA Editor to generate the required test file set instead of resorting to multiple Place and Route (PAR) runs. Performing the test on design files that differ only in phase shift setting prevents other variables from affecting the test results. FPGA Editor operations can even be scripted further, reducing the effort needed to perform this characterization.

Appendix D

Core Latency

These measurements are for the core only; they do not include the latency through the transceiver. The latency through the transceiver can be obtained from the relevant user guide.

Transmit Path Latency

As measured from the input port `xgmii_txd[63:0]` of the transmitter side XGMII (until that data appears on `mgt_txdata[63:0]` on the transceiver interface), the latency through the core for the internal XGMII interface configuration in the transmit direction is 3 clk periods of the core input usrclk.

There is an additional one clock cycle of `usrclk` TX pipelining in the Spartan®-6 xaui_block.v[hd] file if this is being used.

Receive Path Latency

Measured from the input into the core on mgt_rxdata[63:0] until the data appears on `xgmii_rxdata[63:0]` of the receiver side XGMII interface, the latency through the core in the receive direction is equal to 3-4 clock cycles of usrclk.

If the word appears on the upper half of the 2 byte transceiver interface, the latency is four clock cycles of usrclk and it appears on the lower half of the XGMII interface. If it appears on the lower half of the 2 byte interface, the latency is three clock cycles of `usrclk` and it appears on the upper half of the XGMII interface.

There is an additional one clock cycle of `usrclk` RX pipelining in the xaui_block.v[hd] file if this is being used.

XILINX®

Appendix E

Debugging Designs

This chapter provides information on using resources available on the Xilinx Support website, available debug tools, and a step-by-step process for debugging designs that use the XAUI core. The following information is found in this chapter:

- Finding Help on xilinx.com
- Contacting Xilinx Technical Support
- Debug Tools
- Simulation Specific Debug
- Hardware Debug

Finding Help on xilinx.com

To help in the design and debug process when using the XAUI core, the Xilinx Support web page (www.xilinx.com/support) contains key resources such as Product documentation, Release Notes, Answer Records, and links to opening a Technical Support case.

Documentation

In addition to this User Guide, there are the following XAUI publications:

- XAUI Data Sheet

These documents along with documentation related to all products that aid in the design process can be found on the Xilinx Support web page. Documentation is sorted by product family at the main support page or by solution at the Documentation Center.

To see the available documentation by device family:

1. Navigate to www.xilinx.com/support.
2. Select Virtex-6 from the Device List drop-down menu.

This sorts all available Virtex®-6 FPGA documentation by Hardware Documentation, Configuration Solutions Documentation, Related Software Documentation, Tools, and Data Files.

To see the available documentation by solution:

1. Navigate to www.xilinx.com/support.
2. Select the Documentation tab located at the top of the web page.

This is the Documentation Center where Xilinx documentation is sorted by Devices, Boards, IP, Design Tools, Doc Type, and Topic.

Release Notes and Known Issues

Known issues for all cores, including the XAUI core, are described in the IP Release Notes Guide.

Answer Records

Answer Records include information on commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a product. Answer Records are created and maintained daily ensuring that users have access to the most up-to-date information on Xilinx products. Answer Records can be found by searching the Answers Database.

To use the Answers Database Search:

1. Navigate to www.xilinx.com/support. The Answers Database Search is located at top of this web page.
2. Enter keywords in the provided search field and select **Search**.

- Examples of searchable keywords are product names, error messages, or a generic summary of the issue encountered.
- To see all answer records directly related to the XAUI core, search for the phrase "XAUI."

Contacting Xilinx Technical Support

Xilinx provides premier technical support for customers encountering issues that requires additional assistance.

To contact Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under **Support Quick Links**.

When opening a WebCase, include:

- Target FPGA including package and speed grade
- All applicable software versions of ISE® design suite, synthesis (if not XST), and simulator
- The XCO file created during generation of the LogiCORE™ IP wrapper. This file is located in the directory targeted for the CORE Generator™ tool project.

Additional files might be required based on the specific issue. See the relevant sections in this debug guide for further information on specific files to include with the WebCase.

Debug Tools

There are many tools available to debug XAUI design issues. It is important to know which tools are useful for debugging various situations that you encounter. This chapter references the following tools:

- Example Design
- ChipScope Pro Tool
- Available Reference Designs
- Link Analyzers

Example Design

The XAUI core comes with a synthesizable example design complete with functional and post-place and route simulation test benches. Information on the example design can be found in Chapter 10, Detailed Example Design.

ChipScope Pro Tool

The ChipScope™ Pro tool inserts logic analyzer, bus analyzer, and virtual I/O software cores directly into your design. The ChipScope Pro tool allows you to set trigger conditions to capture application and Integrated Block port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool. For detailed information on the ChipScope Pro tool, visit www.xilinx.com/chipscope.

Available Reference Designs

Xilinx Application Note 955 "10-Gigabit Ethernet Hardware Demonstration Platform" provides a demonstration design for the 10GEMAC and XAUI cores on the Virtex-5 FPGA ML523 and Virtex-4 FPGA ML421 boards.

Link Analyzers

Link Analyzers can be used to generate and analyzer traffic for hardware debug and testing. Common link analyzers include:

- SMARTBITS
- IXIA

Simulation Specific Debug

This section provides simulation debug flow diagrams for some of the most common issues experienced by users. Endpoints that are shaded gray indicate that more information can be found in sections after the figure.

ModelSim Debug

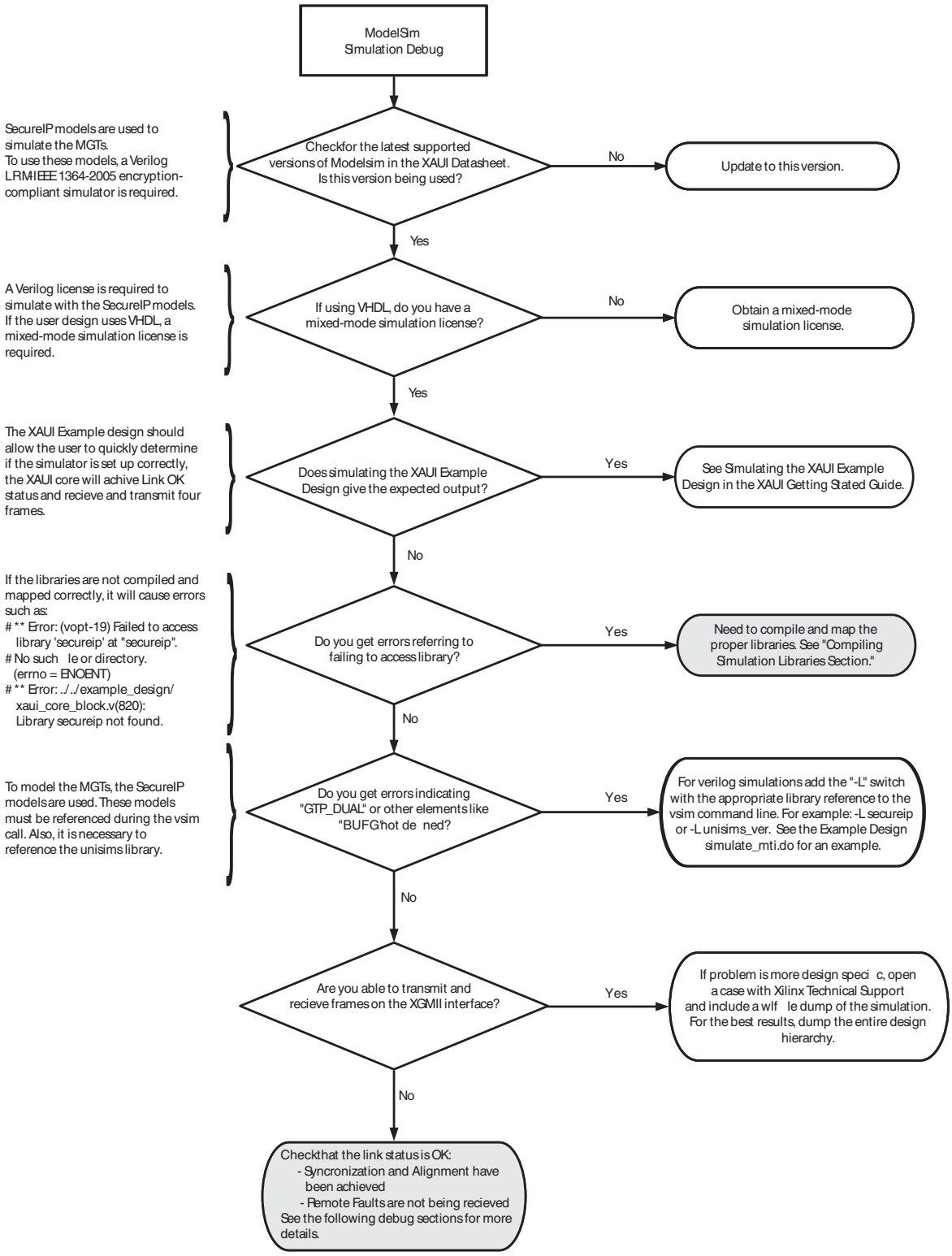

Figure E-1: **ModelSim Debug Flow Diagram**

Compiling Simulation Libraries

Compile the Xilinx simulation libraries, either by using the Xilinx Simulation Library Compilation Wizard, or by using the compxlib command line tool.

Xilinx Simulation Library Compilation Wizard

A GUI wizard provided as part of the Xilinx software can be launched to assist in compiling the simulation libraries by typing "compxlib" in the command prompt.

Compxlib

A compxlib command line can also be used to compile simulation libraries. This tool is delivered as part of the Xilinx software. For more information see the ISE Software Manuals and specifically the *Command Line Tools User Guide* under the section titled compxlib.

Assuming the Xilinx and ModelSim environments are set up correctly, this is an example of compiling the SecureIP and UNISIM libraries for Verilog into the current directory.

```
compxlib -s mti_se -arch virtex6 -l verilog -lib secureip -lib unisims
-dir ./
```

There are many other options available for compxlib described in the *Command Line Tools User Guide*.

Compxlib produces a modelsim.ini file containing the library mappings. In ModelSim, to see the current library mappings, type "vmap" at the prompt. The mappings can be updated in the ini file or to map a library at the ModelSim prompt type:

```
vmap [<logical_name>] [<path>]
```

For example:

```
vmap unisims_ver C:\my_unisim_lib
```

Next Step

If the debug suggestions listed previously do not resolve the issue, open a support case to have the appropriate Xilinx expert assist with the issue.

To create a technical support case in WebCase, see the Xilinx website at:

www.xilinx.com/support/clearexpress/websupport.htm

Items to include when opening a case:

- Detailed description of the issue and results of the steps listed previously.
- Attach a VCD or WLF dump of the simulation.

To discuss possible solutions, use the Xilinx User Community:

forums.xilinx.com/xlnx/

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The ChipScope tool is a valuable resource to use in hardware debug and the signal names mentioned in the following individual sections can be probed using the ChipScope tool for debugging the specific problems. Many of these common issue can also be applied to debugging design simulations.

General Checks

Ensure that all the timing constraints for the core were met during Place and Route.

- Does it work in timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue.
- Ensure that all clock sources are clean. If using DCMs in the design, ensure that all DCMs have obtained lock by monitoring the LOCKED port.

Monitoring the XAUI Core with ChipScope Tool

- XGMII signals and signals between XAUI core and the transceiver can be added to monitor data transmitted and received. See Table 2-1, page 19 and Table 2-2, page 19 for a list of signal names.
- Status signals added to check status of link: `STATUS_VECTOR[7:0]`, `ALIGN_STATUS`, and `SYNC_STATUS[3:0]`.
- To interpret control codes in on the XGMII interface or the interface to the transceiver, see Table E-1 and Table E-2.
- An Idle (0x07) on the XGMII interface is encoded to be a randomized sequence of /K/ (Sync), /R/ (Skip), /A/(Align) codes on the XAUI interface. For more information on this encoding, see the IEEE 802.3-2008 specification (section 48.2.4.2) for more details.

Table E-1: **XGMII Control Codes**

TXC	TXD	Description
0	0x00 through 0xFF	Normal data transmission
1	0x07	Idle
1	0x9C	Sequence
1	0xFB	Start
1	0xFD	Terminate
1	0xFE	Error

Table E-2: **XAUI Control Codes**

Codegroup	8-bit value	Description
Dxx.y	0xXX	Normal data transmission
K28.5	0xBC	/K/ (Sync)
K28.0	0x1C	/R/ (Skip)
K28.3	0x7C	/A/ (Align)
K28.4	0x9C	/Q/ (Sequence)
K27.7	0xFB	/S/ (Start)
K29.7	0xFD	/T/ (Terminate)
K30.7	0xFE	/E/ (Error)

Problems with Data Reception or Transmission

Problems with data reception or transmission can be caused by a wide range of factors. Following is a flow diagram of steps to debug the issue. Each of the steps are discussed in more detail in the following sections.

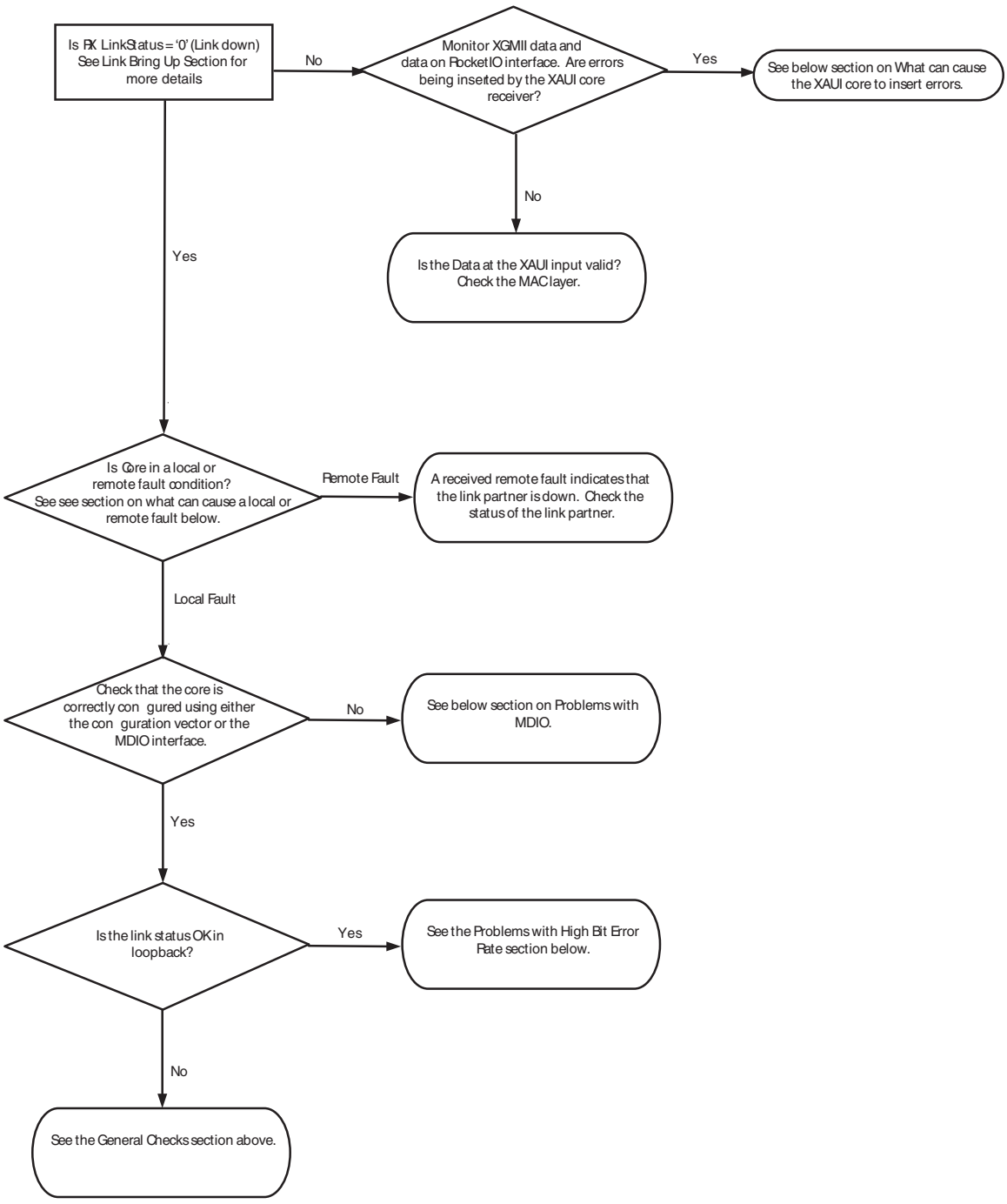

Figure E-2: **Flow Diagram for Debugging Problems with Data Reception or Transmission**

What Can Cause a Local or Remote Fault?

Local Fault and Remote Fault codes both start with the sequence TXD/RXD=0x9C, TXC/RXC=1 in XGMII lane 0. Fault conditions can also be detected by looking at the status vector or MDIO registers. The Local Fault and Link Status are defined as latching error indicators by the IEEE specification. This means that the Local Fault and Link Status bits in the status vector or MDIO registers must be cleared with the Reset Local Fault bits and Link Status bits in the Configuration vector or MDIO registers.

Local Fault

The receiver outputs a local fault when the receiver is not up and operational. This rx local fault is also indicated in the status and MDIO registers. The most likely causes for an rx local fault are:

- The transceiver has not locked or the receiver is being reset.
- At least one of the lanes is not synchronized - `SYNC_STATUS[3:0]`
- The lanes are not properly aligned - `ALIGN_STATUS`

Note: The `SYNC_STATUS` and `ALIGN_STATUS` signals are not latching.

A tx local fault is indicated in the status and MDIO registers when the transceiver transmitter is in reset or has not yet completed any other initialization or synchronization procedures needed.

Remote Fault

Remote faults are only generated in the MAC reconciliation layer in response to a Local Fault message. When the receiver receives a remote fault, this means that the link partner is in a local fault condition.

When the MAC reconciliation layer receives a remote fault, it silently drops any data being transmitted and instead transmits IDLEs to help the link partner resolve its local fault condition. When the MAC reconciliation layer receives a local fault, it silently drops any data being transmitted and instead transmits a remote fault to inform the link partner that it is in a fault condition. Be aware that the Xilinx 10GEMAC core has an option to disable remote fault transmission.

Link Bring Up

The following link initialization stages describe a possible scenario of the Link coming up between device A and device B.

Stage 1: Device A Powered Up, but Device B Powered Down

- Device A is powered up and reset.
- Device B powered down
- Device A detects a fault because there is no signal received. The Device A XAUI core indicates an rx local fault.
- The Device A MAC reconciliation layer receives the local fault. This triggers the MAC reconciliation layer to silently drop any data being transmitted and instead transmit a remote fault.
- RX Link Status = '0' (link down) in Device A

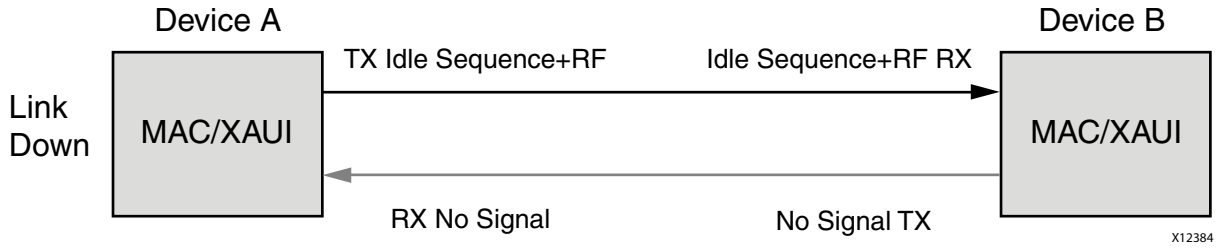

Figure E-3: **Device A Powered Up, but Device B Powered Down**

Stage 2: Device B Powers Up and Resets

- Device B powers up and resets.
- Device B XAUI completes Synchronization and Alignment.
- Device A has not synchronized and aligned yet. It continues to send remote faults.
- Device B XAUI passes received remote fault to MAC.
- Device B MAC reconciliation layer receives the remote fault. It silently drops any data being transmitted and instead transmits IDLEs.
- Link Status = '0' (link down) in both A and B.

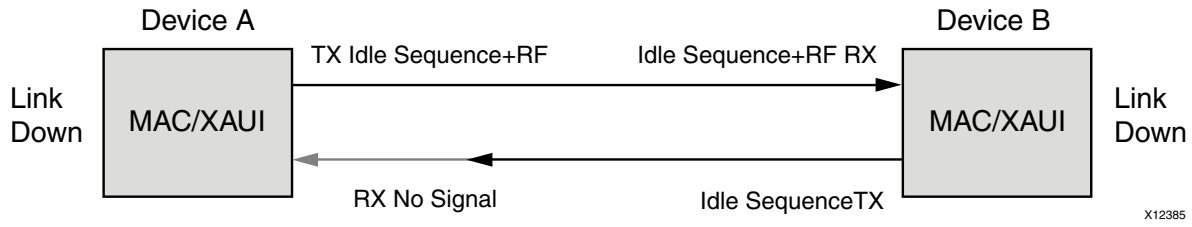

Figure E-4: **Device B Powers Up and Resets**

Stage 3: Device A Receives Idle Sequence

- Device A XAUI RX detects idles, synchronizes and aligns.
- Device A reconciliation layer stops dropping frames at the output of the MAC transmitter and stops sending remote faults to Device B.
- Device A Link Status='1' (Link Up)
- When Device B stops receiving the remote faults, normal operation starts.

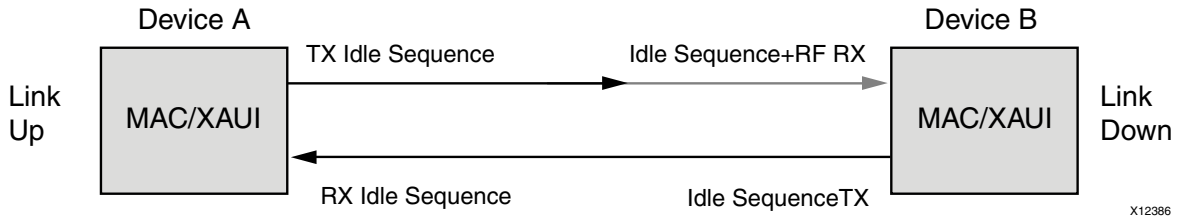

Figure E-5: **Device A Receives Idle Sequence**

Stage 4: Normal Operation

In Stage 4 shown in Figure E-6, Device A and Device B have both powered up and been reset. The link status is '1' (link up) in both A and B and in both the MAC can transmit frames successfully.

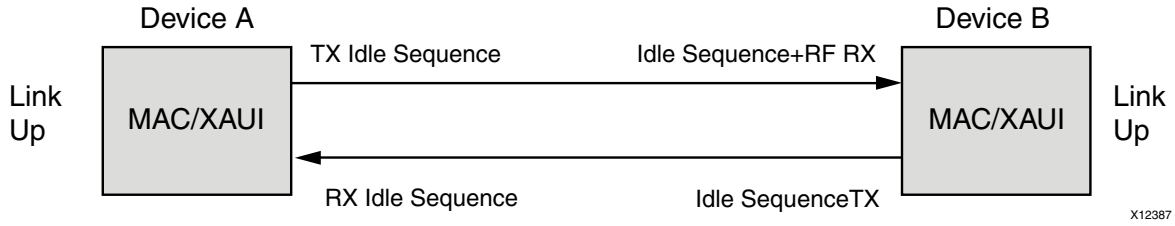

Figure E-6: **Normal Operation**

What Can Cause Synchronization and Alignment to Fail?

Synchronization (`sync_status[3:0]`) occurs when each respective XAUI lane receiver is synchronized to byte boundaries. Alignment (`align_status`) occurs when the XAUI receiver is aligned across all four lanes.

Following are suggestions for debugging loss of Synchronization and Alignment:

- Monitor the state of the `SIGNAL_DETECT[3:0]` input to the core. This should either be:
- connected to an optical module to detect the presence of light. Logic '1' indicates that the optical module is correctly detecting light; logic '0' indicates a fault. Therefore, ensure that this is driven with the correct polarity.
- tied to logic '1' (if not connected to an optical module).

Note: When `signal_detect` is set to logic '0,' this forces the receiver synchronization state machine of the core to remain in the loss of sync state.

- Loss of Synchronization can happen when invalid characters are received.
- Loss of Alignment can happen when invalid characters are seen or if an /A/ code is not seen in all four lanes at the same time.
- See the following section, Problems with a High Bit Error Rate.

Transceiver Specific

- Ensure that the polarities of the TXN/TXP and RXN/RXP lines are not reversed. If they are, these can be fixed by using the `TXPOLARITY` and `RXPOLARITY` ports of the transceiver.
- Check that the transceiver is not being held in reset or still be initialized by monitoring the `mgt_tx_reset`, `mgt_rx_reset`, and `mgt_rxlock` input signals to the XAUI core. The `mgt_rx_reset` signal is also asserted when there is an rx buffer error. An rx buffer error means that the Elastic Buffer in the receiver path of the transceiver is either under or overflowing. This indicates a clock correction issue caused by differences between the transmitting and receiving ends. Check all clock management circuitry and clock frequencies applied to the core and to the transceiver.

What Can Cause the XAUI Core to Insert Errors?

On the receive path the XAUI core will insert errors RXD=FE, RXC=1, when disparity errors or invalid data are received or if the received interframe gap (IFG) is too small.

Disparity Errors or Invalid Data

Disparity Errors or Invalid data can be checked for by monitoring the `mgt_code_valid` input to the XAUI core.

Small IFG

The XAUI Core inserts error codes into the Received XGMII data stream, RXD, when there are three or fewer IDLE characters (0x07) between frames. The error code (0xFE) precedes the frame's "Terminate" delimiter (0xFD).

The IEEE 802.3-2008 specification (Section 46.2.1) requires a minimum interframe gap of five octets on the receive side. This includes the preceding frame's Terminate control character and all Idles up to and immediately preceding the following frame's Start control character. Because three (or fewer) Idles and one Terminate character are less than the required five octets, this would not meet the specification; therefore, the XAUI Core is expected to signal an error in this manner if the received frame does not meet the specification.

Problems with a High Bit Error Rate

Symptoms

If the link comes up but then goes down again or never comes up following a reset, the most likely cause for a Rx Local Fault is a BER (Bit Error Rate) that is too high. A high BER causes incorrect data to be received, which leads to the lanes losing synchronization or alignment.

Debugging

Compare the issue across several devices or PCBs to ensure that the issue is not a one-off case.

- Try using an alternative link partner or test equipment and then compare results.
- Try putting the core into loopback (both by placing the core into internal loopback, and by looping back the optical cable) and compare the behavior. The core should always be capable of gaining synchronization and alignment when looping back with itself from transmitter to receiver so direct comparisons can be made. If the core exhibits correct operation when placed into internal loopback, but not when loopback is performed via an optical cable, this might indicate a faulty optical module or a PCB issue.
- Try swapping the optical module on a misperforming device and repeat the tests.

Transceiver Specific Checks

- Monitor the `MGT_CODEVALID`[7:0] input to the XAUI core by triggering on it using the ChipScope tool. This input is a combination of the transceiver rx disparity error and rx not in table error outputs.
- These signals should not be asserted over the duration of a few seconds, minutes or even hours. If they are frequently asserted, it might indicate an issue with the transceiver.
- Place the transceiver into parallel or serial near-end loopback.
- If correct operation is seen in the transceiver serial loopback, but not when loopback is performed via an optical cable, it might indicate a faulty optical module.
- If the core exhibits correct operation in the transceiver parallel loopback but not in serial loopback, this might indicate a transceiver issue.
- A mild form of bit error rate might be solved by adjusting the transmitter Pre-Emphasis and Differential Swing Control attributes of the transceiver.

Problems with the MDIO

See MDIO Interface for detailed information about performing MDIO transactions.

Things to check for:

- Ensure that the MDIO is driven properly. Check that the `mdc` clock is running and that the frequency is 2.5 MHz or less.
- Ensure that the XAUI core is not held in reset.
- Read from a configuration register that does not have all 0s as a default. If all 0s are read back, the read was unsuccessful. Check that the PRTAD field placed into the MDIO frame matches the value placed on the PRTAD[4:0] port of the XAUI core.
- Verify in simulation and/or a ChipScope capture that the waveform is correct for accessing the host interface for a MDIO read/write.

Next Steps

If the debug suggestions listed previously do not resolve the issue, open a support case to have the appropriate Xilinx expert assist with the issue.

To create a technical support case in Webcase, see the Xilinx website at:

www.xilinx.com/support/clearexpress/websupport.htm

Items to include when opening a case:

- Detailed description of the issue and results of the steps listed previously.
- Attach ChipScope VCD captures taken in the steps previously.

To discuss possible solutions, use the Xilinx User Community:

forums.xilinx.com/xlnx/