LogiCORE™ XAUI V6.2

User Guide

UG150 July 13, 2006

XILINX®

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 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.
7/13/06	2.3	Updated to core version 6.2 and Xilinx software 8.2i.

Table of Contents

Chapter 7: Constraining the Core

Chapter 8: Design Considerations

Chapter 9: Implementing the Core

Appendix A: Related Information

Appendix B: Core Directory Structure

Appendix C: Verification and Interoperability

Appendix D: Calculating the DCM Phase Shift

Schedule of Figures

Chapter 1: Introduction

Chapter 2: Installing and Licensing the Core

Chapter 3: Core Architecture

Chapter 4: Customizing and Generating the Core

Chapter 5: Designing with the Core

Chapter 6: Interfacing to the Core

Chapter 7: Constraining the Core

Chapter 8: Design Considerations

Appendix B: Core Directory Structure

Appendix C: Verification and Interoperability

Schedule of Tables

Chapter 1: Introduction

Chapter 2: Installing and Licensing the Core

Chapter 3: Core Architecture

Chapter 4: Customizing and Generating the Core

Chapter 5: Designing with the Core

Chapter 6: Interfacing to the Core

About This Guide

The *XAUI V6.2 User Guide* provides information about generating a LogiCORE™ XAUI core, customizing and simulating the core utilizing the provided example design, and running the design files through implementation using the Xilinx tools.

Guide Contents

This guide contains the following chapters:

- Preface, "About This Guide" introduces the user to the organization and purpose of the design guide, a list of additional resources and the conventions used in this document.
- Chapter 1, "Introduction" introduces the XAUI core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Installing and Licensing the Core" provides instructions for installing the XAUI core and obtaining a license for the core, which you must do before using the core in your designs.
- Chapter 3, "Core Architecture" describes the overall architecture of the XAUI core and also describes the major interfaces to the core.
- Chapter 4, "Customizing and Generating the Core" describes how to customize the XAUI core for specific applications and generate the core netlist using Xilinx CORE Generator.
- Chapter 5, "Designing with the Core" contains a general description of how to use the XAUI core in your own design.
- Chapter 6, "Interfacing to the Core" defines the data interfaces and the configuration and status interfaces available for dynamically setting configuration and status.
- Chapter 7, "Constraining the Core" describes how to constrain a design, illustrated by the default user constraints file (UCF) included with the XAUI core.
- Chapter 8, "Design Considerations" describes considerations that may apply in specific design cases.
- Chapter 9, "Implementing the Core" describes how to simulate and implement your design containing the XAUI core.
- Appendix A, "Related Information" provides links to support resources and XAUI core related information.
- Appendix B, "Core Directory Structure" describes the files generated by CORE Generator for the XAUI core.
- Appendix C, "Verification and Interoperability" describes the XAUI verification methods in simulation and hardware testing environments.

- Appendix D, "Calculating the DCM Phase Shift" describes how a DCM is used in the transmitter clock path to meet the input setup and hold requirements when using the core with an XGMII.

Additional Resources

For additional information, go to http://support.xilinx.com. The following table lists some of the resources you can access from this website. You can also directly access these resources using the provided URLs.

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

Conventions

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

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands you enter in a syntactical statement	**`ngdbuild`** *`design_name`*

Convention	Meaning or Use	Example
Italic font	Variables in a syntax statement for which you must supply values	See the *Development System Reference Guide* for more information.
	References to other manuals	See the *User Guide* for details.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **bus[7:0]**, they are required.	**ngdbuild** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**lowpwr** ={**on**|**off**}
Vertical bar |	Separates items in a list of choices	**lowpwr** ={**on**|**off**}
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis . . .	Repetitive material that has been omitted	**allow block** *block_name loc1 loc2 ... locn;*
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	A '_n' means the signal is active low	usr_teof_n is active low.

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. See "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Chapter 1

Introduction

The LogiCORE™ XAUI core is a fully verified solution design that supports Verilog-HDL and 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.

About the Core

The XAUI core is a Xilinx CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see http://www.xilinx.com/systemio/xaui/index.htm. For information about system requirements, installation, and licensing options, see Chapter 2, "Installing and Licensing the Core."

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 FPGA designs using Xilinx implementation software and UCF is recommended.

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

Additional Core Resources

For detailed information and updates about the XAUI core, see the following documents, located on the XAUI product page at http://www.xilinx.com/systemio/xaui/index.htm

- *XAUI Data Sheet*
- *XAUI Release Notes*
- *XAUI User Guide*

For updates to this document, see the *LogiCORE XAUI Getting Started Guide*, also located on the XAUI product page.

Technical Support

To obtain technical support specific to the XAUI core, visit http://support.xilinx.com/. Questions are routed to a team of engineers with expertise using the XAUI core. In addition, support resources such as Application Notes, User Guides, Reference and Software Manuals, and Answer Records can also be accessed from the support site.

Xilinx will provide technical support for use of this product as described in the *LogiCORE XAUI User Guide* and the *LogiCORE XAUI Getting Started Guide*. 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.

XAUI Core

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

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

Document

For comments or suggestions about this document, please submit a webcase from http://support.xilinx.com/. 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

Installing and Licensing the Core

This chapter provides instructions for installing the XAUI core and obtaining a license for the core, which you must do before using the core in your designs. The XAUI core is provided under the terms of the Xilinx LogiCORE Site License Agreement, which conforms to the terms of the SignOnce IP License standard defined by the Common License Consortium.

System Requirements

Windows

- Windows® 2000 Professional with Service Pack 2-4
- Windows XP Professional with Service Pack 1

Solaris/Linux

- Sun Solaris® 8/9
- Red Hat® Enterprise Linux 3.0 (32-bit and 64-bit)

Software

- ISE™ 8.2i with applicable Service Pack

Check the release notes for the required Service Pack; ISE Service Packs can be downloaded from www.xilinx.com/xlnx/xil_sw_updates_home.jsp?update=sp.

Before you Begin

Before installing the core, you must have a Xilinx.com account and the ISE 8.2i software installed on your system. If you have already completed these steps, go to "Installing the Core."

1. Click Login at the top of the Xilinx home page; then follow the onscreen instructions to create a support account.
2. Install the ISE 8.2i software and the applicable Service Pack software.

Installing the Core

You can install the core in two ways—using the CORE Generator IP Software Update option to select from a list of updates, or by performing a manual installation after downloading the core from the web.

Using the CORE Generator Software Update Installer

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

1. Start the CORE Generator; then open an existing project or create a new one.
2. From the main CORE Generator window, choose Tools > Software Update. The WebUpdate screen appears.
3. If you are behind a firewall, click Set Proxy to either verify or set your proxy host and port settings.
4. Click Check for Updates. The Software Update installer appears.
5. Select the ISE 8.2IP Update 1 option; then click Install Selected. Informational messages may appear indicating that additional installations are required.
6. Click OK to accept any messages and continue. The User Login dialog box appears.
7. Enter your login name and password; then click OK. The selected update products are downloaded and installed.
8. To confirm the installation, check the following file: `C:\Xilinx\coregen\install\install_history`.

   Note that this step assumes your Xilinx software is installed in C:\Xilinx.

Manually

1. Close the CORE Generator if it is running.
2. Download the IP Update ZIP file from the following location and save it to a temporary directory: www.xilinx.com/support/download.htm.
3. Unpack the ZIP files using either WinZip (Windows) or Unzip (UNIX).
4. Extract the **ise_82i_ip_update1.zip** archive to the root directory of your Xilinx software installation. (Allow the extractor utility you use to overwrite all existing files and maintain the directory structure defined in the archive.)
5. If you do not have a zip utility, do one of the following:
   - **Windows**. From a command window, type the following:

     ```
     %XILINX%/bin/nt/unzip -d %XILINX% ise_82i_ip_update1.zip
     ```

   - **Linux**. From a UNIX shell, type the following:

     ```
     $XILINX/bin/lin/unzip -d $XILINX ise_82i_ip_update1.zip
     ```

   - **Solaris**. From a UNIX shell, type the following:

     ```
     $XILINX/bin/sol/unzip -d $XILINX ise_82i_ip_update1.zip
     ```

6. To verify the root directory of your Xilinx installation, do one of the following:
   - **Windows**. Type `echo %XILINX%` from a DOS prompt.
   - **UNIX**. If you have already installed the Xilinx ISE software, the Xilinx variable defined by your set-up script identifies the location of the Xilinx installation directory. After sourcing the Xilinx set-up script, type `echo $XILINX` to determine the location of the Xilinx installation.

Verifying your Installation

1. Start the CORE Generator.
2. After creating a new project or opening an existing one, the IP core functional categories appear at the left side of the window.

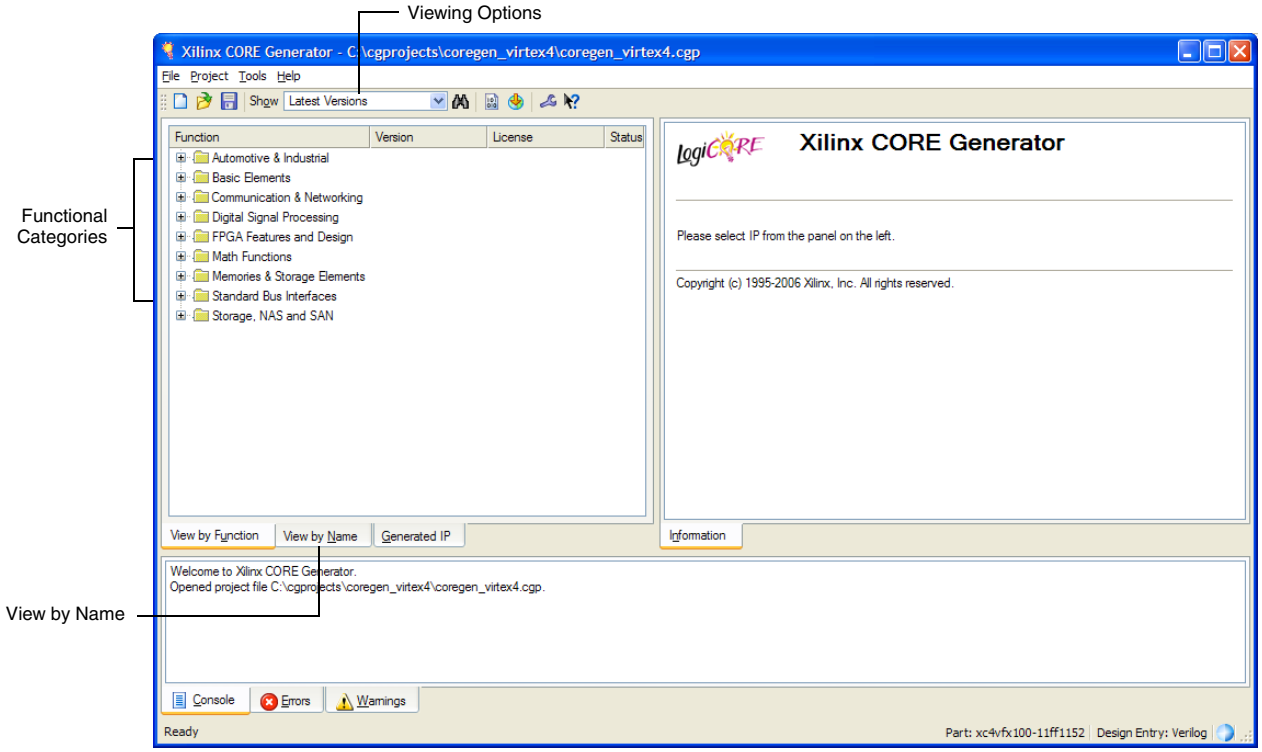

Figure 2-1: **CORE Generator Window**

3. Click to expand or collapse the view of individual functional categories, or click the View by Name tab at the bottom of the list to see an alphabetical list of all cores in all categories.
4. To view specific versions of the cores, choose an option from the Show drop-down list at the top of the window:
   - **Latest Versions**. Display the latest versions of all cores.
   - **All Versions**. Display all versions of cores, including new cores and new versions of cores.
   - **All Versions including Obsolete**. Display all cores, including those scheduled to become obsolete.
5. To determine that the installation is successful, be sure that the new core or cores appear in the CORE Generator GUI.

   For additional assistance installing the IP Update, contact www.xilinx.com/support.

License Options

The XAUI core provides two licensing options: a Simulation Only Evaluation license and a Full license. After installing the core, the Simulation Only Evaluation license is provided by default with the CORE Generator. To install the Full License, follow the directions below.

Simulation Only

The Simulation Only Evaluation license is provided with the XAUI core from the Xilinx CORE Generator tool. This license lets you assess the core functionality with either the provided example design or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated gate-level netlist).

Full

The Full license provides full access to all core functionality both in simulation and in hardware, including:

- Gate-level functional simulation support
- Back annotated gate-level simulation support
- Full implementation support including place and route and bitstream generation
- Full functionality in the programmed device with no time outs

Obtaining Your License

Simulation Only Evaluation License

No action is required because the Simulation Only Evaluation license is provided with the CORE Generator.

Obtaining a Full License

To obtain a Full license, you must register for access to the *lounge*, a secured area of the XAUI product page.

- From the product page, click Register to register and request access to the lounge. Access to the lounge is automatic and granted immediately.
- After you receive confirmation of lounge access, click Access Lounge on the XAUI product page and log in.
- Click Access Lounge on the product lounge page and fill out the license request form linked from this location; then click Submit to automatically generate the license. An e-mail containing the license and installation instructions will be sent immediately to the email address you specified.

Installing Your License File

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

Chapter 3

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 4-lane, 3.125 Gbps-per-lane serial interface. Each lane is a differential pair, carrying current mode logic (CML) signalling and 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 de-skew 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. 802.3ae-2002*.

The XAUI standard was initially developed as a means to extend the physical separation possible between MAC and 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 XGMII (Ten Gigabit Ethernet Media Independent Interface). Figure 3-1 shows the XAUI core being used to connect to a 10-Gigabit XPAK optical module.

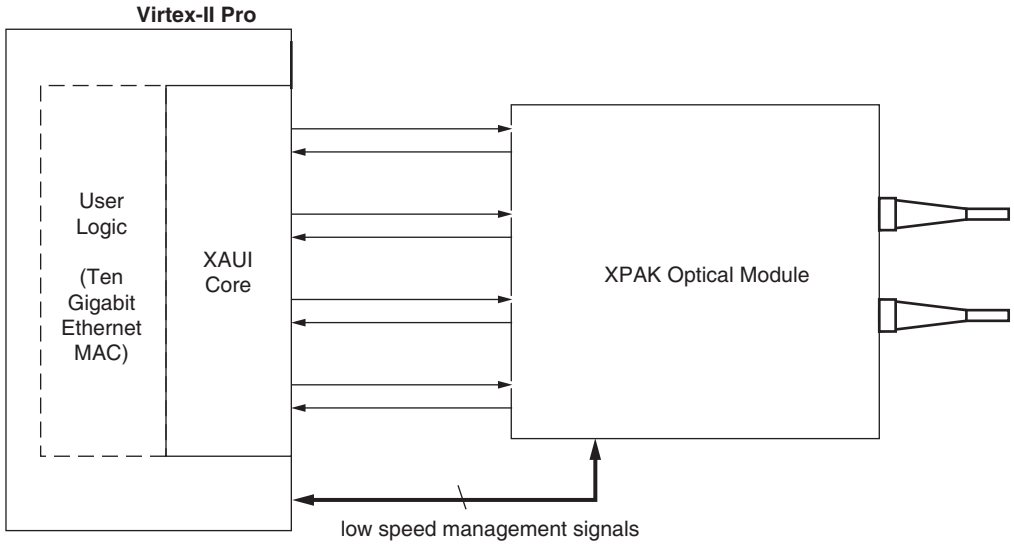

Figure 3-1: **Connecting XAUI to an Optical Module**

Since its publication, the applications of XAUI have extended beyond 10-Gigabit Ethernet to backplane and other general high-speed interconnect applications. A typical backplane application is shown in Figure 3-2.

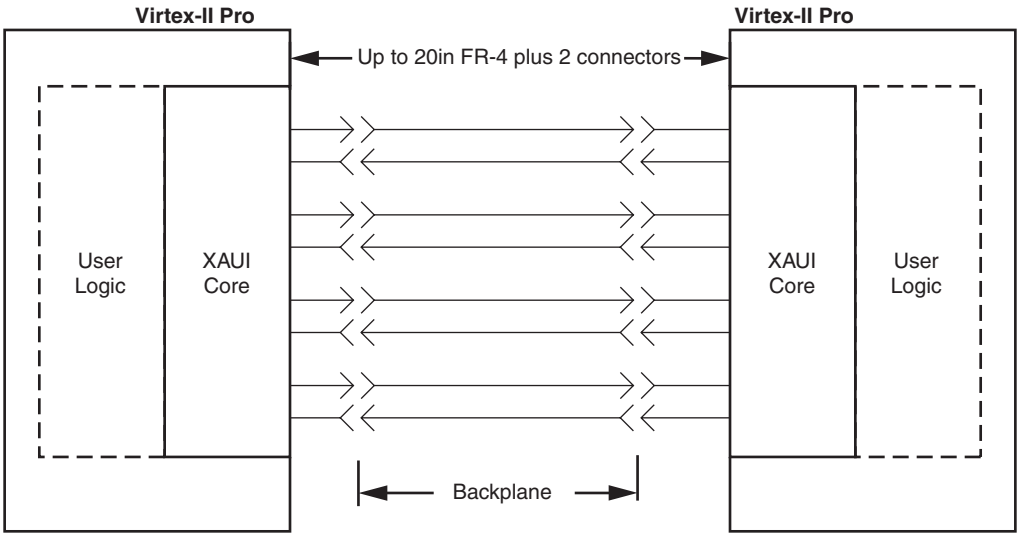

Figure 3-2: **Typical Backplane Application for XAUI**

Functional Description

Figure 3-3 shows a block diagram of the implementation of the XAUI core. The architecture is similar for both Virtex-II Pro and Virtex-4 cores. The major functional blocks of the core include the following:

- **Client-side interface**. If necessary, converts 32-bit DDR data into 64-bit 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.
- **De-skew state machine**. De-skews the four received lanes into alignment.
- **Optional MDIO interface**. A 2-wire low-speed serial interface used to manage the core.
- **Four RocketIO™ transceivers embedded in the Virtex-II Pro or Virtex-4 device**. Provides the high-speed transceivers as well as 8B/10B encode and decode and elastic buffering in the receive data path.

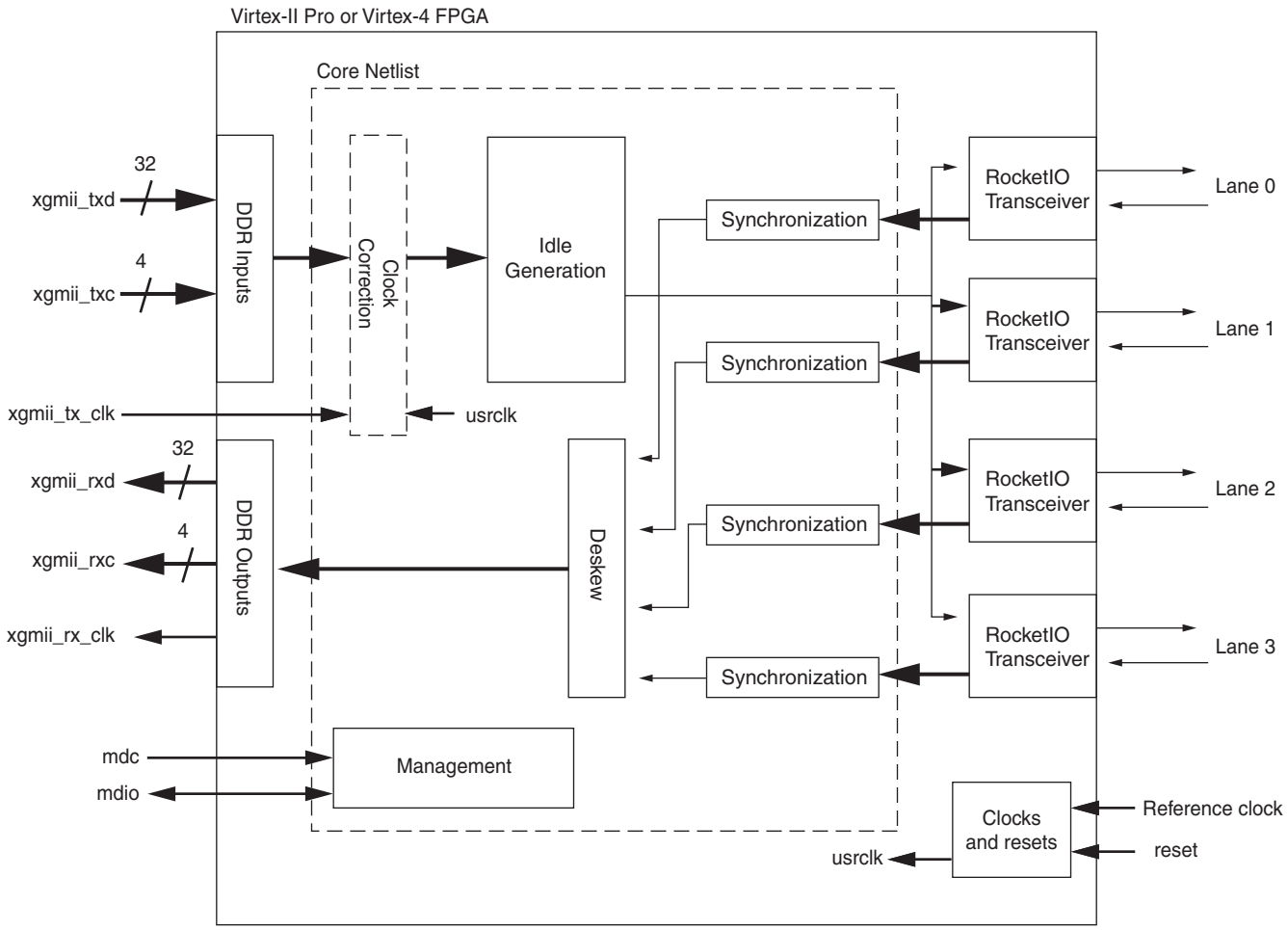

Figure 3-3: **Architecture of the XAUI Core with External XGMII Interface**

A significant number of customer applications will not require an external XGMII interface, but will instead will add user logic on the client-side interface. This application architecture is shown in Figure 3-4.

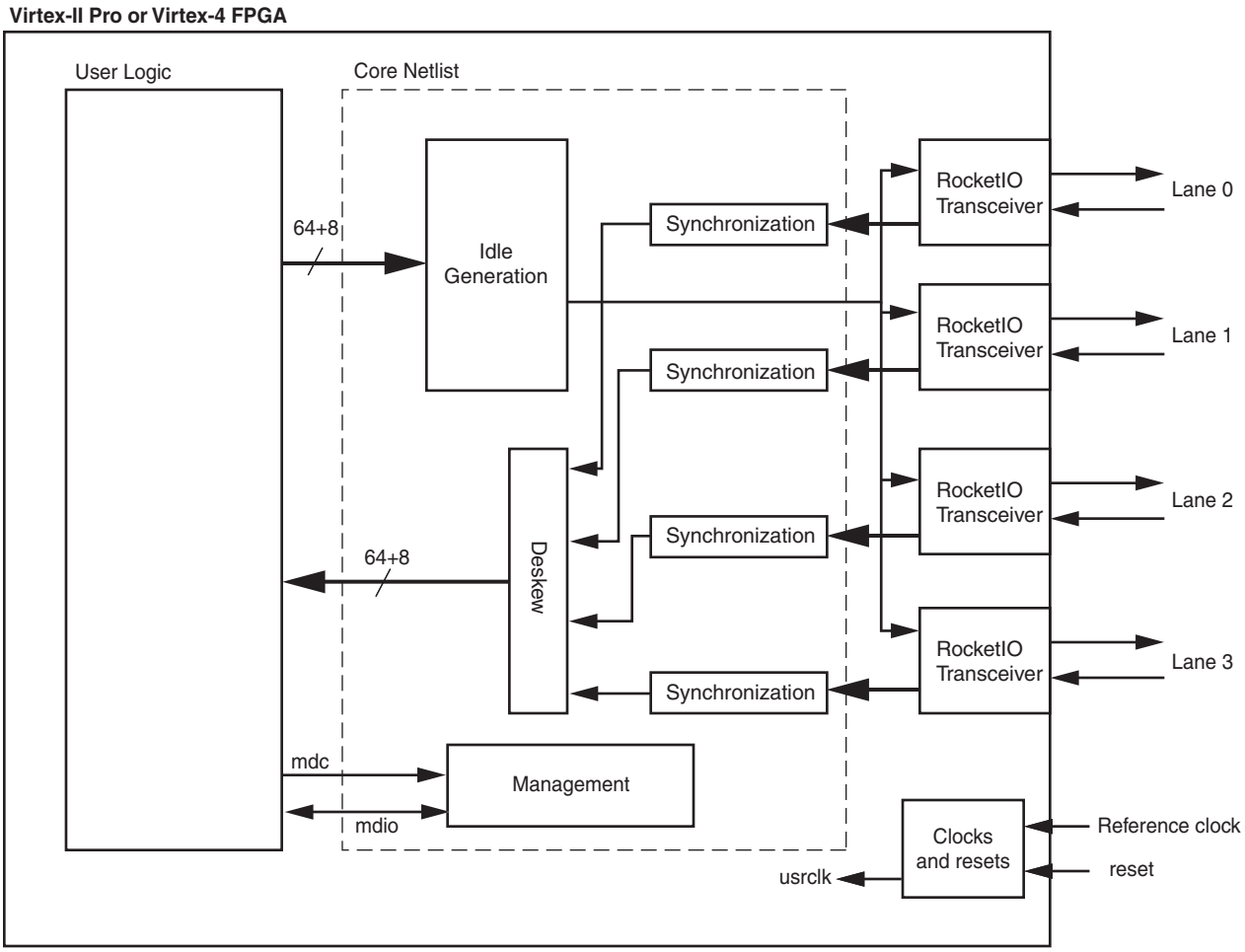

Figure 3-4: **Architecture of the XAUI Core with Client-side User Logic**

Core Interfaces and Modules

Client-side Interface

The signals of the client-side interface are shown in Table 3-1. See Chapter 6, "Interfacing to the Core" for more information on connecting to the client-side interface.

Table 3-1: **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.

Table 3-1: **Client-side Interface Ports *(Continued)***

Signal Name	Direction	Description
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.

RocketIO Interface and Module

The interface to the RocketIO transceivers is a simple pin-to-pin interface on those pins that need to be connected. The signals are described in Table 3-2. See Chapter 6, "Interfacing to the Core" for more information on connecting RocketIO transceivers to the XAUI core.

Table 3-2: **RocketIO Interface Ports**

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

MDIO Interface

The MDIO Interface signals are shown in Table 3-3. More information on using this interface can be found in Chapter 6, "Interfacing to the Core."

Table 3-3: **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 tristate. "1" disconnects the output driver from the MDIO bus.
TYPE_SEL[1:0]	IN	Type select.
PRTAD[4:0]	IN	MDIO port address.

Configuration and Status Signals

The Configuration and Status Signals are shown in Table 3-4. See "Configuration and Status Interfaces," page 50 for more information on these signals, including a breakdown of the configuration and status vectors.

Table 3-4: **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 may include Digital Clock Managers (DCMs), reset synchronizers, or other useful utility circuits that may be useful in your particular application.

Table 3-5 shows the ports on the netlist that are associated with system clocks and resets.

Table 3-5: **Clock and Reset Ports**

Signal Name	Direction	Description
USRCLK	IN	System clock for core; must also be used to clock RocketIO fabric ports.

Table 3-5: **Clock and Reset Ports *(Continued)***

Signal Name	Direction	Description
RESET	IN	Reset port synchronous to USRCLK
SOFT_RESET	OUT	Reset signal controlled by MDIO register bit.

Chapter 4

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 the user's requirements then generate the core netlist.

GUI Interface

Figure 4-1 displays the main screen for customizing the XAUI core.

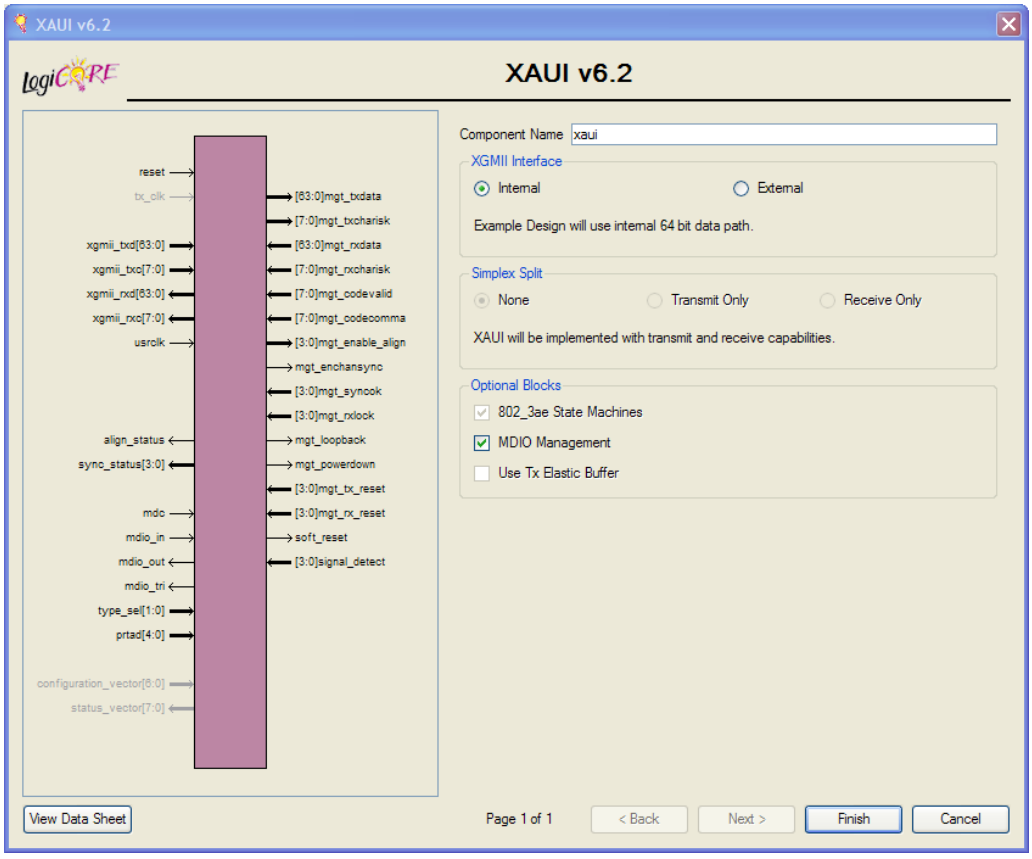

Figure 4-1: **XAUI Main Screen**

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

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.

802_3ae State Machines

This control whether the receive synchronization and alignment state machines are implemented as full *IEEE 802.3ae-2002* state machines in the fabric of the FPGA, or using the simplified state machines implemented inside the RocketIO transceivers.

The default is to implement the *IEEE 802.3ae-2002* 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 7 for more information on the use of the transmit elastic buffer.

The default is to omit the transmit elastic buffer.

Simplex Split

It is possible to generate cores that implement transmit-only and receive-only functions as well as the regular full-duplex implementation. This is controlled with the Simplex Split option.

The default is to have full-duplex transmit and receive operation.

Parameter Values in the XCO File

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

Table 4-1 shows the XCO file parameters and values, and summarizes the GUI defaults. The following is an example extract from an XCO file:

```
SELECT XAUI Virtex2P Xilinx,_Inc. 6.2
CSET component_name = the_core
```

```
CSET 802_3ae_state_machines = true
CSET simplex_split = None
CSET mdio_management = true
CSET use_tx_elastic_buffer = false
CSET xgmii_interface = internal
GENERATE
```

Table 4-1: **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
802_3ae_state_machines	True, false	True
simplex_split	None, transmit_only, receive_only	None
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 HDL example design
- Scripts to synthesize, implement and simulate the example design.

See the *LogiCORE XAUI Getting Started Guide* for a complete description of the CORE Generator output files and for details of the HDL example design.

Chapter 5

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 6, "Interfacing to the Core," which describes specific core interfaces.

General Design Guidelines

This section 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 CORE Generator 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 the user's own design or can be used to sanity-check the user application in the event of difficulty.

See the *XAUI Getting Started Guide* 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 the user 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 the user application and the core. This means that all inputs and outputs from the user application should come from, or connect to a flip-flop. While registering signals may not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx tools to place and route the design.

Recognize Timing Critical Signals

The UCF provided with the example design for the core identifies the critical signals and the timing constraints that should be applied. See Chapter 7, "Constraining the Core" for further information.

Use Supported Design Flows

The core is synthesized in the CORE Generator and is delivered to the user as an NGC netlist. The example implementation scripts provided currently use XST as the synthesis tool for the HDL example design that is delivered with the core. Other synthesis tools may be used for the user application logic: the core will always be unknown to the synthesis tool and should appear as a black box.

Post synthesis, only ISE 8.2i tools are supported.

Make Only Allowed Modifications

The XAUI Core is not user-modifiable. Do not make modifications as they may have adverse effects on system timing and protocol compliance. Supported user configurations of the XAUI Core can only be made by the selecting the options from within the CORE Generator tool when the core is generated. See Chapter 4, "Customizing and Generating the Core."

Chapter 6

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 below, the 32-bit DDR interface is functionally identical to the XGMII interface and is therefore easier to relate to the *IEEE Std. 802.3ae-2002* specification.

Virtex-II Pro

The interface uses the Virtex-II Pro SelectIO™ DDR input and output registers to translate into the DDR domain. The mapping of lanes to data bits is shown in Table 6-1.

Table 6-1: **XGMII_TXD, XGMII_RXD Lanes for External 32-bit DDR Interface**

Lane	XGMII_TXD, XGMII_RXD Bits
0	7:0
1	15:8
2	23:16
3	31:24

On the transmit side of the core, the DDR registers 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. However, the upper half of the bus is clocked out into the fabric on the falling edge of the clock and this falling-edge clocking is maintained into the core netlist; in other words, when the external interface is selected, the netlist port width of,

say, the data port is still 64 bits but the upper 32 bits are falling-edge domain. This is illustrated in the schematic of Figure 6-1 and the timing diagram of Figure 6-2.

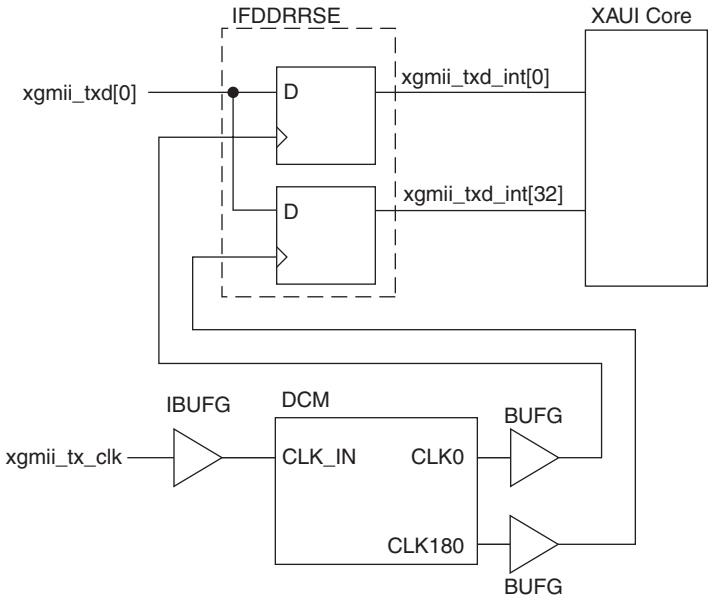

Figure 6-1: **Schematic of Inbound DDR Interface: Virtex-II Pro**

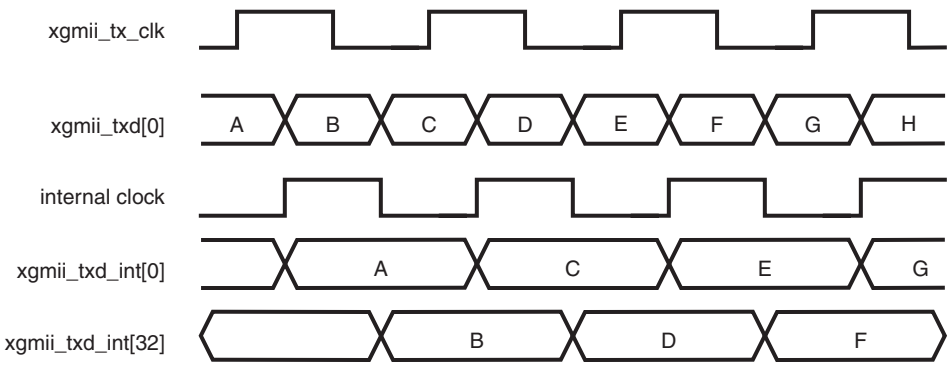

Figure 6-2: **Timing Diagram of Operation of Inbound DDR Interface: Virtex-II Pro**

On the outbound receive side of the core, a similar situation prevails; the upper half of the bus is clocked out of the core netlist on the falling edge clock, ready to be taken straight into the DDR output registers with no further handling by the user.

Virtex-4

XGMII on Virtex-4 uses the IDDR and ODDR primitives to convert from single data rate to double data rate and back again.

The mapping of lanes to data bits in Virtex-4 is identical to that in Virtex-II Pro; see Table 6-1.

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 6-3 and a timing diagram is shown in Figure 6-4. 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 *Virtex-4 User Guide*.

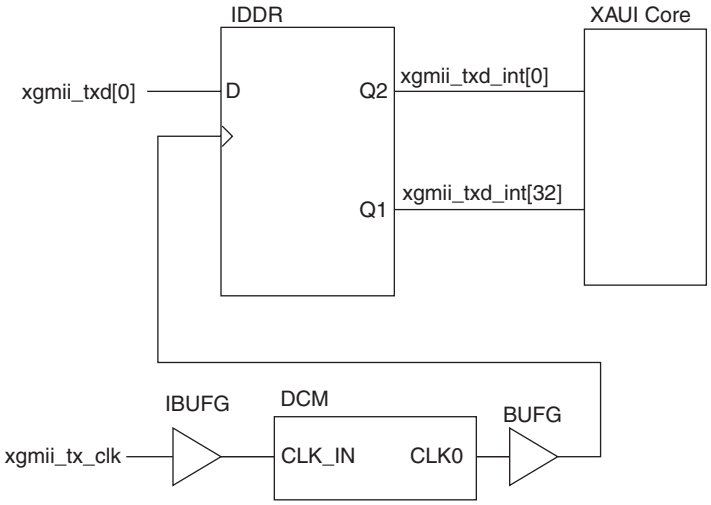

Figure 6-3: **Schematic of Inbound DDR Interface: Virtex-4**

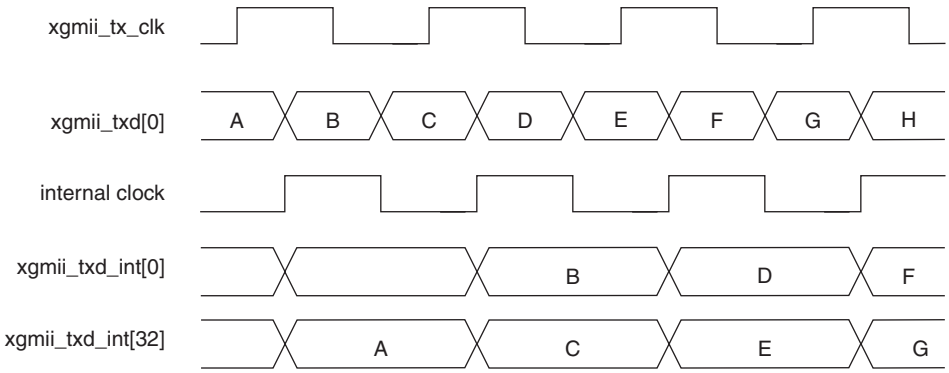

Figure 6-4: **Timing Diagram of Operation of Inbound DDR Interface: Virtex-4**

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 above, the key difference being 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 6-2. The lane number is also the index of the control bit for that particular lane; e.g., XGMII_TXC[2] and XGMII_TXD[23:16] are the control and data bits respectively for lane 2.

Table 6-2: **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

In the following discussion, reference is regularly made to certain XGMII control characters signifying Start, Terminate, Error, etc. 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. 802.3ae-2002* and are reproduced in Table 6-3 for reference.

Table 6-3: **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 6-5. 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 being padded by XGMII Idle characters.

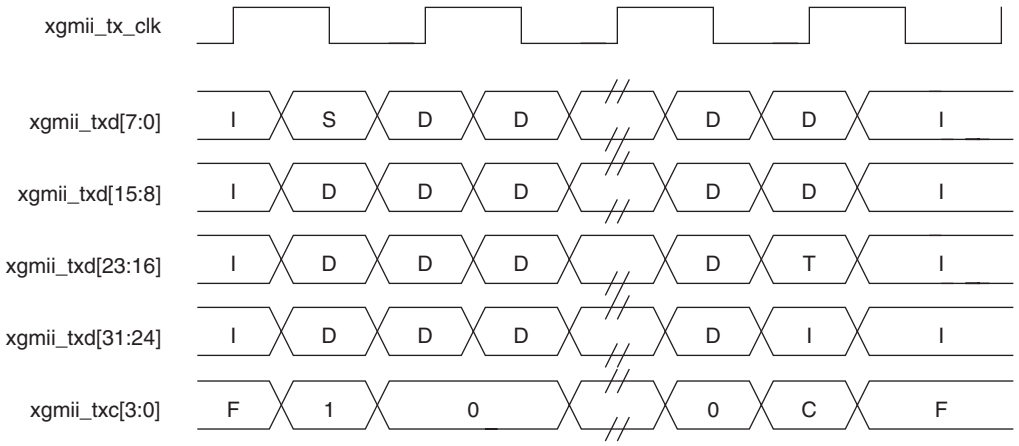

Figure 6-5: **Frame Transmission Across the 32-bit XGMII Interface**

The timing of a data frame transmission containing an error via the external XGMII interface is shown in Figure 6-6. The presence of the error is denoted by the letter E in lanes 0, 1, 2, and 3.

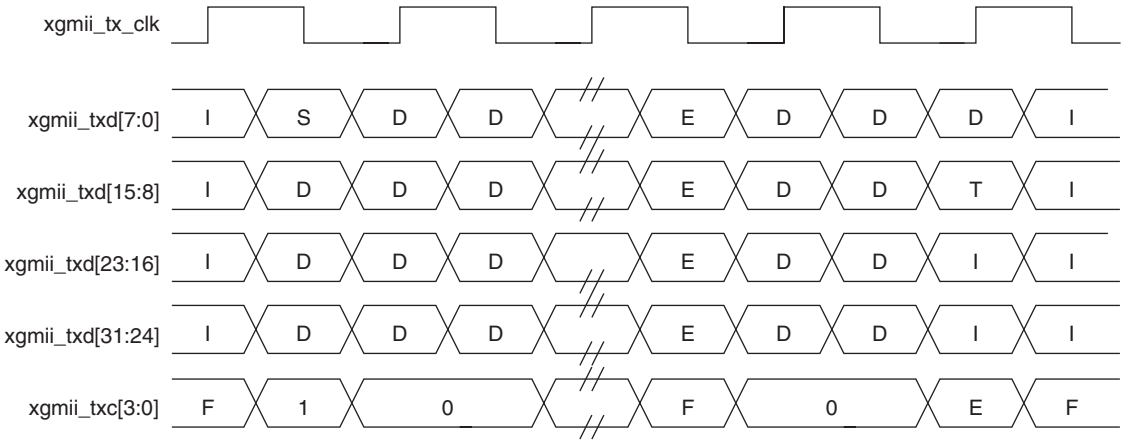

Figure 6-6: **Frame Transmission with Errors Across 32-bit XGMII Interface**

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 6-7. The beginning of the data frame is shown by the presence of the Start character (the /S/ codegroup in lane 4 of Figure 6-7), 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.

Once the frame is complete, the frame is completed by a Terminate character (the T in lane 1 of Figure 6-7). The terminate character can occur in any lane, the remaining lanes being padded by XGMII idle characters.

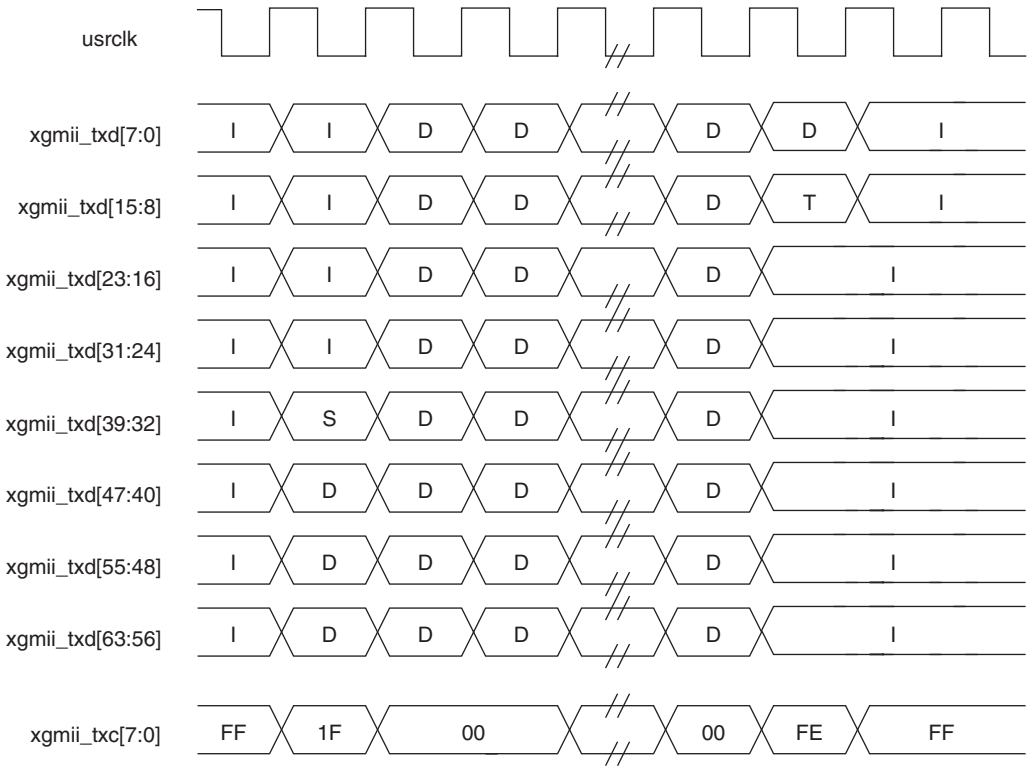

Figure 6-7: **Normal Frame Transmission Across the Internal 64-bit Client-side I/F**

Figure 6-8 depicts a similar frame to that in Figure 6-7, 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.

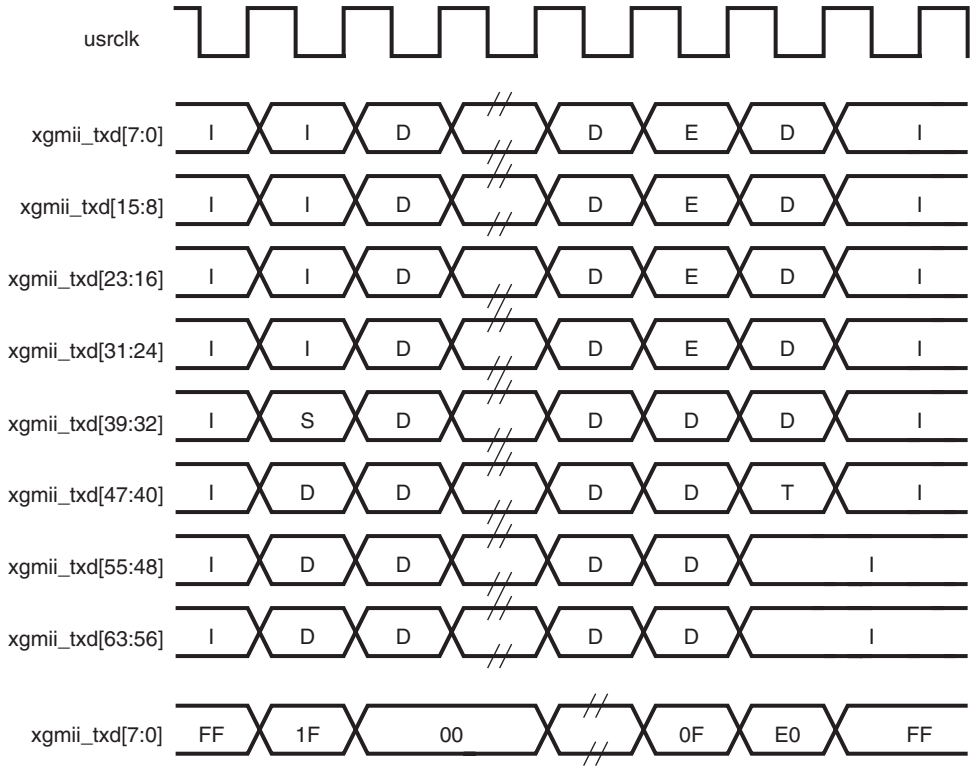

Figure 6-8: **Frame Transmission with Error Across Internal 64-bit Client-side I/F**

Interfacing to the Receive Client Interface

External 32-bit DDR Client-side Interface

Figure 6-9 shows an 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.

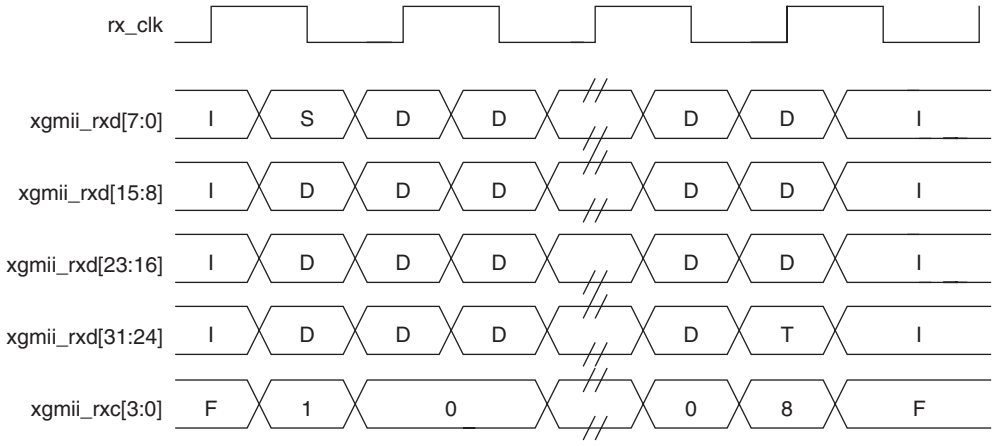

Figure 6-9: **Frame Reception Across External 32-bit XGMII Interface**

Figure 6-10 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.

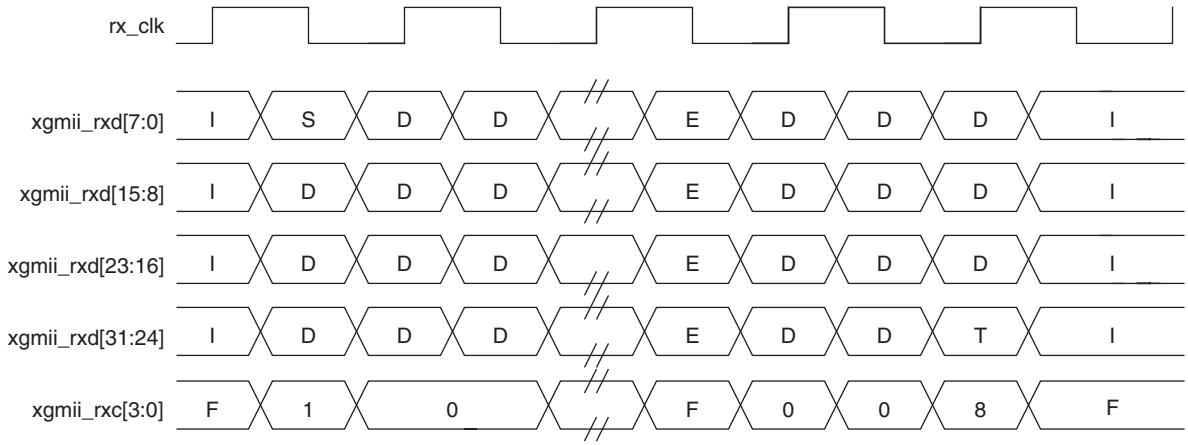

Figure 6-10: **Frame Reception with Error Across External 32-bit XGMII Interface**

Internal 64-bit Client-side Interface

The timing of a normal inbound frame transfer is shown in Figure 6-11. As in the transmit case, the frame is delimited by a Start character (S) and by a Terminate character (T). Note

that 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.

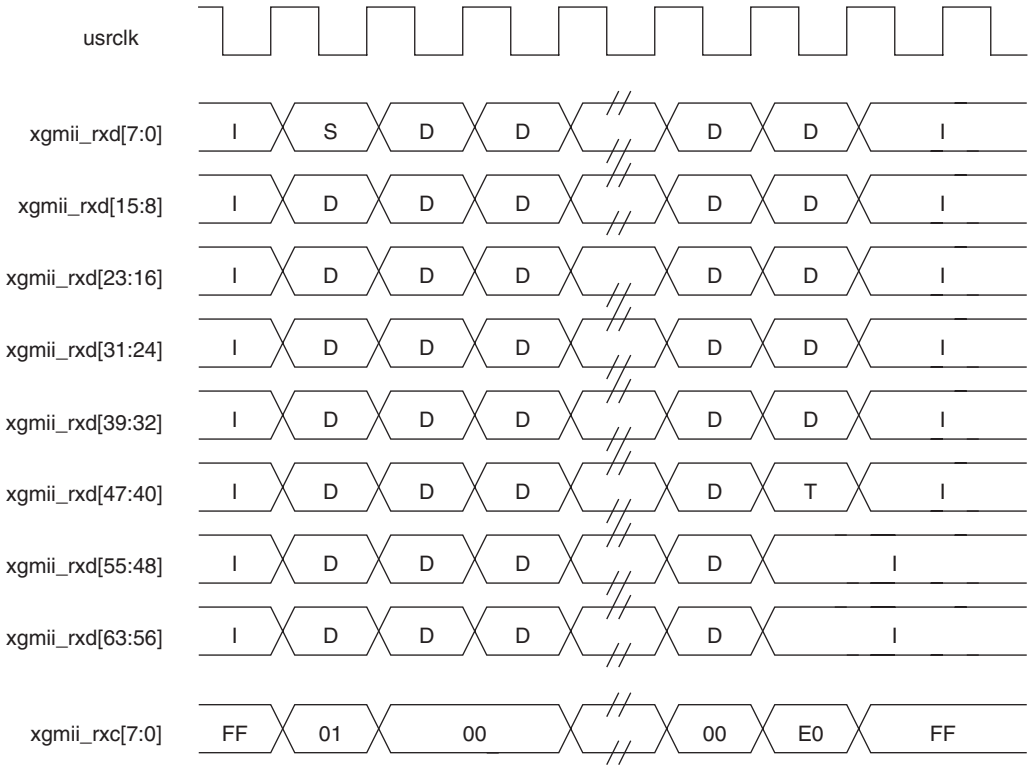

Figure 6-11: **Frame Reception Across the Internal 64-bit Client Interface**

Figure 6-12 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.

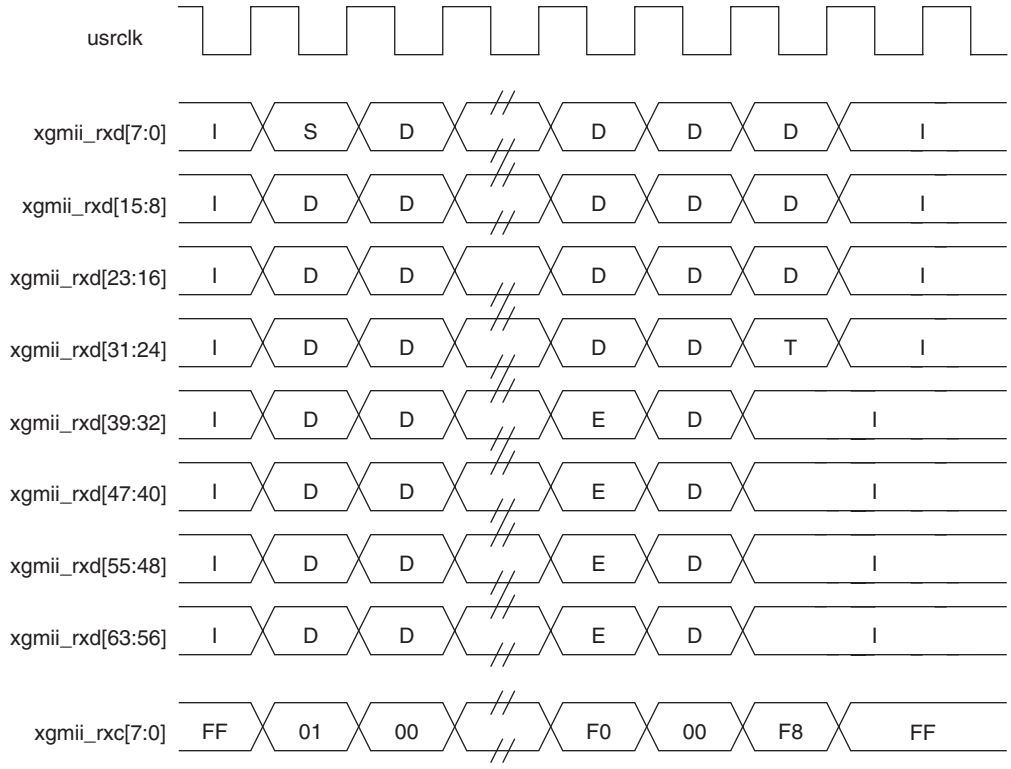

Figure 6-12: **Frame Reception with Error Across the Internal 64-bit Client Interface**

Interfacing to the RocketIO Transceivers

Virtex-II Pro

Table 6-4 shows the ports of the netlist that are to be connected to the Virtex-II Pro RocketIO transceivers. The remainder of the RocketIO ports are not connected to the netlist but are connected in the core source code (`transceiver.vhd` or `transceiver.v`) or are wired to static values.

Table 6-4: **RocketIO Interface Ports - Virtex-II Pro**

Signal Name	Direction	Description
MGT_TXDATA[63:0]	OUT	RocketIO transmit data
MGT_TXCHARISK[7:0]	OUT	RocketIO transmit control flags
MGT_RXDATA[63:0]	IN	RocketIO receive data
MGT_RXCHARISK[7:0]	IN	RocketIO receive control signals
MGT_CODEVALID[7:0]	OUT	RocketIO receive control signals
MGT_CODECOMMA[7:0]	IN	RocketIO receive control signals
MGT_ENABLE_ALIGN[3:0]	OUT	RocketIO control signals

Table 6-4: **RocketIO Interface Ports - Virtex-II Pro *(Continued)***

Signal Name	Direction	Description
MGT_ENCHANSYNC	OUT	RocketIO control signal
MGT_SYNCOK[3:0]	IN	RocketIO control signal
MGT_LOOPBACK	OUT	RocketIO control signal
MGT_POWERDOWN	OUT	RocketIO control signal
SIGNAL_DETECT[3:0]	IN	Status signal from attached optical module

Also contained in the transceiver HDL files are synchronizing registers for the comma alignment ports ENPCOMMAALIGN, ENMCOMMAALIGN. It is important that these registers are correctly placed with respect to the RocketIO they are associated with; see "RocketIO Placement," on page 91 for instructions on modifying this constraint for your application.

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 4-wire bus should be tied to "1111."

No timing diagrams are presented here for the RocketIO signals; this interface should be treated as a black box by the user. If customization of this interface is required, please see the *RocketIO User Guide* for detailed descriptions of the transceiver ports.

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 will be available in a particular core instance. The interfaces are:

- "The MDIO Interface," on page 51
- "Configuration and Status Vectors," on page 86

In addition, there are output ports on the core signalling alignment and synchronization status. These ports are described in "Alignment and Synchronization Status Ports," on page 88.

Virtex-4

Table 6-5 shows the ports of the netlist that are to be connected to the Virtex-4 RocketIO™ transceivers. The remainder of the RocketIO ports are not connected to the netlist but are connected in the core source code (`transceivers.vhd` or `transceivers.v`) or are wired to static values.

Table 6-5: **RocketIO Interface Ports - Virtex-4**

Signal Name	Direction	Description
MGT_TXDATA[63:0]	OUT	RocketIO transmit data
MGT_TXCHARISK[7:0]	OUT	RocketIO transmit control flags
MGT_RXDATA[63:0]	IN	RocketIO receive data
MGT_RXCHARISK[7:0]	IN	RocketIO receive control signals
MGT_CODEVALID[7:0]	IN	RocketIO receive control signals

Table 6-5: **RocketIO Interface Ports - Virtex-4 *(Continued)***

Signal Name	Direction	Description
MGT_CODECOMMA[7:0]	Input	RocketIO receive control signals
MGT_ENABLE_ALIGN[3:0]	Output	RocketIO control signals
MGT_ENCHANSYNC	Output	RocketIO control signal
MGT_RXLOCK[3:0]	Input	RocketIO control signal
MGT_LOOPBACK	Output	RocketIO control signal
MGT_POWERDOWN	Output	RocketIO control signal
SIGNAL_DETECT[3:0]	Input	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 4-wire bus should be tied to "1111."

No timing diagrams are presented here for the RocketIO signals; this interface should be treated as a black box by the user. If customization of this interface is required, please see the *RocketIO User Guide* for detailed descriptions of the transceiver ports.

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 will be available in a particular core instance. The interfaces are:

- The MDIO Interface below
- "Configuration and Status Vectors," on page 86

In addition, there are output ports on the core signalling alignment and synchronization status. These ports are described in "Alignment and Synchronization Status Ports," on page 88.

The Virtex-4 FX RocketIO transceivers require a Calibration Block to be included in the fabric logic. (See the Calibration Block User Guide for additional 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.

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:

- The MDIO Interface below
- "Configuration and Status Vectors," on page 86

In addition, there are output ports on the core signalling alignment and synchronization status. These ports are described in "Alignment and Synchronization Status Ports," on page 88.

The MDIO Interface

The Management Data Input/Output (MDIO) Interface is a simple low-speed 2-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 802.3ae-2002*.

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 6-13 illustrates a typical system. All transactions are initiated by the STA entity. The XAUI core implements an MMD.

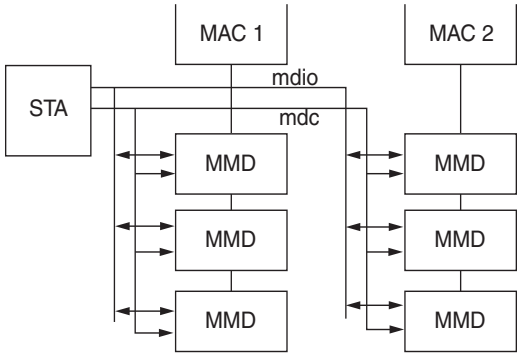

Figure 6-13: **A Typical MDIO-managed System**

MDIO Ports

The core ports associated with MDIO are shown in Table 6-6.

Table 6-6: **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 tristate. ”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 tri-state buffer either in the FPGA SelectIO buffer or in a separate

device. Figure 6-14 illustrates the use of a Virtex-II Pro or Virtex-4 SelectIO tri-state buffer as the bus interface.

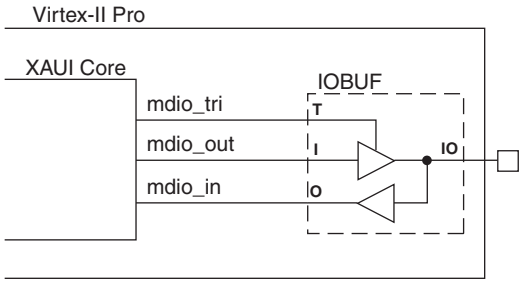

Figure 6-14: **Using a SelectIO Tri-state Buffer to Drive MDIO**

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 6-7 shows the mapping of the type_sel setting to the implemented register map.

Table 6-7: **Mapping of type_sel Port Settings to MDIO Register Type**

Signal Setting	Implemented MDIO Registers
"00" or "01"	1000BASE-X PCS/PMA
"10"	DTE XGXS
"11"	PHY XGXS

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 the prtad[4:0] to a unique value for each instance; the XAUI core will ignore 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. 802.3ae-2002*. 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 6-15 shows an Address transaction; this is defined by OP="00." This is used to set the internal 16-bit address register of the XAUI core for subsequent data transactions. This is called the "current address" in the following sections.

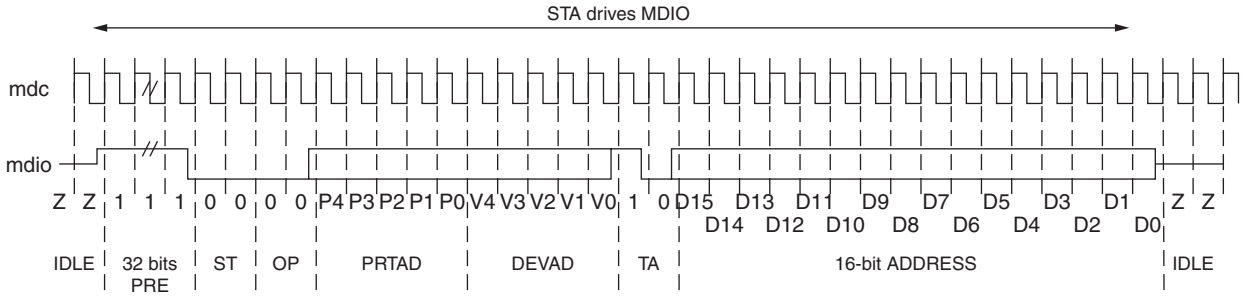

Figure 6-15: **MDIO Set Address Transaction**

Write Transaction

Figure 6-16 shows a Write transaction; this is 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.

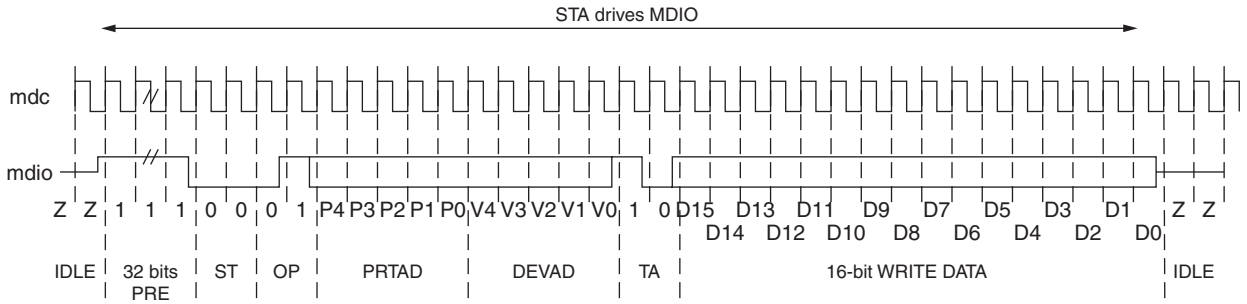

Figure 6-16: **MDIO Write Transaction**

Read Transaction

Figure 6-17 shows a Read transaction; this is defined by OP="11." The XAUI core returns the 16-bit word from the register at the current address.

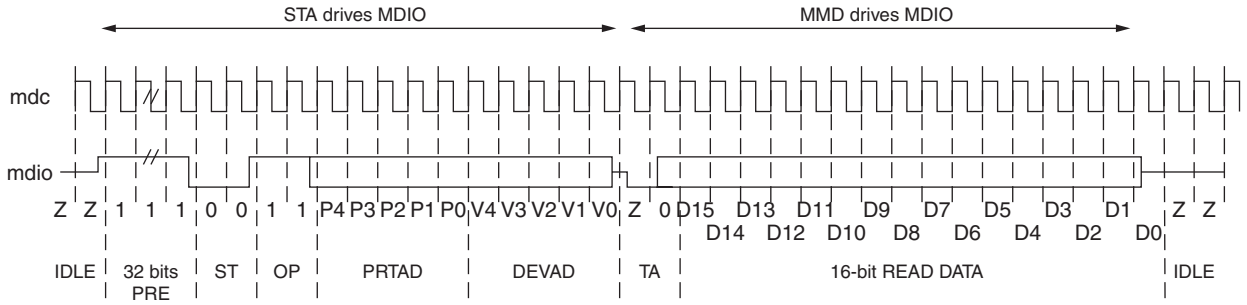

Figure 6-17: **MDIO Read Transaction**

Post-Read-increment-address Transaction

Figure 6-18 shows a Post-read-increment-address transaction; this is 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.

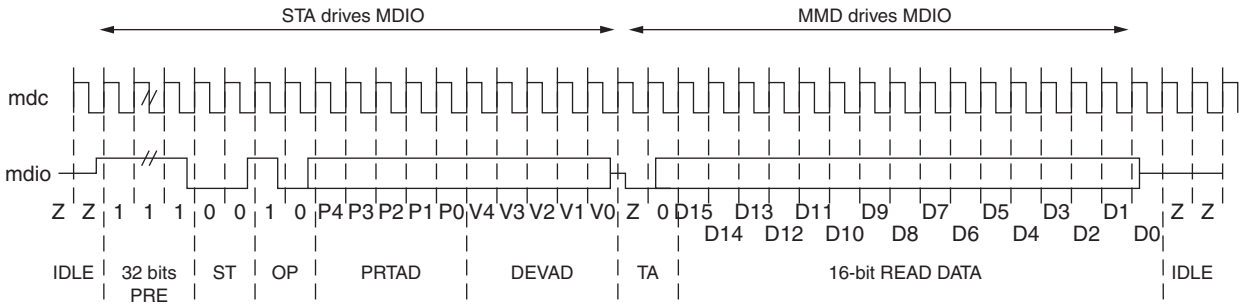

Figure 6-18: **MDIO Read-and-increment Transaction**

10GBASE-X PCS/PMA Register Map

When the core is configured as a 10GBASE-X PCS/PMA, it occupies MDIO Device Addresses 1 and 3 in the MDIO register address map shown in Table 6-8.

Table 6-8: **10GBASE-X PCS/PMA MDIO Registers**

Register Address	Register Name
1.0	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

Table 6-8: **10GBASE-X PCS/PMA MDIO Registers *(Continued)***

Register Address	Register Name
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 6-19 shows the MDIO Register 1.0: PMA/PMD Control 1.

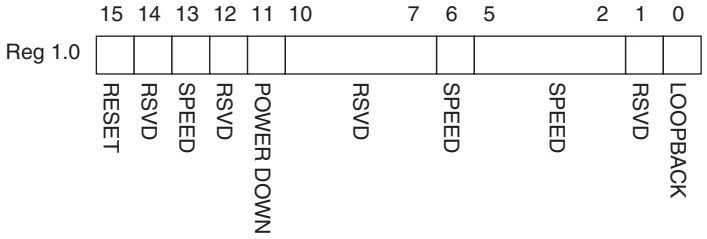

Figure 6-19: **PMA/PMD Control 1 Register**

Table 6-9 shows the PMA Control 1 register bit definitions.

Table 6-9: **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.	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 MGTs are placed in a low power state. This bit requires a reset (see bit 1.0.15) to clear.	R/W	0
1.0.10:7	Reserved	The block always returns “0” for these bits and ignores writes.	R/O	All “0s”

Table 6-9: **PMA/PMD Control 1 Register Bit Definitions *(Continued)***

Bit(s)	Name	Description	Attributes	Default Value
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”
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 will loop the signal in the MGTs back into the receiver.	R/W	0

MDIO Register 1.1: PMA/PMD Status 1

Figure 6-20 shows the MDIO Register 1.1: PMA/PMD Status 1.

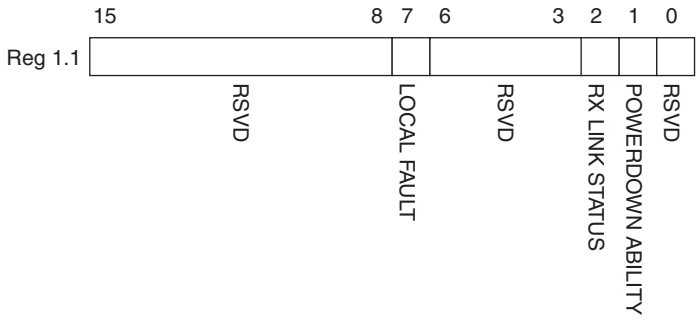

Figure 6-20: **PMA/PMD Status 1 Register**

Table 6-10 shows the PMA/PMD Status 1 register bit definitions.

Table 6-10: **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	1 = Local fault detected 0 = No local fault detected This bit is set to “1” whenever either of the bits 1.8.11, 1.8.10 are set to “1.”	R/O	-
1.1.6:3	Reserved	The block always returns “0” for this bit.	R/O	0

Table 6-10: **PMA/PMD Status 1 Register Bit Definitions *(Continued)***

Bit(s)	Name	Description	Attributes	Default Value
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 6-21 shows the MDIO Registers 1.2 and 1.3: PMA/PMD Device Identifier.

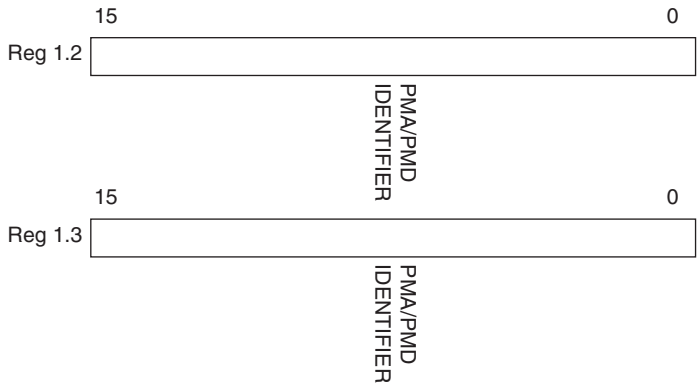

Figure 6-21: **PMA/PMD Device Identifier Registers**

Table 6-11 shows the PMA/PMD Device Identifier registers bit definitions.

Table 6-11: **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 6-22 shows the MDIO Register 1.4: PMA/PMD Speed Ability.

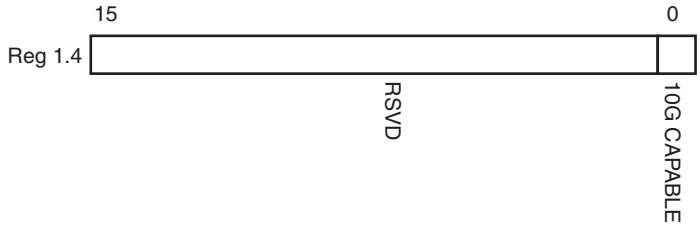

Figure 6-22: **PMA/PMD Speed Ability Register**

Table 6-12 shows the PMA/PMD Speed Ability register bit definitions.

Table 6-12: **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 6-23 shows the MDIO Registers 1.5 and 1.6: PMA/PMD Devices in Package.

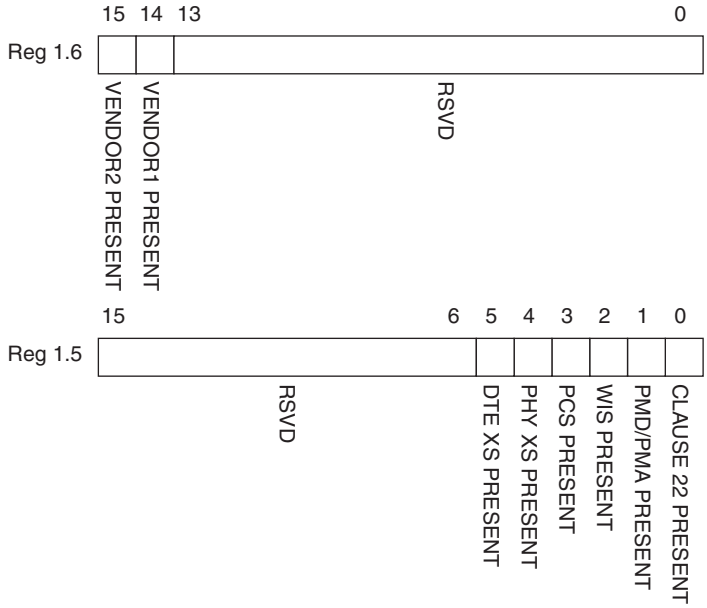

Figure 6-23: **PMA/PMD Devices in Package Registers**

Table 6-13 shows the PMA/PMD Device in Package registers bit definitions.

Table 6-13: **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 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 6-24 shows the MDIO Register 1.7: 10G PMA/PMD Control 2.

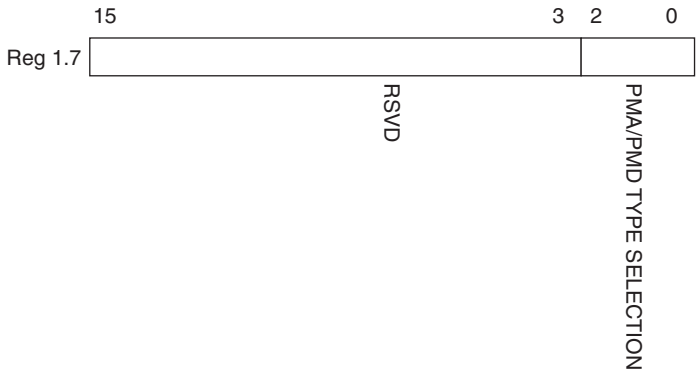

Figure 6-24: **10G PMA/PMD Control 2 Register**

Table 6-14 shows the PMA/PMD Control 2 register bit definitions.

Table 6-14: **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 6-25 shows the MDIO Register 1.8: 10G PMA/PMD Status 2.

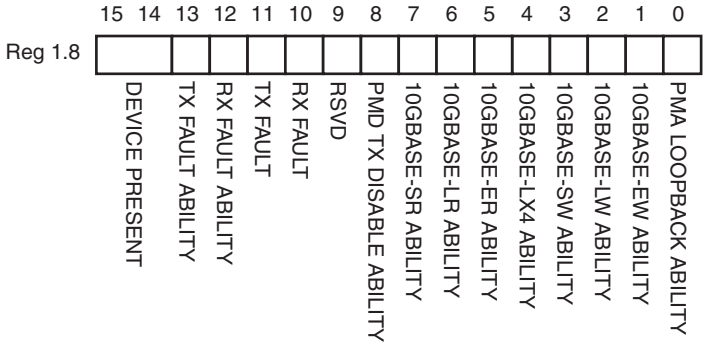

Figure 6-25: **10G PMA/PMD Status 2 Register**

Table 6-15 shows the PMA/PMD Status 2 register bit definitions.

Table 6-15: **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	1 = Fault condition on transmit path; 0 = No fault condition on transmit path.	R/O Latching High	0
1.8.10	Receive Fault	1 = Fault condition on receive path; 0 = No fault condition on receive path.	R/O Latching High	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 6-15: **10G PMA/PMD Status 2 Register Bit Definitions *(Continued)***

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 6-26 shows the MDIO 1.10 Register: 10G PMD Signal Receive OK.

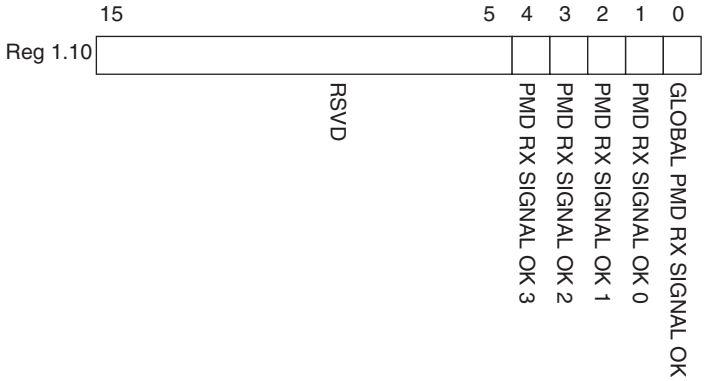

Figure 6-26: **10G PMD Signal Receive OK Register**

Table 6-16 shows the 10G PMD Signal Receive OK register bit definitions.

Table 6-16: **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 6-27 shows the MDIO Registers 1.14 and 1.15: PMA/PMD Package Identifier register.

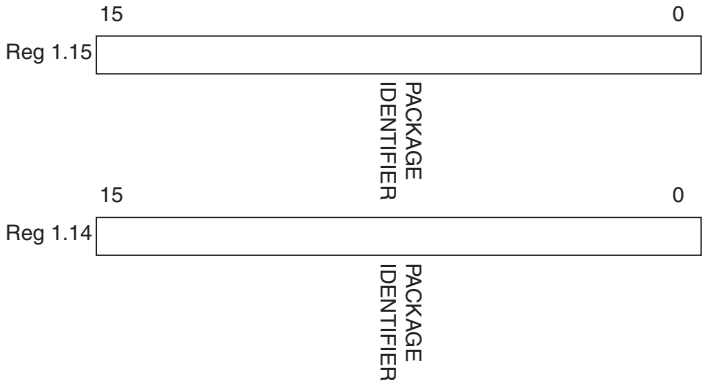

Figure 6-27: **PMA/PMD Package Identifier Registers**

Table 6-17 shows the PMA/PMD Package Identifier registers bit definitions.

Table 6-17: **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 6-28 shows the MDIO Register 3.0: PCS Control 1.

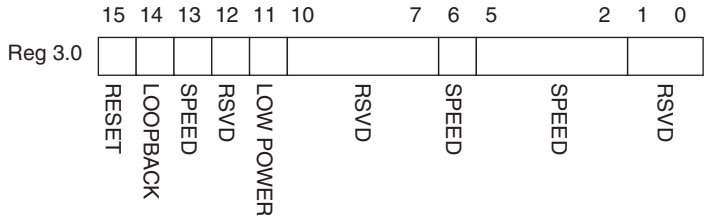

Figure 6-28: **PCS Control 1 Register**

Table 6-18 shows the PCS Control 1 register bit definitions.

Table 6-18: **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 MGTs are placed in a low power state. This bit requires a reset (see bit 3.0.15) to clear.	R/W	0
3.0.10:7	Reserved	The block always returns “0” for these bits and ignores writes.	R/O	All “0s”

Table 6-18: **PCS Control 1 Register Bit Definitions *(Continued)***

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 6-29 shows the MDIO Register 3.1: PCS Status 1.

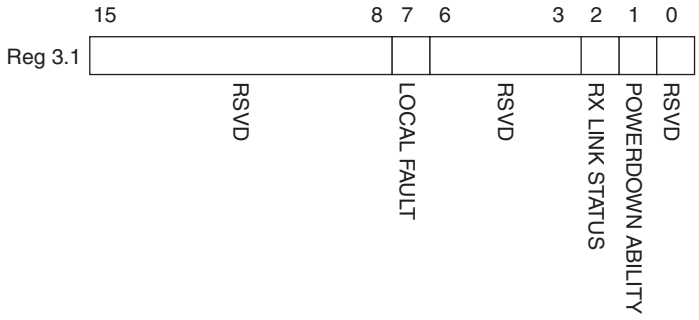

Figure 6-29: **PCS Status 1 Register**

Table 6-19 show the PCS 1 register bit definitions.

Table 6-19: **PCS Status 1 Register Bit Definition**

Bit(s)	Name	Description	Attributes	Default Value
3.1.15:8	Reserved	The block always returns “0”s 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”

Table 6-19: **PCS Status 1 Register Bit Definition *(Continued)***

Bit(s)	Name	Description	Attributes	Default Value
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	-
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 6-30 shows the MIDO Registers 3.2 and 3.3: PCS Device Identifier

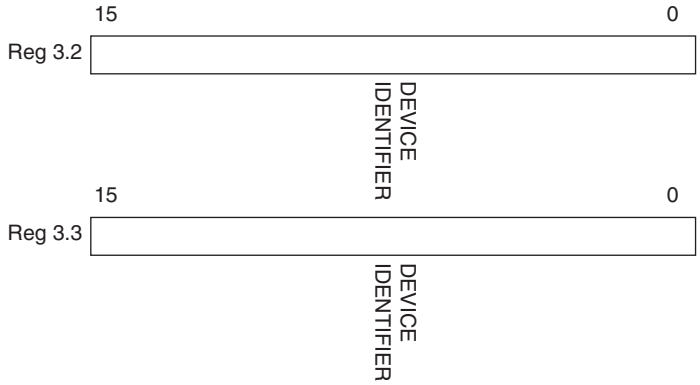

Figure 6-30: **PCS Device Identifier Registers**

Table 6-20 shows the PCS Device Identifier registers bit definitions.

Table 6-20: **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 6-31 shows the MDIO Register 3.4: PCS Speed Ability.

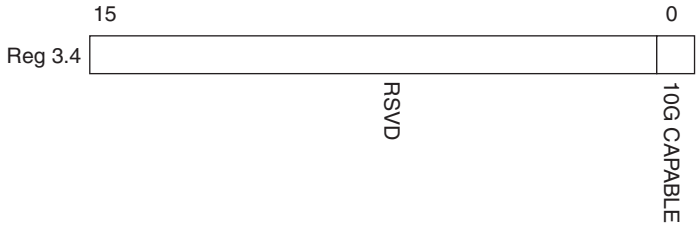

Figure 6-31: **PCS Speed Ability Register**

Table 6-21 shows the PCS Speed Ability register bit definitions.

Table 6-21: **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	10G 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 6-32 shows the MDIO Registers 3.5 and 3.6: PCS Devices in Package.

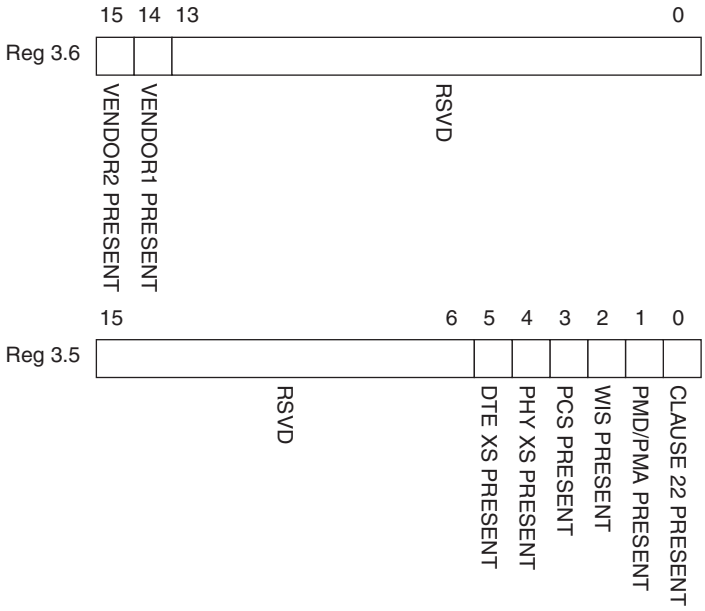

Figure 6-32: **PCS Devices in Package Registers**

Table 6-22 shows the PCS Devices in Package registers bit definitions.

Table 6-22: **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 6-33 shows the MDIO Register 3.7: 10G PCS Control 2.

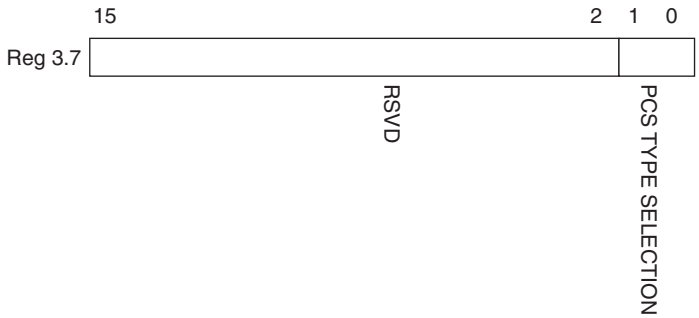

Figure 6-33: **10G PCS Control 2 Register**

Table 6-23 shows the 10 G PCS Control 2 register bit definitions.

Table 6-23: **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 6-34 shows the MDIO Register 3.8: 10G PCS Status 2.

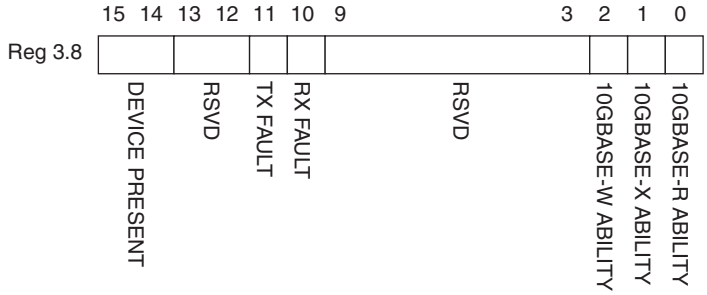

Figure 6-34: **10G PCS Status 2 Register**

Table 6-24 shows the 10G PCS Status 2 register bit definitions.

Table 6-24: **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 6-35 shows the MDIO Registers 3.14 and 3.15: PCS Package Identifier.

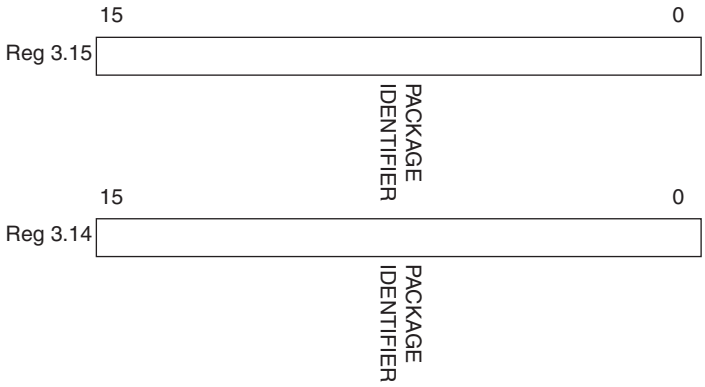

Figure 6-35: **Package Identifier Registers**

Table 6-25 shows the PCS Package Identifier registers bit definitions.

Table 6-25: **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 6-36 shows the MDIO Register 3.24: 10GBase-X Status.

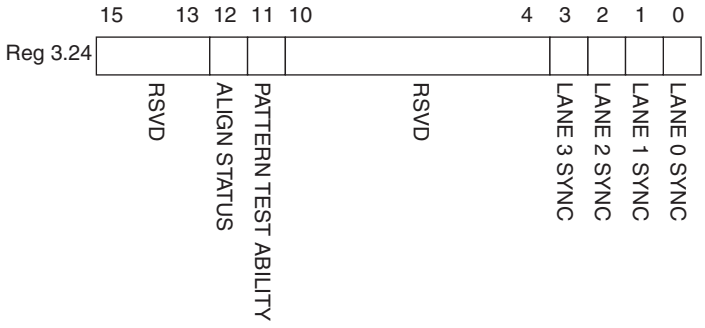

Figure 6-36: **10GBASE-X Status Register**

Table 6-26 shows the 10GBase-X Status register bit definitions.

Table 6-26: **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 6-37 shows the MDIO Register 3.25: 10GBase-X Test Control.

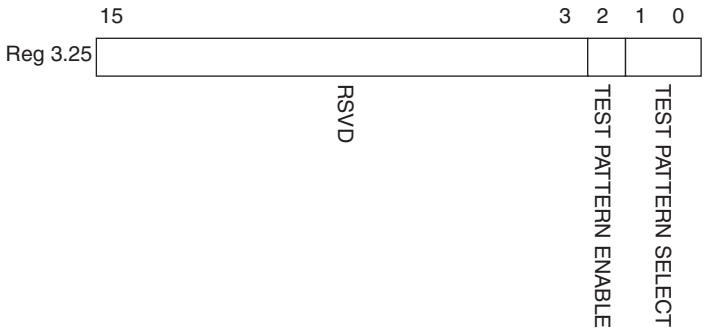

Figure 6-37: **Test Control Register**

Table 6-27 shows the 10GBase-X Test Control register bit definitions.

Table 6-27: **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 6-28).

Table 6-28: **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 6-38 shows the MDIO Register 5.0: DTE XS Control 1.

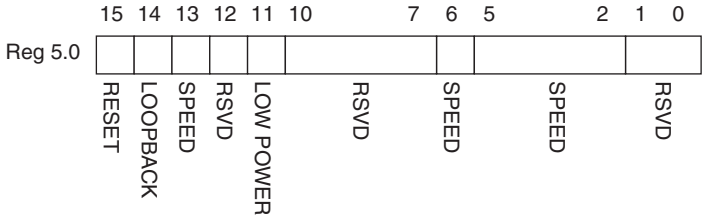

Figure 6-38: **DTE XS Control 1 Register**

Table 6-29 shows the DTE XS Control 1 register bit definitions.

Table 6-29: **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 will loop the signal in the MGTs back into the receiver.	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 MGTs are placed in a low power state. This bit requires a reset (see bit 5.0.15) to clear.	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
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 6-39 shows the MDIO Register 5.1: DTE XS Status 1.

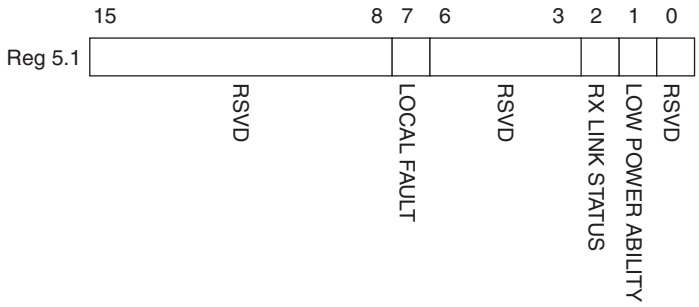

Figure 6-39: **DTE XS Status 1 Register**

Table 6-30 shows the DET XS Status 1 register bit definitions.

Table 6-30: **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 6-40 shows the MDIO Registers 5.2 and 5.3: DTE XS Device Identifier.

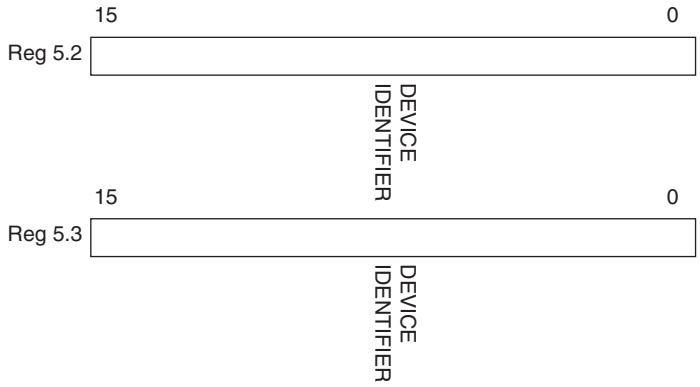

Figure 6-40: **DTE XS Device Identifier Registers**

Table 6-31 shows the DTE XS Device Identifier registers bit definitions.

Table 6-31: **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 6-41 shows the MDIO Register 5.4: DTE Speed Ability.

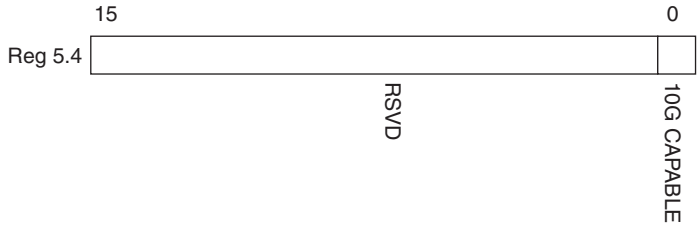

Figure 6-41: **DTE XS Speed Ability Register**

Table 6-32 shows the DTE XS Speed Ability register bit definitions.

Table 6-32: **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 6-42 shows the MDIO Registers 5.5 and 5.6: DTE XS Devices in Package.

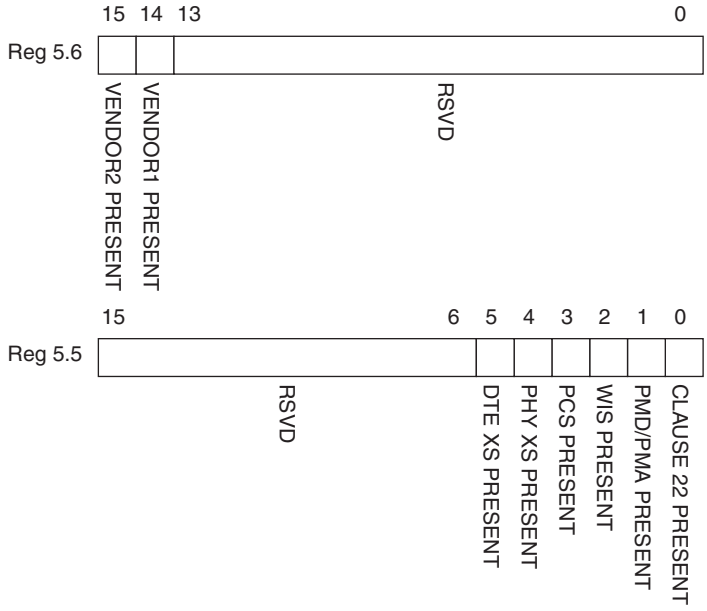

Figure 6-42: **DTE XS Devices in Package Register**

Table 6-33 shows the DTE XS Devices in Package registers bit definitions.

Table 6-33: **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	0
5.5.4	DTE 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 6-43 shows the MDIO Register 5.8: DTE XS Status 2.

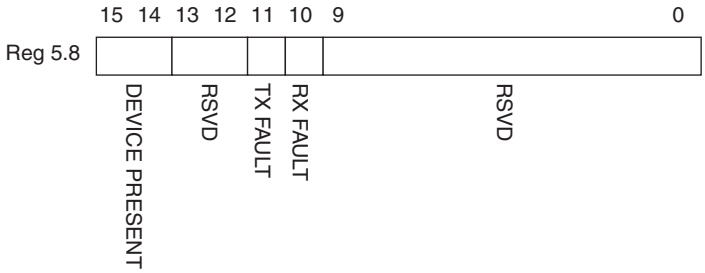

Figure 6-43: **DTE XS Status 2 Register**

Table 6-34 show the DTE XS Status 2 register bits definitions.

Table 6-34: **DTE XS Status 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.8.15:14	Device Present	The block shall always return “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 6-44 shows the MDIO Registers 5.14 and 5.15: DTE XS Package Identifier.

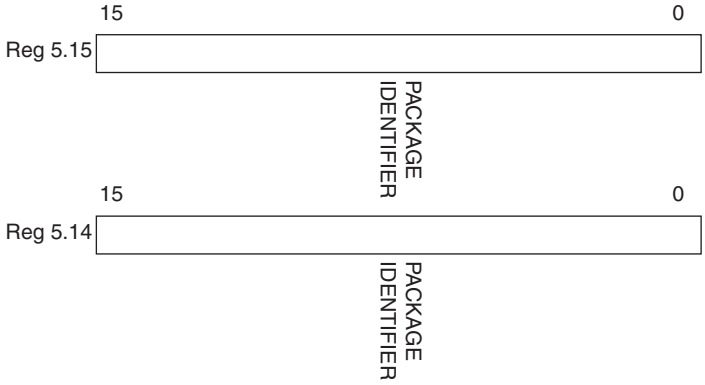

Figure 6-44: **DTE XS Package Identifier Registers**

Table 6-35 shows the DTE XS Package Identifier registers bit definitions.

Table 6-35: **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. 802.3ae-2002* 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 RocketIO output
- Low frequency test pattern of “11111000001111100000111110000....” at each RocketIO output
- mixed frequency test pattern of “11111010110000010100111110101100000101 0...” at each RocketIO output.

MDIO Register 5.24: DTE XS Lane Status

Figure 6-45 shows the MDIO Register 5.24: DTE XS Lane Status.

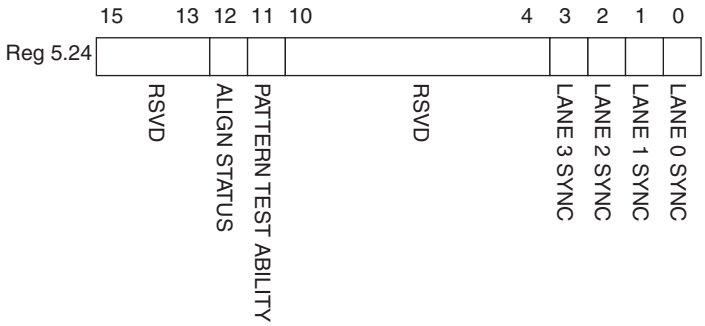

Figure 6-45: **DTE XS Lane Status Register**

Table 6-36 shows the DTE XS Lane Status register bit definitions.

Table 6-36: **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 6-46 shows the MDIO Register 5.25: 10G DTE XGXS Test Control.

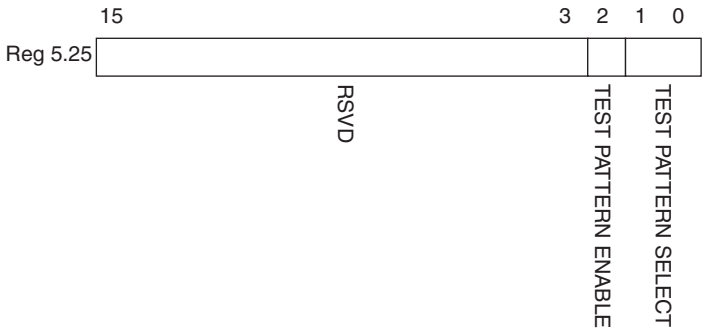

Figure 6-46: **10G DTE XGXS Test Control Register**

Table 6-37 shows the 10G DTE XGXS Test Control register bit definitions.

Table 6-37: **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 6-38).

Table 6-38: **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 6-47 shows the MDIO Register 4.0: PHY XS Control 1.

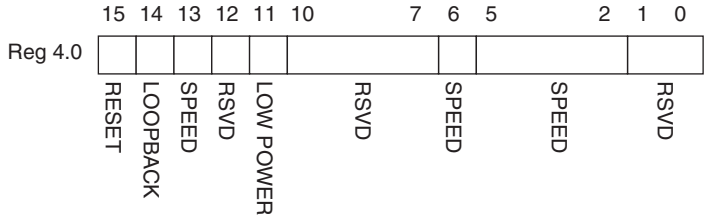

Figure 6-47: **PHY XS Control 1 Register**

Table 6-39 shows the PHY XS Control 1 register bit definitions.

Table 6-39: **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 will loop the signal in the MGTs back into the receiver.	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 MGTs are placed in a low power state. This bit requires a reset (see bit 4.0.15) to clear.	R/W	0
4.0.10:7	Reserved	The block always returns “0s” for these bits and ignores writes.	R/O	All "0s"
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 6-48 shows the MDIO Register 4.1: PHY XS Status 1.

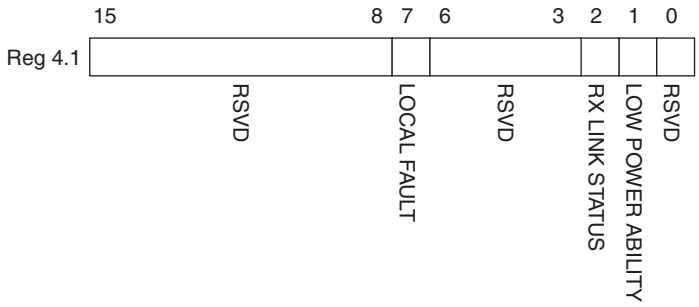

Figure 6-48: **PHY XS Status 1 Register**

Table 6-40 shows the PHY XS Status 1 register bit definitions.

Table 6-40: **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 6-49 shows the MDIO Registers 4.2 and 4.3: PHY XS Device Identifier.

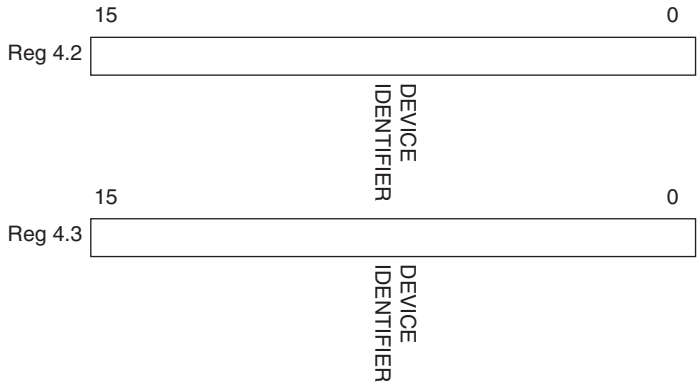

Figure 6-49: **PHY XS Device Identifier Registers**

Table 6-41 shows the PHY XS Devices Identifier registers bit definitions.

Table 6-41: **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 6-50 shows the MDIO Register 4.4: PHY XS Speed Ability.

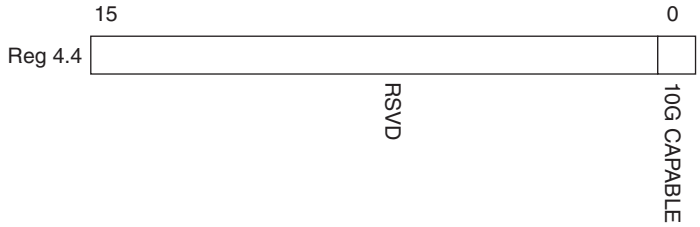

Figure 6-50: **PHY XS Speed Ability Register**

Table 6-42 shows the PHY XS Speed Ability register bit definitions.

Table 6-42: **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 6-51 shows the MDIO Registers 4.5 and 4.6: PHY XS Devices in Package.

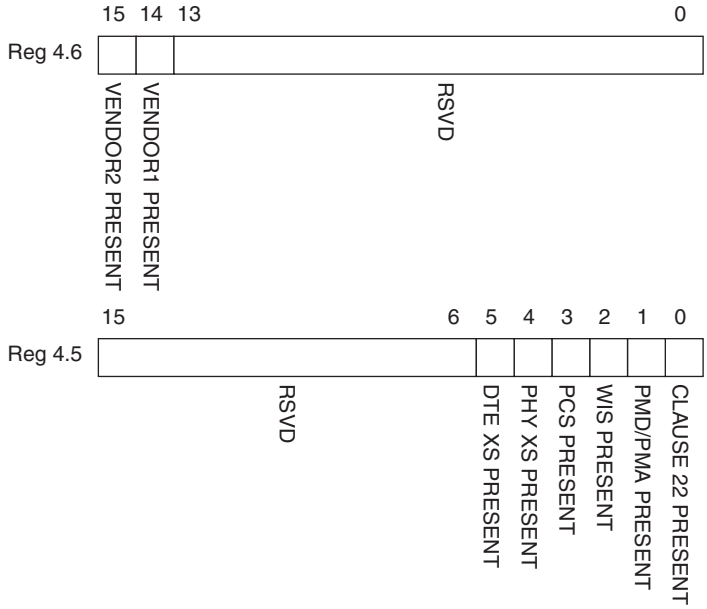

Figure 6-51: **PHY XS Devices in Package Registers**

Table 6-43 shows the PHY XS Devices in Package registers bit definitions.

Table 6-43: **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	PHY XS Present	The block always returns “1” for this bit.	R/O	All “0s”
4.5.4	PHY XS Present	The block always returns “0” for this bit.	R/O	0
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 6-52 shows the MDIO Register 4.8: PHY XS Status 2.

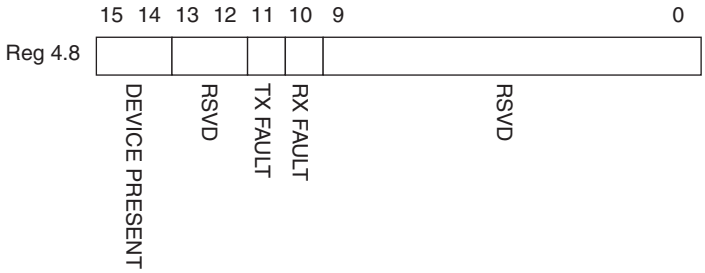

Figure 6-52: **PHY XS Status 2 Register**

Table 6-44 shows the PHY XS Status 2 register bit definitions.

Table 6-44: **PHY XS Status 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.8.15:14	Device Present	The block shall always return “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 6-53 shows the MDIO 4.14 and 4.15 Registers: PHY XS Package Identifier.

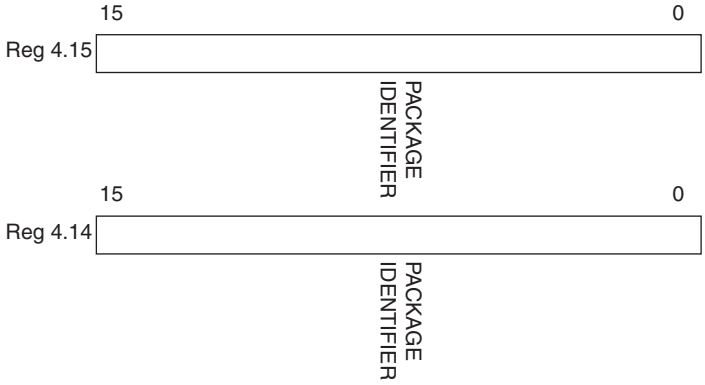

Figure 6-53: **PHY XS Package Identifier Registers**

Table 6-45 shows the Package Identifier registers bit definitions.

Table 6-45: **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 6-54 shows the MDIO Register 4.24: 10G XGXS Lane Status.

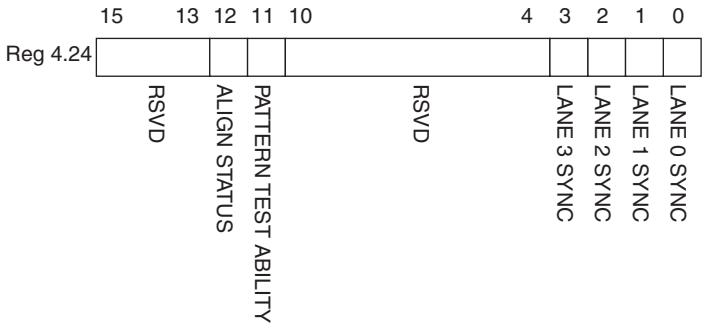

Figure 6-54: **10G PHY XGXS Lane Status Register**

Table 6-46 shows the 10G PHY XGXS Lane register bit definitions.

Table 6-46: **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	-

Table 6-46: **10G PHY XGXS Lane Status Register Bit Definitions *(Continued)***

Bit(s)	Name	Description	Attributes	Default Value
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 6-55 shows the MDIO Register 4.25: 10G XGXS Test Control.

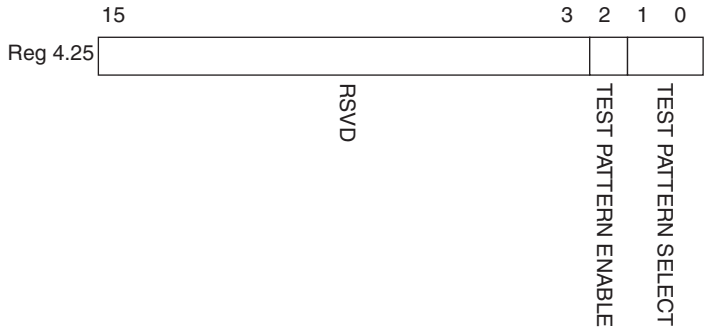

Figure 6-55: **10G PHY XGXS Test Control Register**

Table 6-47 shows the 10G PHY XGXS Test Control register bit definitions.

Table 6-47: **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 6-48 shows the Configuration Vector bit definitions.

Table 6-48: **Configuration Vector Bit Definitions**

Bit(s)	Name	Description
0	Loopback	Sets serial loopback in the RocketIO™ transceivers. See bit 5.0.14 in Table 6-29, page 72.
1	Power Down	Sets the RocketIO transceivers into power down mode. See bit 5.0.11 in Table 6-29, page 72.
2	Reset Local Fault	Clears both TX Local Fault and RX Local Fault bits (status_vector[0] and status_vector[1]). See below.
3	Reset Rx Link Status	Sets the RX Link Status bit (status_vector[7]). See below.
4	Test Enable	Enables transmit test pattern generation. See bit 5.25.2 in Table 6-37, page 79.
6:5	Test Select(1:0)	Selects the test pattern. See bits 5.25.1:0 in Table 6-37, page 79.

Table 6-49 shows the Status Vector bit definitions.

Table 6-49: **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 6-34, page 76. 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 6-34, page 76. 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 6-35, page 77. These four bits are also used to generate the sync_status[3:0] signal described in Table 6-50.
6	Alignment	"1" if the XAUI receiver is aligned over all four lanes, otherwise "0"; see bit 5.24.12 in Table 6-35, page 77. This is also used to generate the align_status signal described in Table 6-50.
7	Rx Link Status	"1" if the Receiver link is up, otherwise "0"; see bit 5.1.2 in Table 6-30, page 73. 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 6-56 shows how the status bits are cleared.

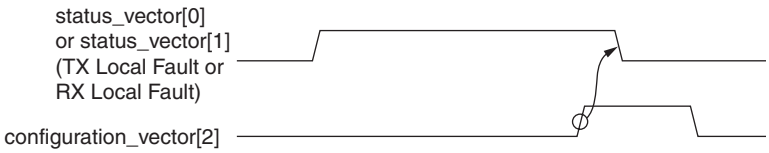

Figure 6-56: **Clearing the Local Fault Status Bits**

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 6-57 shows how the status bit is set.

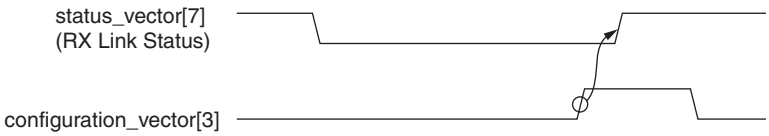

Figure 6-57: **Setting the RX Link Status Bit**

Alignment and Synchronization Status Ports

In addition to the configuration and status interfaces described above, there are always available two output ports signalling the alignment and synchronization status of the receiver. (Table 6-50.)

Table 6-50: **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 7

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 the *LogiCORE XAUI Getting Started Guide* for a complete description of the CORE Generator output files.

Important: 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 Speedgrade 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 = xc2vp7-ff672-6;
```

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

- Virtex-II Pro family devices XC2VP4 and larger, with speedgrade of -6 or better. "FF" flip-chip packaging is required to support the 3.125 Gbps line rate of the RocketIO™ transceivers.
- Virtex-4 FX family devices, XC4VFX20 and larger, with speedgrade of -10 or higher.

Clock Frequencies, Clock Management, and Placement

The XAUI core can have one or two clock domains:

- The refclk domain, derived from the BREFCLK signal used as a reference for the RocketIO in Virtex-II Pro, or derived from the TXOUTCLK1 output of the RocketIO in Virtex-4 FX
- 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) primitives included in the design.

Virtex-II Pro

```
##############################################################################
# Clock frequencies and clock management#
##############################################################################
NET "refclk_buf" TNM_NET="refclk_top";
# Clock rate below is the 10-Gigabit Ethernet speed; change to
# 159.375 MHz for 10-Gigabit Fibre Channel applications
TIMESPEC "TS_refclk_top" = PERIOD "refclk_top" 156.25 MHz;
```

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 RocketIO serial rate of 3.1875 Gbps per lane.

Virtex-4 FX

```
##########################################################################
# Clock frequencies and clock management #
##########################################################################
NET "*GT_TXOUTCLK1*" TNM_NET = TXOUTCLK1_CLK;
TIMESPEC "TS_TXOUTCLK1_CLK" = PERIOD TXOUTCLK1_CLK 160 MHz HIGH 50%;
```

The RocketIO output clock frequency is constrained to 160MHz and this will cover 10-Gigabit Ethernet and Fibre Channel operation.

```
# System DCM constraints
INST “dcm_refclk” DLL_FREQUENCY_MODE = LOW;
INST “dcm_refclk” DUTY_CYCLE_CORRECTION = TRUE;
INST “dcm_refclk” CLK_FEEDBACK = 1X;

INST "dcm_locked_reg1" ASYNC_REG=TRUE;
```

and

```
INST “dcm_txclk” DLL_FREQUENCY_MODE = LOW;
INST “dcm_txclk” DUTY_CYCLE_CORRECTION = TRUE;
INST “dcm_txclk” CLK_FEEDBACK = 1X;
```

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_refclk CLKOUT_PHASE_SHIFT = FIXED;
INST dcm_refclk PHASE_SHIFT = 33;
```

or:

```
##########################################################
#                            #
# ****** 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" CLKOUT_PHASE_SHIFT = FIXED;
INST "dcm_txclk" PHASE_SHIFT = 33;
```

This constraint sets a phase shift on the main output of the system DCM with respect to the input clock; this is used to obtain clock/data alignment on the inbound XGMII interface. See Chapter 8, "Design Considerations," for a description of the clock scheme, and Appendix D, "Calculating the DCM Phase Shift," for information about setting the phase-shift value. These constraints are only present in the example if the external XGMII is used.

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

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

```
NET "refclk_p" LOC = "B14";
NET "refclk_n" LOC = "C14";
```

These lines constrain the clock input to the differential BREFCLK inputs for the XC2VP7 part declared above. If a different part and package is to be used, change the refclk pin locations to the correct location for the BREFCLK on the part.

RocketIO Placement

```
##########################################################################
# Note these UCF parameters are provided as an example of a    #
# particular implementation - targeted at a 2vp7.        #
# If the user wishes to use a different part they can do so    #
# but they must ensure that these parameters are correct.     #
# Guidance can be found in the relevant user guide.        #
##########################################################################
#
#INST "mgt_0_MGT" LOC = "GT_X0Y1";
#INST "mgt_1_MGT" LOC = "GT_X1Y1";
#INST "mgt_2_MGT" LOC = "GT_X2Y1";
#INST "mgt_3_MGT" LOC = "GT_X3Y1";
NET "xaui_tx_l0_p" LOC = "A22";
NET "xaui_tx_l0_n" LOC = "A23";
NET "xaui_tx_l1_p" LOC = "A17";
NET "xaui_tx_l1_n" LOC = "A18";
NET "xaui_tx_l2_p" LOC = "A11";
NET "xaui_tx_l2_n" LOC = "A12";
NET "xaui_tx_l3_p" LOC = "A6";
NET "xaui_tx_l3_n" LOC = "A7";
NET "xaui_rx_l0_p" LOC = "A21";
NET "xaui_rx_l0_n" LOC = "A20";
NET "xaui_rx_l1_p" LOC = "A16";
NET "xaui_rx_l1_n" LOC = "A15";
NET "xaui_rx_l2_p" LOC = "A10";
NET "xaui_rx_l2_n" LOC = "A9";
NET "xaui_rx_l3_p" LOC = "A5";
NET "xaui_rx_l3_n" LOC = "A4";
```

These constraints lock down the placement of the RocketIO transceivers.

```
INST "mgt_0_reclock_align" LOC = "SLICE_X15Y72";
INST "mgt_1_reclock_align" LOC = "SLICE_X27Y72";
INST "mgt_2_reclock_align" LOC = "SLICE_X39Y72";
INST "mgt_3_reclock_align" LOC = "SLICE_X51Y72";
INST "mgt_?_reclock_align" ASYNC_REG = TRUE;
```

The constraints above are necessary to place synchronizing registers in the appropriate place for each RocketIO transceiver; it is critical that the registers are correctly placed in your final application for successful use of the XAUI core in Virtex-II Pro designs. See the *RocketIO User Guide* for the correct placement of these registers for your particular part, package and RocketIO location in the design.

XGMII

```
# XGMII input/output buffer attributes
INST txd_ddr* IOB = TRUE;
INST txd_ddr* IOSTANDARD = HSTL_I;

INST txc_ddr* IOB = TRUE;
INST txc_ddr* IOSTANDARD = HSTL_I;

INST xgmii_tx_clk_ibufg IOSTANDARD = HSTL_I;
INST xgmii_tx_clk_ibufg IOBDELAY = NONE;

NET xgmii_rxd<*> IOSTANDARD = HSTL_I;
INST rxd_ddr* IOB = TRUE;

NET xgmii_rxc<?> IOSTANDARD = HSTL_I;
INST rxc_ddr* IOB = TRUE;

NET xgmii_rx_clk IOSTANDARD = HSTL_I;
NET xgmii_rx_clk IOB = TRUE;
```

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

```
INST "*xaui_core/BU2/U0?xgmii_tx_i_?_falling_reclock_bits*_i_reg_i" TNM =
"xgmiitxfalling_group";
TIMESPEC "TS_xgmiitxfalling" = FROM "xgmiitxfalling_group" 2.0 ns; # 2.5 ns is specified
maximum
```

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 = 5.0 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;
```

These constraints set the correct attributes for the registers at the edge of the MDIO block.

Other Constraints

```
##################################################################
# Floorplanning                          #
#                                 #
# Again, care should be taken to ensure these directives are   #
# correct for your design.                    #
##################################################################
INST xaui_core AREA_GROUP = xaui_top_group;
AREA_GROUP xaui_top_group RANGE=SLICE_X0Y56:SLICE_X67Y79;
```

The constraint above keeps the XAUI core within a restricted area of the device. Experimentation with placement and timing results in your design is required to determine if floorplanning is required.

Chapter 8

Design Considerations

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

Clocking: Virtex-II Pro

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

Reference Clock

The Virtex-II Pro transceivers require a reference clock of 156.25 MHz to operate at a line rate of 3.125 Gbps.

Internal Client-side Interface

The simplest clocking scheme is for the internal client-side interface, shown in Figure 8-1. In this clocking arrangement, it is likely that the usrclk output of the BUFG primitive will be used for as the system clock in the user application.

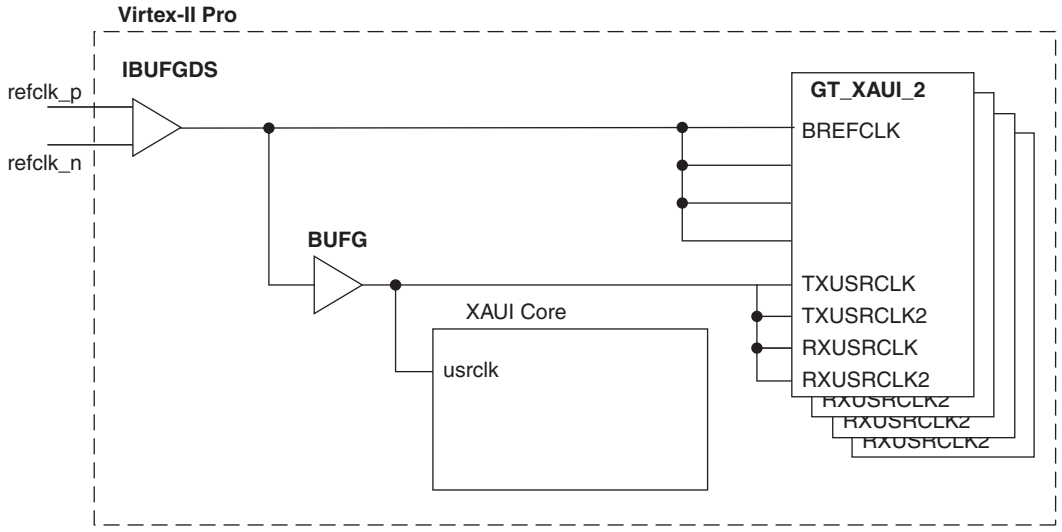

Figure 8-1: **Clock Scheme for Internal Client-side Interface: Virtex-II Pro**

External XGMII Interface: No Transmit Elastic Buffer

The clock scheme in Figure 8-2 shows the clocking arrangement for the external XGMII client-side interface with no transmit elastic buffer. RocketIO™ clocking requirements are that the TXUSRCLK inputs are derived from the BREFCLK inputs; it follows that the xgmii_tx_clk input *must* be derived from the refclk_p/refclk_n differential pair in the system.

The usrclk, usrclk_90, usrclk_180, usrclk_270 pins are used to clock DDR input and output registers in the correct phasing for XGMII signalling.

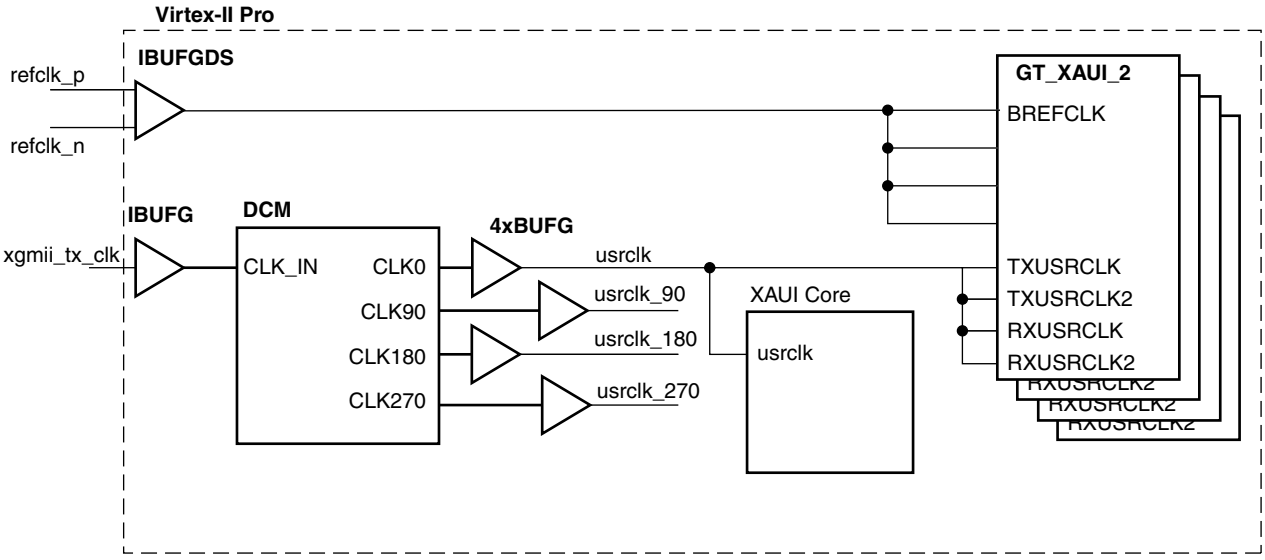

Figure 8-2: **Clock Scheme for External XGMII Client-side Interface without Transmit Elastic Buffer: Virtex-II Pro**

External XGMII Interface: Transmit Elastic Buffer

Often, it cannot be arranged that the inbound transmit data clock domain is derived from the BREFCLK reference clock. In these circumstances, the optional transmit elastic buffer in the core must be used. Figure 8-3 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 usrclk, usrclk_90, usrclk_180, usrclk_270 signals are used to clock DDR output registers in the correct phasing for XGMII signalling. The tx_clk0, tx_clk180 signals are used to clock DDR input registers.

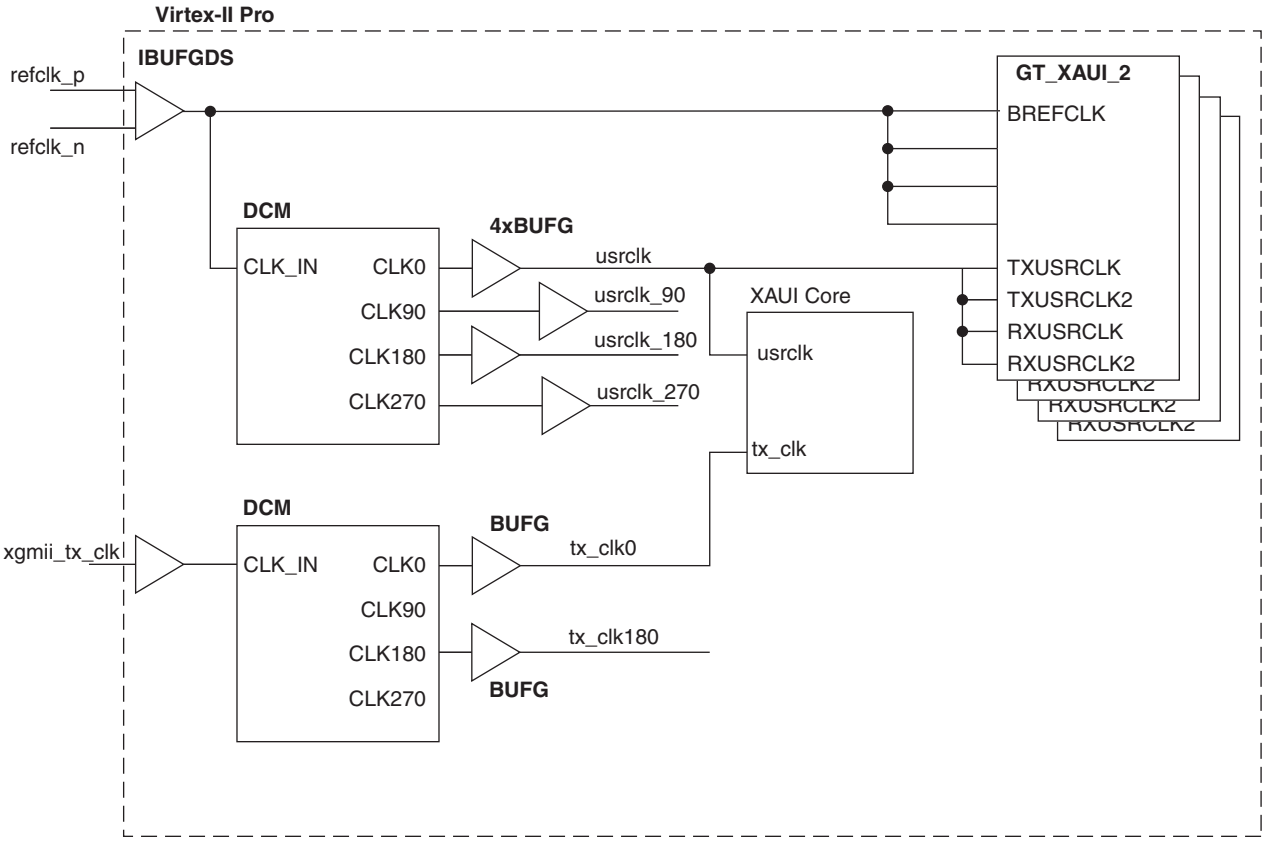

Figure 8-3: **Clock Scheme for External XGMII Client-side Interface with Transmit Elastic Buffer: Virtex-II Pro**

Use of BREFCLK2

BREFCLK2 may be used in place of BREFCLK in any of the examples shown. As well as changing the wiring in the RocketIO instantiation, the setting of the REFCLKSEL pin must be changed from '0' to '1' and the input pin for the clock changed in the UCF. See the *RocketIO User Guide* for more information on the selection of BREFCLK/BREFCLK2 in RocketIO™ transceivers.

Reducing Global Clock Buffers Use in DDR Interfaces

The clock schemes above are fairly expensive in global clock buffer resources due to their use in providing both the in-phase and out-of-phase clocks for the DDR registers; eight global clock buffers are used in the worst case.

To halve the number of clock buffers used in each DDR register implementation, see Xilinx Application Note XAPP685, "High-Speed Clock Architecture for DDR Designs Using Local Inversion", which can be obtained at:

http://direct.xilinx.com/bvdocs/appnotes/xapp685.pdf

Clocking: Virtex-4

The clocking schemes 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 Gbps.

Transceiver placement

Common to all schemes shown is that a single GT11CLK_MGT block is used to feed the MGT clock tree for all 4 transceivers; it follows that all 4 transceivers in an instance of the core *must* be in a single column of MGT tiles. In addition, timing requirements will be more easily met if the 4 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 RocketIO MGT clock distribution.

Internal Client-side Interface

The simplest clocking scheme is that for the internal client interface, as shown in Figure 8-4. The GT11 transceiver primitives require a 78.125MHz 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 the user logic.

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

the Calibration Block User Guide before changing this clock. (See Answer Record 22477 for information about the Calibration Block User Guide.)

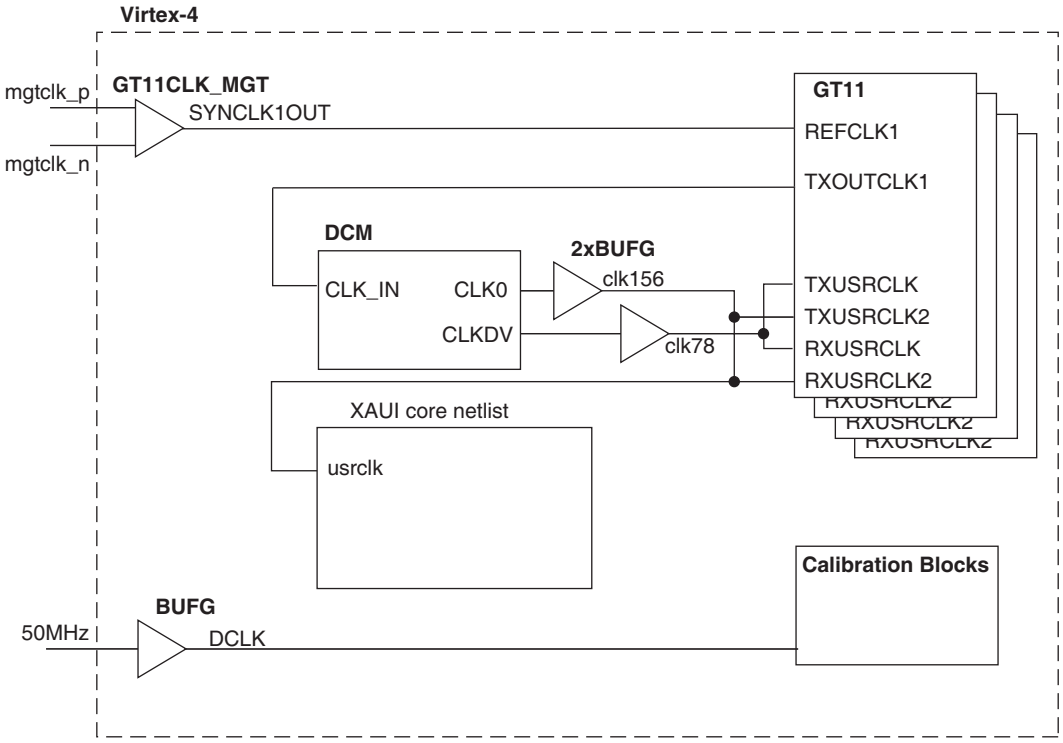

Figure 8-4: **Clock Scheme for Internal Client-side Interface: Virtex-4**

External XGMII Interface: No Transmit Elastic Buffer

The clock scheme in Figure 8-5 shows the clocking arrangement for the external XGMII client-side interface with no transmit elastic buffer on Virtex-4 devices.

RocketIO clocking requirements are that the TXUSRCLK, TXUSRCLK2, RXUSRCLK, RXUSRCLK2 inputs are derived from the same clock that drives the MGTCLK inputs; it follows that 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 signalling.

A dedicated clock is used for the calibration blocks. The example design uses a 50 MHz clock. You may choose to use a different frequency to allow sharing of clock resources. See the Calibration Block User Guide before changing this clock. (See Answer Record 22477 for information about the Calibration Block User Guide.)

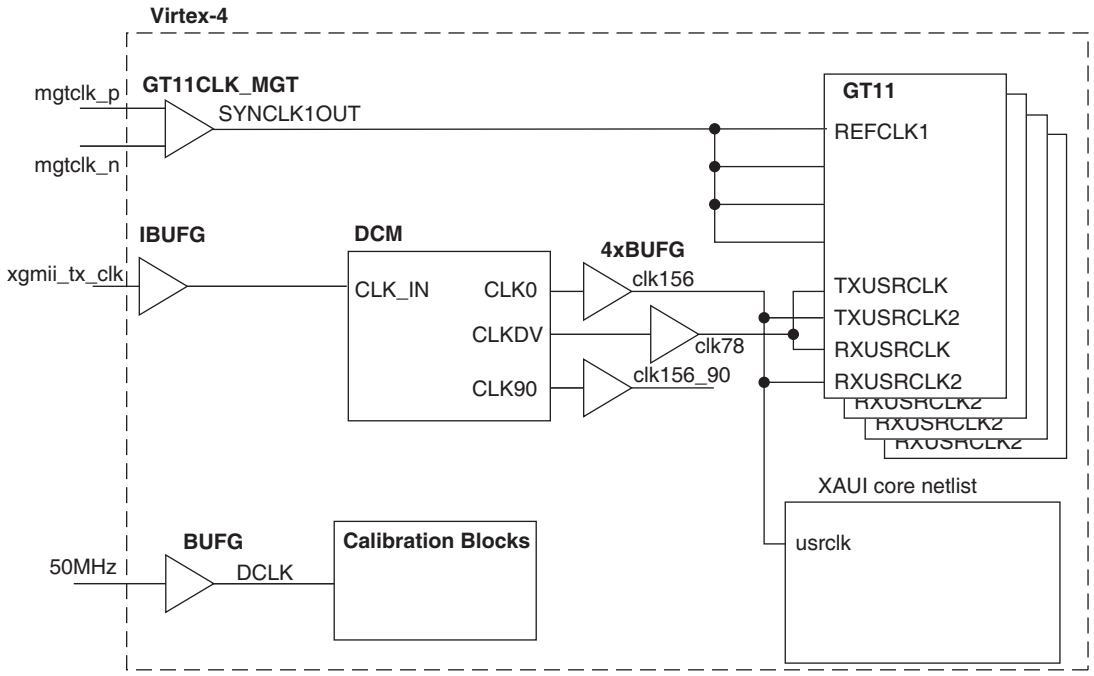

Figure 8-5: **Clock Scheme for External XGMII Client-side Interface without Transmit Elastic Buffer: Virtex-4**

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 8-6 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 signalling. 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 may choose to use a different frequency to allow sharing of clock resources. See the Calibration Block User Guide before changing this clock. (See Answer Record 22477 for information about the Calibration Block User Guide.)

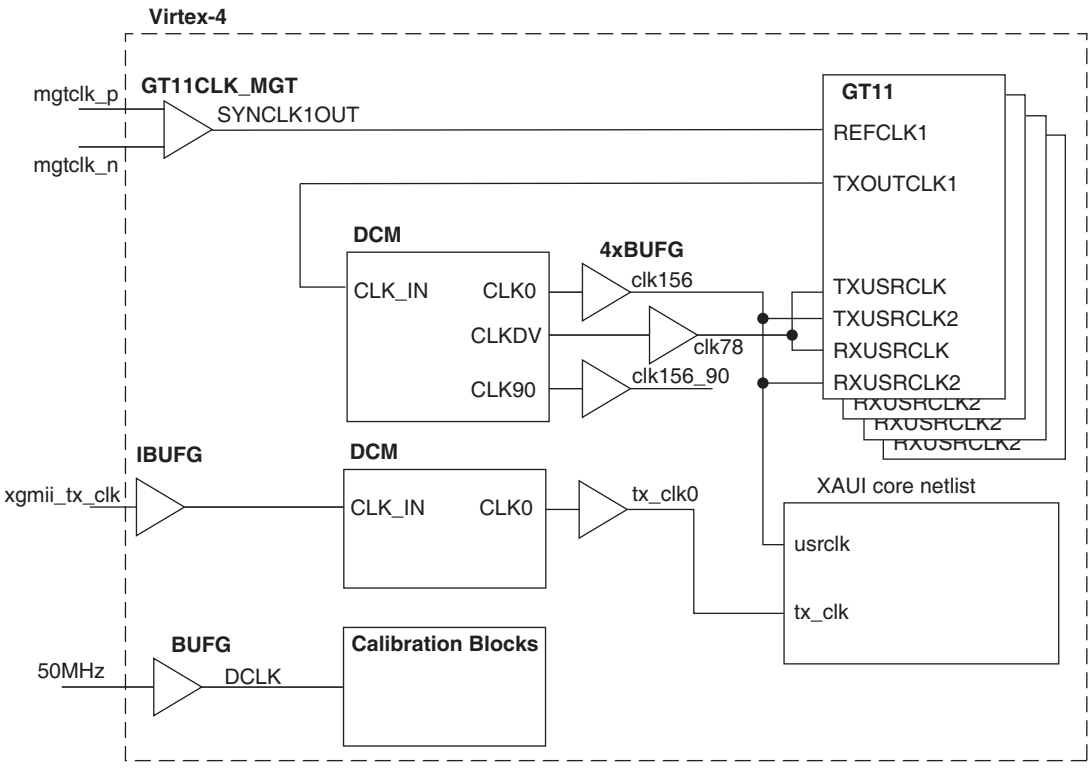

Figure 8-6: **Clock Scheme for External XGMII Client-side Interface with Transmit Elastic Buffer: Virtex-4**

Using the Core in XC2VP4 Devices

The Virtex-II Pro family XC2VP2 and XC2VP4 devices are unusual in that they have four RocketIO transceivers, two on top of the die and two on the bottom. This requires particular care in implementation of a XAUI core.

The primary issue is that each pair of transceivers are fed by a different BREFCLK/ BREFCLK2 low jitter clock signal, one for the top edge of the chip and one for the bottom edge. These must be driven in a manner similar to that shown in Figure 8-7.

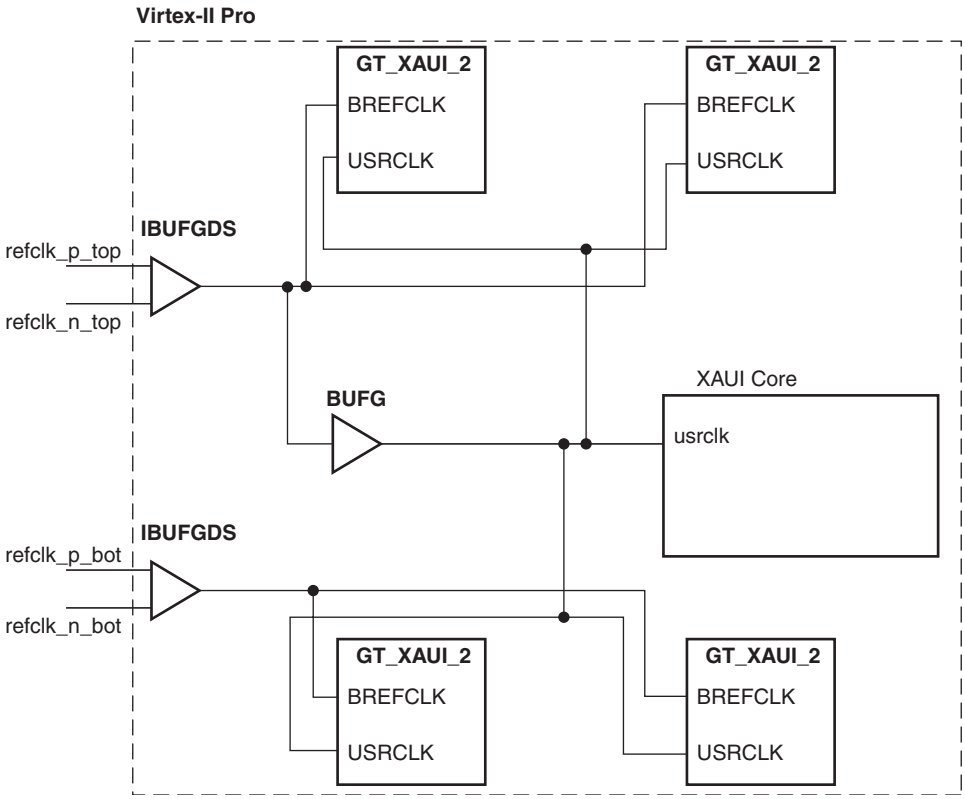

Figure 8-7: **Clock Distribution for XC2VP2, XC2VP4**

As shown in Figure 8-7, usrclk derived from the upper BREFCLK has been used to provide the system clock for the entire XAUI core as well as the fabric ports of all four RocketIO transceivers. For this reason, the BREFCLK for the top and bottom transceivers ***must be derived from the same clock source***. If this is not the case, there is a high risk of unreliable operation of the core.

Multiple Core Instances - Virtex-II Pro

In a large design it may be necessary or desirable to have more than one instance of the XAUI core on a single FPGA.

If the two instances are placed in RocketIOs such that one is on the top edge of the device and one is on the bottom edge of the device, the implementation is fairly straightforward. Figure 8-8 shows an example implementation on Virtex-II Pro.

Note that the two cores are operating in different clock domains, so careful design and standard clock-domain-crossing techniques will be required at any point where data or control passes from one side to the other.

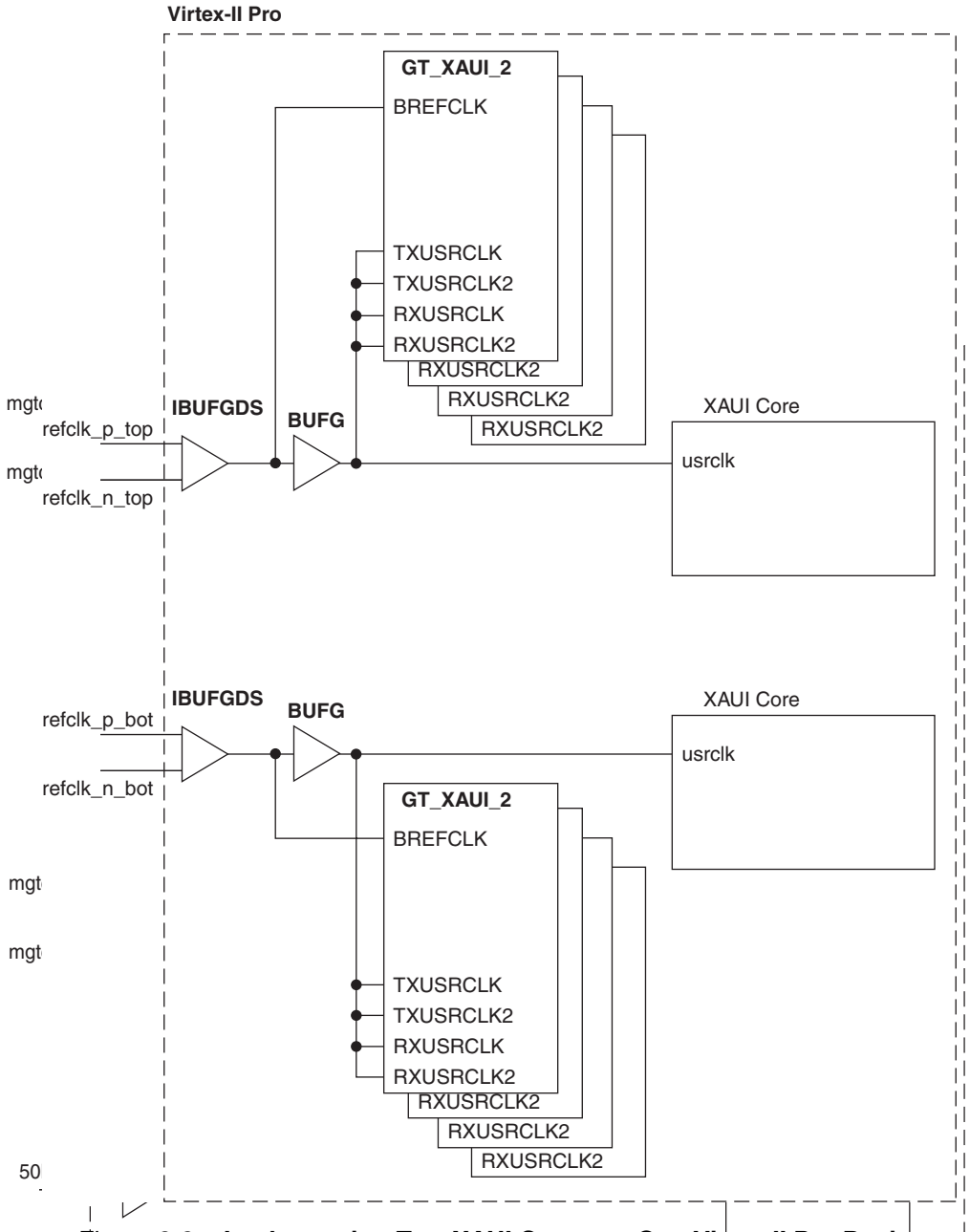

Figure 8-8: **Implementing Two XAUI Cores on One Virtex-II Pro Device**

If the cores are placed such that any one of the cores is split across top and bottom edges of the die, the technique shown in "Using the Core in XC2VP4 Devices" above must be used to force the use of a single clock domain across the split core.

Using the Core in XC4VFX20 Devices

The Virtex-4 FX family XC4VFX20 device is unusual in that it has four RocketIO transceivers split between two columns, which requires particular care in implementation of a XAUI core.

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 chip, derived from the same clock source).

As shown in Figure 8-9, the usrclk derived from one TXOUTCLK has been used to provide the system clock for the entire XAUI core as well as the fabric ports of all four RocketIO transceivers. The Virtex-4 example design instantiates the necessary circuitry to ensure that the transceivers are all phase aligned.

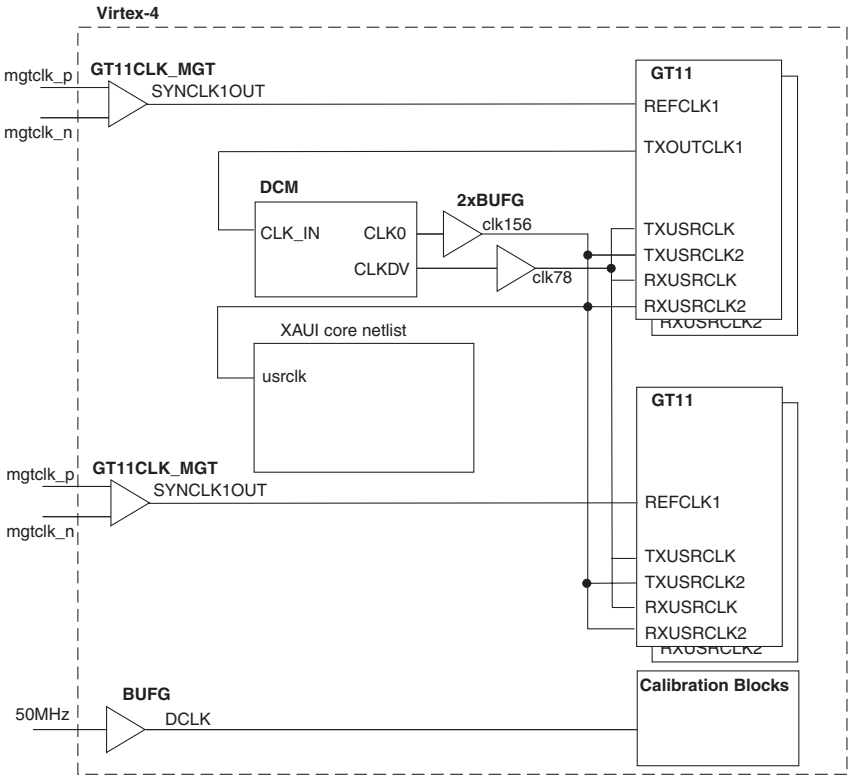

Figure 8-9: **Clocking on the XC4VFX20**

Multiple Core Instances: Virtex-4

If more than one instance of the XAUI core is implemented in a Virtex-4 device, then 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 (similar to the Virtex-II Pro case illustrated in Figure 8-8).

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

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.

Chapter 9

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 Core Generator output file. This can be used for simulation of the block in the design phase of a project.

Using the Simulation Model

For information on 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 project directory. Details of the Core Generator outputs can be found in Appendix B, "Core Directory Structure."

For VHDL

component_name.vhd

For 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.

Once your entire design is complete, create:

- An XST project file *top_level_module_name*.prj listing all the user 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.

Once your entire design is complete, create:

- An XST project file *top_level_module_name*.prj listing all the user 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 tools is used to translate and merge the individual design netlists into a single design database, the 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 the user design netlist into the 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 the user design's logic components (mapped physical logic cells) contained within an 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 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 -w top_level_module_name.ncd
```

Post-Implementation Simulation

The purpose of post-implementation simulation is to verify that the design as implemented in the FPGA works as expected.

Generating a Simulation Model

To generate a chip-level simulation netlist for your design, the netgen command must be run.

For VHDL

```
$ netgen -sim -ofmt vhdl \
    -tm netlist top_level_module_name.ncd \
    top_level_module_name_postimp.vhd
```

For Verilog

```
$ netgen -sim -ofmt verilog \
    -tm netlist top_level_module_name.ncd \
    top_level_module_name_postimp.v
```

Using the Model

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

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 software.

Appendix A

Related Information

Xilinx XAUI Web Page

All information directly related to the Xilinx XAUI core can be found at:

http://www.xilinx.com/systemio/xaui/index.htm

Xilinx Support

Support resources such as Application Notes, User Guides, Reference and Software Manuals, and Answer Records can all be accessed from the Xilinx support website located at:

http://www.xilinx.com/support/

Ethernet Specification

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

- *IEEE Std. 802.3-2002*
- *IEEE Std. 802.3ae-2002*

Other Information

The website of the 10-Gigabit Ethernet Consortium at the University of New Hampshire's Interoperability Lab is an excellent source of information on 10-Gigabit Ethernet technology:

http://www.iol.unh.edu/consortiums/10gec/index.html

Appendix B

Core Directory Structure

This appendix briefly describes the files generated by the CORE Generator for the XAUI core. See the *XAUI Getting Started Guide* for more detailed information.

Directory Structure and File Descriptions

VHDL Design Flow

Directory Structure

Figure B-1 illustrates the directories and files created by the CORE Generator for a VHDL based project. <*project_dir*> is the CORE Generator project directory; <*component_name*> is the component name as entered in the XAUI core customization dialog box. The implement and timing simulation directories are only present when the core is generated with a full license.

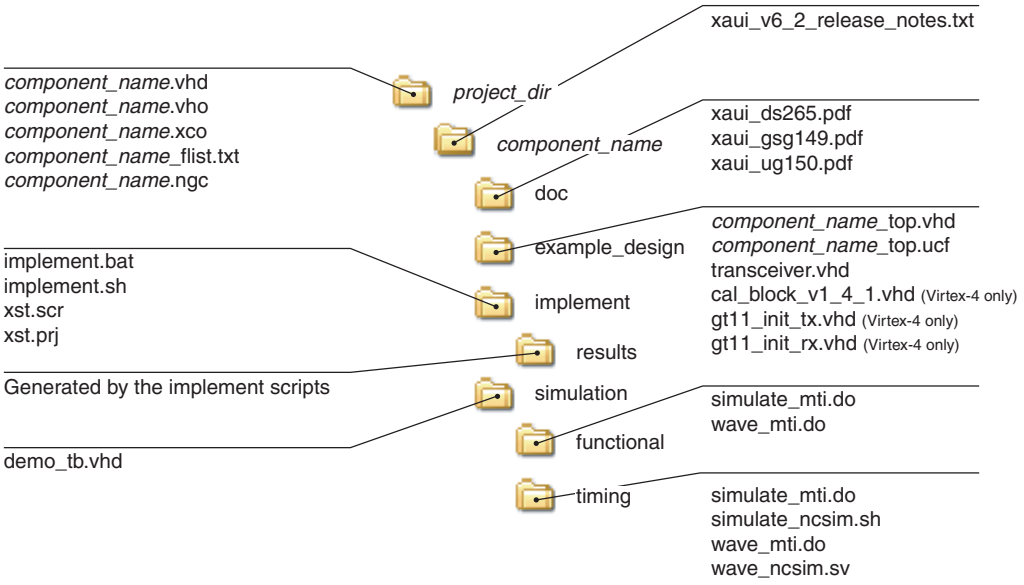

Figure B-1: **XAUI Core Directories and Files: VHDL**

Project Directory (*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*.vhd**

VHDL structural simulation model. File used to support VHDL functional simulation of a core.

***component_name*.vho**

A VHDL template for the core. This can be copied into the user design.

***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 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.

***component_name*_flist.txt**

A text file listing all of the output files produced when customized core was generated in the CORE Generator.

project_dir/component_name

xaui_v6_2_release_notes.txt

The XAUI release notes text file, which contains updates and information about the core.

project_dir/component_name/doc

xaui_ds265.pdf

The *XAUI Data Sheet*.

xaui_gsg149.pdf

The *XAUI Getting Started Guide.*

xaui_ug150.pdf

The *XAUI User Guide*.

project_dir/component_name/example_design

This directory contains the support files necessary for a VHDL implementation of the example design.

***component_name*_top.vhd**

The top-level entity for the example design.

transceiver.vhd (Virtex-II Pro only)

transceivers.vhd (Virtex-4 only)

Wrappers for the RocketIO™ transceivers.

cal_block_v1_4_1.vhd (Virtex-4 only)

Virtex-4 FX Calibration Block.

gt11_init_tx.vhd (Virtex-4 only)

gt11_init_rx.vhd (Virtex-4 only)

Virtex-4 transceiver reset circuitry.

***component_name*_top.ucf**

User constraints file for the core and example design.

project_dir/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.

implement.sh

A UNIX shell script that processes the example design through the Xilinx tool flow.

implement.bat

A Windows batch file that process the example design through the Xilinx tool flow.

xst.prj

The XST project file for the example design.

xst.scr

The XST script file for the example design.

project_dir/component_name/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.

routed.vhd

The back-annotated SimPrim based VHDL design. Used for timing simulation.

routed.sdf

The timing information for simulation is contained in this file.

project_dir/component_name/simulation

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

demo_tb.vhd

The VHDL demonstration test bench for the XAUI core.

project_dir/component_name/simulation/functional

simulate_mti.do

A 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.

wave_mti.do

A ModelSim macro file that opens a wave window and adds interesting signals to it. It is called by the simulate_mti.do macro file.

simulate_ncsim.do

A UNIX shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using NC-Sim.

wave_ncsim.sv

A NC-Sim macro file that opens a wave windows and adds interesting signals to it. It is called used by the simulate_ncsim.sh script.

project_dir/component_name/simulation/timing

simulate_mti.do

A ModelSim macro file that compiles the timing simulation model and the demonstration test bench then runs the timing simulation to completion.

wave_mti.do

A ModelSim macro file that opens a wave window and adds interesting signals to it. It is called by the simulate_mti.do macro file.

simulate_ncsim.do

A UNIX shell script that compiles the test bench and the timing model then runs the timing simulation to completion using NC-Sim.

wave_ncsim.sv

A NC-Sim macro file that opens a wave windows and adds interesting signals to it. It is called used by the simulate_ncsim.sh script.

Verilog Design Flow

Directory Structure

Figure B-2 illustrates the directories and files created by the CORE Generator for a VHDL based project. <*project_dir*> is the CORE Generator project directory; <*component_name*> is the component name as entered in the XAUI core customization dialog box. The implement and timing simulation directories are only present when the core is generated with a full or hardware evaluation license.

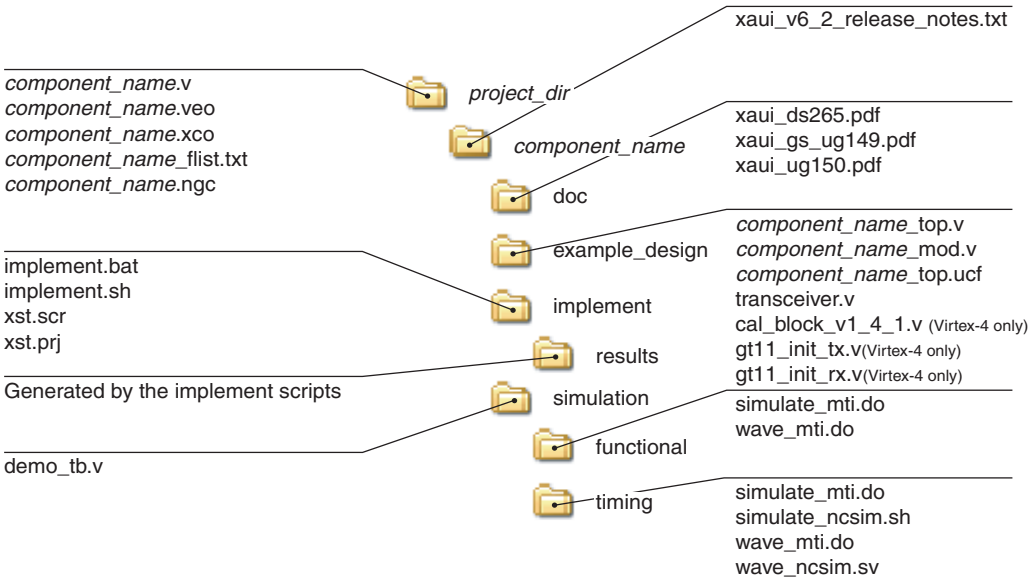

Figure B-2: **XAUI Core Directories and Files: Verilog**

Project Directory (*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**

Verilog structural simulation model. File used to support Verilog functional simulation of a core.

***component_name*.veo**

A Verilog template for the core. This can be copied into the user design.

***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 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.

***component_name*_flist.txt**

A text file listing all of the output files produced when customized core was generated in the CORE Generator.

project_dir/component_name

xaui_v6_2_release_notes.txt

The XAUI release notes text file, which contains updates and information about the core.

project_dir/component_name/doc

xaui_ds265.pdf

The *XAUI Data Sheet*.

xaui_gsg149.pdf

The *XAUI Getting Started Guide.*

xaui_ug150.pdf

The *XAUI User Guide*.

project_dir/component_name/example_design

This directory contains the support files necessary for a Verilog implementation of the example design.

***component_name*_top.v**

The top level entity for the example design.

transceiver.v (Virtex-II Pro only)

transceivers.v (Virtex-4 only)

Wrappers for the RocketIO transceivers.

cal_block_v1_4_1.v (Virtex-4 only)

Virtex-4 FX Calibration Block.

gt11_init_tx.v (Virtex-4 only)

gt11_init_rx.v (Virtex-4 only)

Virtex-4 transceiver reset circuitry.

***component_name*_top.ucf**

User constraints file for the core and example design.

project_dir/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.

implement.sh

A UNIX shell script that processes the example design through the Xilinx tool flow.

implement.bat

A Windows batch file that process the example design through the Xilinx tool flow.

xst.prj

The XST project file for the example design.

xst.scr

The XST script file for the example design.

project_dir/component_name/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.

routed.v

The back-annotated SimPrim based Verilog design. Used for timing simulation.

routed.sdf

The timing information for simulation is contained in this file.

project_dir/component_name/simulation

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

demo_tb.v

The Verilog demonstration test bench for the XAUI core.

project_dir/component_name/simulation/functional

simulate_mti.do

A 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.

wave_mti.do

A ModelSim macro file that opens a wave window and adds interesting signals to it. It is called by the simulate_mti.do macro file.

simulate_ncsim.do

A UNIX shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using NC-Sim.

wave_ncsim.sv

A NC-Sim macro file that opens a wave windows and adds interesting signals to it. It is called used by the simulate_ncsim.sh script.

project_dir/component_name/simulation/timing

simulate_mti.do

A ModelSim macro file that compiles the timing simulation model and the demonstration test bench then runs the timing simulation to completion.

wave_mti.do

A ModelSim macro file that opens a wave window and adds interesting signals to it. It is called by the simulate_mti.do macro file.

simulate_ncsim.do

A UNIX shell script that compiles the test bench and the timing model then runs the timing simulation to completion using NC-Sim.

`wave_ncsim.sv`

A NC-Sim macro file that opens a wave windows and adds interesting signals to it. It is called used by the simulate_ncsim.sh script.

Appendix C

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 include:

- 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 Virtex-II Pro hardware test platforms within Xilinx. In particular, the core has been used in a test platform design with the Xilinx LogiCORE 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.

Appendix D

Calculating the DCM Phase Shift

DCM Phase Shifting Requirement

A DCM 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 to skew the clock; this performs static alignment by using the transmit clock DCM 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 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 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 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 platforms. 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.

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.

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 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.