LogiCORE IP 10-Gigabit Ethernet MAC v11.3

User Guide

UG773 April 24, 2012

XILINX®

Revision History

Date	Version	Revision
03/01/11	1.0	Initial Xilinx Release with AXI interface
10/19/11	1.1	Update for ISE Release 13.3. Added Using the AXI4-Lite Interface to access PHY registers over MDIO, page 57; Updated Core Latency, page 117.
04/24/12	1.2	Updated for ISE Release 14.1. Added hardware validation note; clarified AXI4 compliance of aborted frame transmission; Figure 6-6 updated; Table 7-25 updated; clarified legal values of the MTU registers.

Table of Contents

XILINX®

Chapter 1

Introduction

The Xilinx® LogiCORE™ 10-Gigabit Ethernet MAC core is a fully-verified solution 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 10-Gigabit Ethernet MAC core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.

System Requirements

Windows

- Windows XP Professional 32-bit/64-bit
- Windows 7 Professional 32-bit/64-bit
- Windows Server 2008 32-bit/64-bit

Linux

- Red Hat Enterprise Linux WS v4.0 32-bit/64-bit
- Red Hat Enterprise Desktop v5.0 32-bit/64-bit
- SUSE Linux Enterprise Desktop 11 32-bit/64-bit

Software

- ISE® Design Suite v14.1

About the Core

The 10-Gigabit Ethernet MAC core is a Xilinx CORE Generator™ core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see the 10-Gigabit Ethernet MAC product page. For information about licensing options, see Chapter 2, Licensing the Core.

Recommended Design Experience

Although the 10-Gigabit Ethernet MAC 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 UCFs 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 10-Gigabit Ethernet MAC core, see the documents located on the 10-Gigabit Ethernet MAC product page.

- *LogiCORE IP 10-Gigabit Ethernet MAC Data Sheet*
- *LogiCORE IP 10-Gigabit Ethernet MAC Release Notes*

For updates to this document, see the *LogiCORE IP 10-Gigabit Ethernet MAC User Guide*, also located on the product page.

Technical Support

To obtain technical support specific to the 10-Gigabit Ethernet MAC core, visit www.xilinx.com/support/index.htm. Questions are routed to a team of engineers with expertise using the 10-Gigabit Ethernet MAC core.

Xilinx provides technical support for use of this product as described in the *LogiCORE 10-Gigabit Ethernet MAC User 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 10-Gigabit Ethernet MAC core and the documentation supplied with the core.

10-Gigabit Ethernet MAC Core

For comments or suggestions about the 10-Gigabit Ethernet MAC core, submit a WebCase from www.xilinx.com/support/clearexpress/websupport.htm. Be sure to include this information:

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

Document

For comments or suggestions about this document, submit a WebCase from www.xilinx.com/support/clearexpress/websupport.htm. Be sure to include this information:

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

XILINX®

Chapter 2

Licensing the Core

This chapter provides instructions for obtaining a license for the 10-Gigabit Ethernet MAC core, which you must do before using the core in your designs. The 10-Gigabit Ethernet MAC core is provided under the terms of the Xilinx LogiCORE IP Site License Agreement or the Xilinx LogiCORE IP Project License Agreement. Purchase of the core entitles you to technical support and access to updates for a period of one year.

Before you Begin

This chapter assumes that you have installed all required software specified on the product page for this core.

License Options

The 10-Gigabit Ethernet MAC core provides three licensing options. After installing the required Xilinx® ISE® Design Suite and IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator™ tool. This key lets you assess core functionality with either the example design provided with the 10-Gigabit Ethernet MAC core, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place and route the design, evaluate timing, and perform back-annotated gate-level simulation of the core using the demonstration test bench provided with the core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before *timing out* (ceasing to function) at which time it can be reactivated by reconfiguring the device.

Full

The Full license key is available when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Functional simulation support
- Full implementation support including place and route, and bitstream generation
- Full functionality in the programmed device with no timeouts

Obtaining Your License Key

This section contains information about obtaining a simulation, full system hardware evaluation, and full license keys.

Simulation License

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator tool.

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license, perform these steps:

1. Navigate to the 10-Gigabit Ethernet MAC product page for this core.
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE Design Suite and IP Service Packs.

Obtaining a Full License Key

To obtain a Full license key, you must purchase a license for the core. After you purchase a license, a product entitlement is added to your Product Licensing Account on the Xilinx Product Download and Licensing site. The Product Licensing Account Administrator for your site receives an email from Xilinx with instructions on how to access a Full license and a link to access the licensing site. You can obtain a full key through your account administrator, or your administrator can give you access so that you can generate your own keys.

Further details can be found at:

www.xilinx.com/products/intellectual-property/ipaccess_fee.htm

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email is sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

Chapter 3

Core Architecture

This chapter describes the overall architecture of the 10-Gigabit Ethernet MAC core and also describes the major interfaces to the core.

System Overview

Figure 3-1 shows a typical Ethernet system architecture and the place of the MAC within it. The MAC and all the blocks to the right are defined in the IEEE specifications for Ethernet.

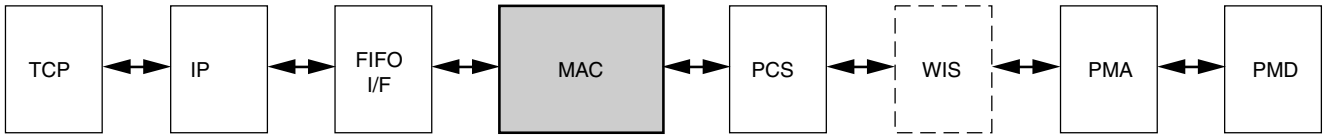

Figure 3-1: **Typical Ethernet System Architecture**

Figure 3-2 shows the 10-Gigabit Ethernet MAC core connected to a physical layer (PHY) device such as an optical module using the XGMII interface.

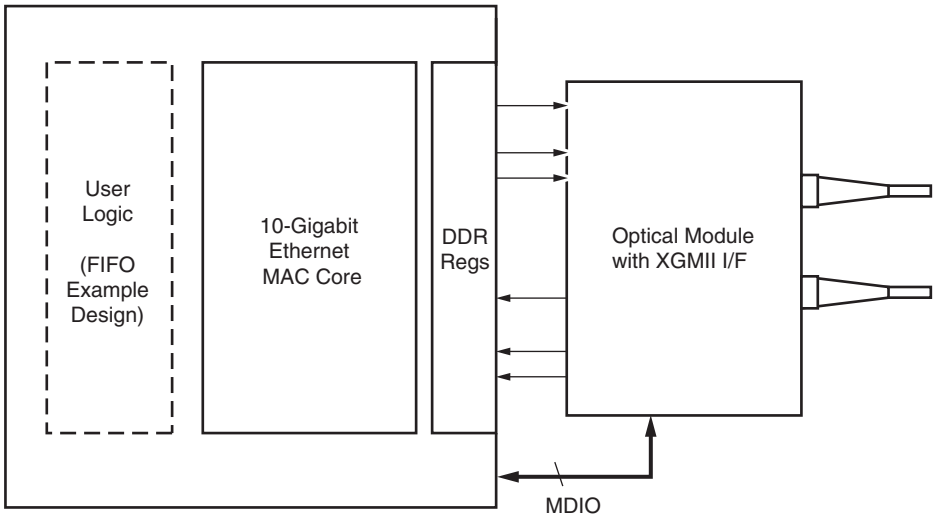

Figure 3-2: **10-Gigabit Ethernet MAC Core Connected to PHY with XGMII Interface**

The 10-Gigabit Ethernet MAC core is designed to be attached to the Xilinx® XAUI core available at the XAUI product page; this gives the advantage over XGMII of reduced pin count and improved operating distance. Figure 3-3 shows the two cores in a system using an XPAK optical module. In this case, the XGMII interface is omitted from the 10-Gigabit

Ethernet MAC core at customization time and the internal FPGA logic interface is used to interface to the XAUI core.

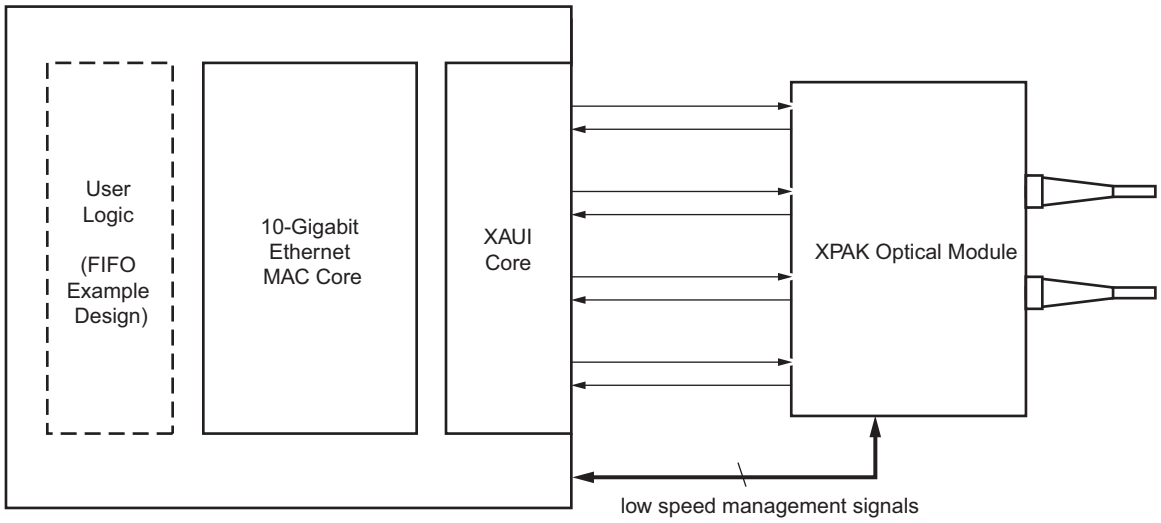

Figure 3-3: **10-Gigabit Ethernet MAC Core Used with Xilinx XAUI Core**

See Interfacing to the Xilinx XAUI Core in Chapter 10 for details on using the two cores together in a system.

The 10-Gigabit Ethernet MAC core can also be attached to the Xilinx® RXAUI core and the Xilinx 10G Ethernet PCS/PMA core. See Interfacing with the RXAUI Core and Interfacing to the 10-Gigabit Ethernet PCS/PMA Core in Chapter 10 for details.

Functional Description

Figure 3-4 shows a block diagram of the implementation of the 10-Gigabit Ethernet MAC core. The major functional blocks of the core include these blocks:

- AXI4-Stream Interface — designed for simple attachment of user logic
- Transmitter
- Receiver
- Flow Control block — implements both Receive Flow Control and Transmit Flow Control
- Reconciliation Sublayer (RS) — processes XGMII Local Fault and Remote Fault messages and handles DDR conversion
- AXI4-Lite Management interface and MDIO (optional)
- Statistics counters (optional)
- XGMII interface - connection to the physical layer device or logic.

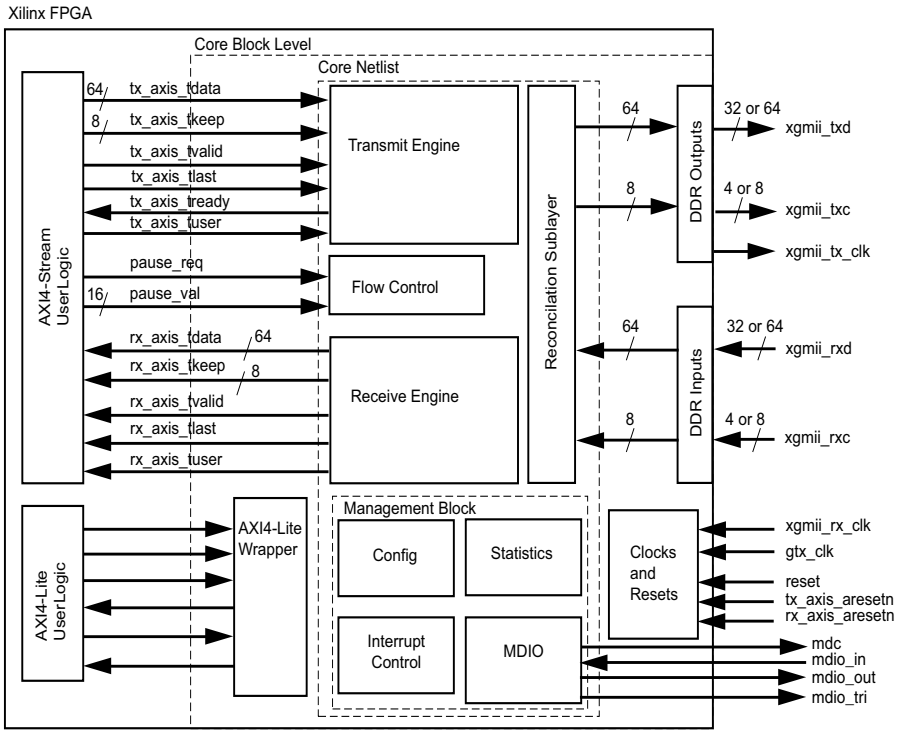

Figure 3-4: **Implementation of the 10-Gigabit Ethernet MAC Core**

Some customer applications do not require an external XGMII interface but instead need a connection to user logic. This application architecture is shown in Figure 3-5.

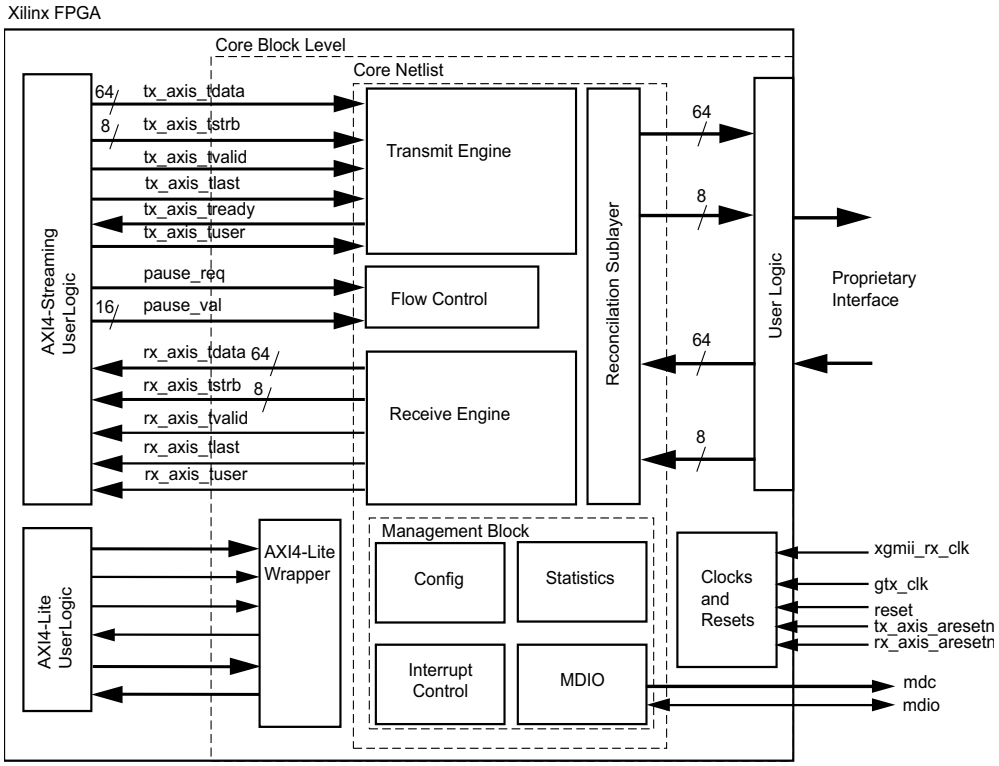

Figure 3-5: **Implementation of the Core with User Logic on PHY Interface**

Core Interfaces and Modules

AXI4-Stream Interface - Transmit

The signals of the transmit AXI4-Stream interface are shown in Table 3-1. See Chapter 6, Interfacing to the Core: Data Interfaces for details on connecting to the transmit interface.

Table 3-1: **AXI4-Stream Interface Ports - Transmit**

Name	Direction	Description
tx_axis_aresetn	IN	AXI4-Stream active-Low reset for Transmit path XGMAC
tx_axis_tdata[63:0]	IN	AXI4-Stream Data to XGMAC
tx_axis_tkeep[7:0]	IN	AXI4-Stream Data Control to XGMAC
tx_axis_tvalid	IN	AXI4-Stream Data Valid input to XGMAC
tx_axis_tuser	IN	AXI4-Stream User signal used to signal explicit underrun
tx_ifg_delay[7:0]	IN	Configures Interframe Gap adjustment between packets.
tx_axis_tlast	IN	AXI4-Stream signal to XGMAC indicating End of Ethernet Packet
tx_axis_tready	OUT	AXI4-Stream acknowledge signal from XGMAC to indicate the start of a Data transfer.

AXI4-Stream Interface - Receive

The signals of the AXI4-Stream interface are shown in Table 3-2. See Chapter 6, Interfacing to the Core: Data Interfaces for details on connecting to the receive interface.

Table 3-2: **AXI4_Stream Interface Ports - Receive**

Name	Direction	Description
rx_axis_aresetn	IN	AXI4-Stream active-Low reset for Receive path XGMAC
rx_axis_tdata	OUT	AXI4-Stream data from XGMAC to upper layer
rx_axis_tkeep	OUT	AXI4-Stream data control from XGMAC to upper layer
rx_axis_tvalid	OUT	AXI4-Stream Data Valid from XGMAC
rx_axis_tuser	OUT	AXI4-Stream User signal from XGMAC 1 indicates that a good packet has been received. 0 indicates that a bad packet has been received.
rx_axis_tlast	OUT	AXI4-Stream signal from XGMAC indicating the end of a packet

Flow Control Interface

The flow control interface is used to initiate the transmission of flow control frames from the core. The ports associated with this interface are shown in Table 3-3.

Table 3-3: **Flow Control Interface Ports**

Name	Direction	Description
pause_req	IN	Request that a flow control frame is emitted from the MAC core.
pause_val[15:0]	IN	Pause value field for flow control frame to be sent when pause_req asserted.

32-bit XGMII PHY Interface or 64-bit SDR PHY Interface

This interface is used to connect to the physical layer, whether this is a separate device or implemented in the FPGA beside the MAC core. Table 3-4 shows the ports associated with this interface. The PHY interface can be a 32-bit DDR XGMII interface a or 64-bit SDR interface, depending on the customization of the core. However, the netlist ports are always the same width and the translation between 32-bit and 64-bit is performed in the HDL wrapper, if required.

Table 3-4: **PHY Interface Port Descriptions**

Name	Direction	Description
xgmii_txd[63:0]	OUT	Transmit data to PHY
xgmii_txc[7:0]	OUT	Transmit control to PHY
xgmii_rxd[63:0]	IN	Received data from PHY
xgmii_rxc[7:0]	IN	Received control from PHY

Management Interface

Configuration of the core, access to the statistics block, access to the MDIO port, and access to the interrupt block can be provided through the Management Interface, a 32-bit AXI4-Lite interface independent of the Ethernet datapath. Table 3-5 defines the ports associated with the Management Interface.

Table 3-5: **Management Interface Port Descriptions**

Name	Direction	Description
s_axi_aclk	IN	AXI4-Lite clock. Range between 10 MHz and 156.25 MHz
s_axi_aresetn	IN	Asynchronous active-Low reset
s_axi_awaddr[31:0]	IN	Write address Bus
s_axi_awvalid	IN	Write address valid
s_axi_awready	OUT	Write address acknowledge
s_axi_wdata[31:0]	IN	Write data bus
s_axi_wvalid	OUT	Write data valid
s_axi_wready	OUT	Write data acknowledge
s_axi_bresp[1:0]	OUT	Write transaction response
s_axi_bvalid	OUT	Write response valid
s_axi_bready	IN	Write response acknowledge
s_axi_araddr[31:0]	IN	Read address Bus
s_axi_arvalid	IN	Read address valid
s_axi_arready	OUT	Read address acknowledge
s_axi_rdata[31:0]	OUT	Read data output
s_axi_rresp[1:0]	OUT	Read data response
s_axi_rvalid	OUT	Read data/response valid
s_axi_rready	IN	Read data acknowledge

The Management Interface can be omitted at core customization stage; if omitted, `configuration_vector_tx/rx` is available instead.

Configuration and Status Signals

If the Management Interface is omitted at core customization time, configuration and status vectors are exposed by the core. This allows you to configure the core by statically or dynamically driving the constituent bits of the port. Table 3-6 describes the configuration and Status signals. See Chapter 7, Interfacing to the Core: Management Interface for details on this signal, including a breakdown of the configuration and status vector bits.

Table 3-6: **Configuration and Status Signals**

Name	Direction	Description
tx_configuration_vector[79:0]	Input	Configuration signals for the Transmitter
rx_configuration_vector[79:0]	Input	Configuration signals for the Receiver
status_vector[1:0]	Output	Status signals for the core

MDIO Interface

The MDIO Interface signals are shown in Table 3-7. See Chapter 7, Interfacing to the Core: Management Interface for details on the use of this interface.

Table 3-7: **MDIO Interface Port Descriptions**

Name	Direction	Description
mdc	Output	MDIO clock
mdio_in	Input	MDIO input
mdio_out	Output	MDIO output
mdio_tri	Output	MDIO 3-state. 1 disconnects the output driver from the MDIO bus.

Statistic Vectors

In addition to the statistic counters described in Management Interface in Chapter 3, there are two statistics vector outputs on the core netlist that are used to signal the core state. These vectors are actually used as the inputs of the counter logic internal to the core; so if you omit the statistic counters at the CORE Generator™ tool customization stage, a relevant subset can be implemented in user logic. The signals are shown in Table 3-8. The contents of the vectors themselves are described in Chapter 7, Interfacing to the Core: Management Interface.

Table 3-8: **Statistic Vector Signals**

Name	Direction	Description
tx_statistics_vector[25:0]	Output	Aggregated statistics flags for transmitted frame.
tx_statistics_valid	Output	Valid strobe for tx_statistics_vector.
rx_statistics_vector[29:0]	Output	Aggregated statistics flags for received frames.
rx_statistics_valid	Output	Valid strobe for rx_statistics_vector.

Clocking and Reset Signals and Module

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

Table 3-9 shows the ports on the netlist associated with system clocks and resets.

Table 3-9: **Clock, Clock Management, and Reset Ports**

Name	Direction	Description
tx_clk0	Input	System clock for transmit side of core; derived from gtx_clk in example design
tx_dcm_lock	Input	Status flag from DCM/MMCM
rx_clk0	Input	System clock for receive side of core; derived from xgmii_rx_clk in example design
rx_dcm_lock	Input	Status flag from DCM/MMCM

Chapter 4

Customizing and Generating Core

The 10-Gigabit Ethernet MAC core is generated through the Xilinx® CORE Generator™ tool. This chapter describes how to customize the 10-Gigabit Ethernet MAC core and generate the core netlist.

GUI Interface

Figure 4-1 displays the CORE Generator tool customization screen for the 10-Gigabit Ethernet MAC core.

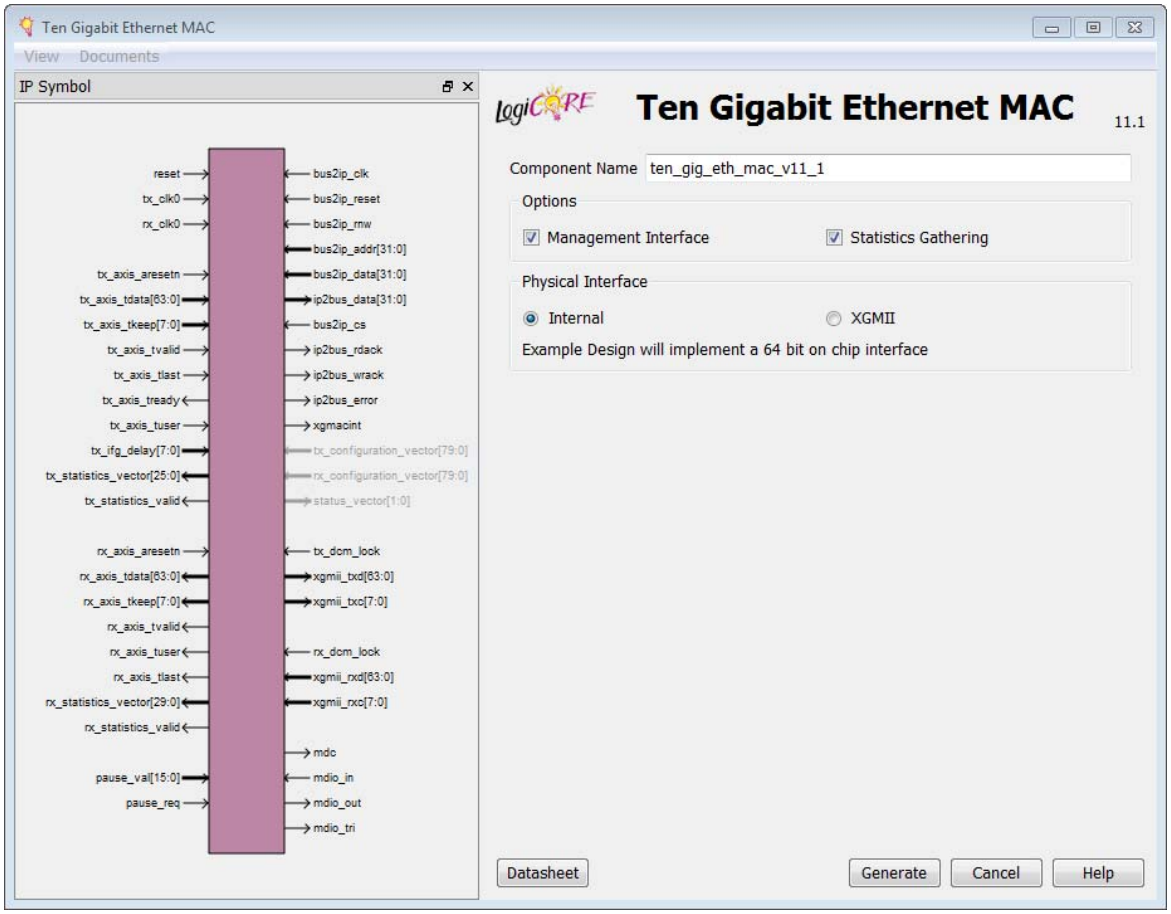

Figure 4-1: **10-Gigabit Ethernet MAC Customization Screen**

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

Component Name

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

Statistics Gathering

This checkbox selects whether the statistics counters are included in the generated core.

This option is only available if the Management Interface option is selected. The default is to have statistics counters included.

Management Interface

Select this option to include the Management Interface in the generated core. Deselect this option to remove the Management Interface and expose a simple bit vector to manage the core.

The default is to have the Management Interface included.

Physical Interface

The physical interface section has a choice of two selections; *XGMII*, which implements the 32-bit DDR interface to the physical layer, and *Internal*, which selects the internal 64-bit SDR interface to the physical layer.

Note: On Spartan®-6 devices, only the internal 64-bit interface is supported.

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 can 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 Ten_Gigabit_Ethernet_MAC Virtex6 Xilinx,_Inc. 11.1
CSET physical_interface = XGMII
CSET statistics_gathering = true
CSET component_name = the_core
CSET management_interface = true
GENERATE
```

Table 4-1: **XCS File Values and Defaults**

Parameter	XCO File Values	Defaults
component_name	ASCII text starting with a letter and based on the following character set: a..z, 0..9 and _	Blank
physical_interface	XGMII, Internal	Internal
statistics_gathering	True, false	True
management_interface	True, false	True

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 Chapter 13, Detailed Example Design for a complete description of the CORE Generator tool output files and for details of the HDL example design.

XILINX®

Chapter 5

Designing with the Core

This chapter contains a general description of how to use the 10-Gigabit Ethernet MAC core in your own design. It should be read in conjunction with Chapter 6, Interfacing to the Core: Data Interfaces and Chapter 7, Interfacing to the Core: Management Interface which describe particular interfaces of the core.

General Design Guidelines

This section describes the steps required to turn a 10-Gigabit Ethernet MAC 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

Every instance of the 10-Gigabit Ethernet MAC core created by the Xilinx® CORE Generator™ tool is delivered with an example design that can be implemented in an FPGA and simulated. This design can be used as a starting point for your own design or can be used to sanity-check your application in the event of difficulty.

Consult Chapter 12, Quick Start Example Design and Chapter 13, Detailed Example Design for information on using and customizing the example designs for the 10-Gigabit Ethernet MAC core.

Know the Degree of Difficulty

10-Gigabit Ethernet designs are challenging to implement in any technology. The degree of difficulty is sharply influenced by:

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

All 10-Gigabit Ethernet 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 might not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx tools to place and route the design.

Recognize Timing Critical Signals

The UCF provided with the example design for the core identifies the critical signals and the timing constraints that should be applied. See Chapter 9, Constraining the Core for further information.

Use Supported Design Flows

The core is synthesized in the Xilinx CORE Generator tool and is delivered to you as an NGC netlist. The example implementation scripts provided currently use XST 14.1 as the synthesis tool for the HDL example design that is delivered with the core. Other synthesis tools can be used for the user application logic; the core is always unknown to the synthesis tool and should appear as a black box.

Post-synthesis, only ISE® Design Suite v14.1 tools are supported.

Make Only Allowed Modifications

The 10-Gigabit Ethernet MAC core is not user-modifiable. Do not make modifications as they can have adverse effects on system timing and protocol compliance. Supported user configurations of the 10-Gigabit Ethernet MAC 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 Core.

XILINX®

Chapter 6

Interfacing to the Core: Data Interfaces

This chapter describes how to connect to the data interfaces of the 10-Gigabit Ethernet MAC core.

Interfacing to the Transmit AXI4-Stream Interface

AXI-4 Stream Interface: Transmit

The client-side interface on the transmit side of XGMAC supports an AXI4-Stream interface. It has a 64-bit datapath with eight control bits to delineate bytes within the 64-bit port. Additionally, there are signals to handshake the transfer of data into the core. An example design which includes source code for a FIFO with an AXI4-Stream interface is provided with the core generated by Xilinx® CORE Generator™ tool. Table 6-1 defines the signals.

Table 6-1: **Transmit Client-Side Interface Port Description**

Name	Direction	Description
tx_axis_aresetn	IN	AXI4-Stream active-Low reset for Transmit path XGMAC
tx_axis_tdata[63:0]	IN	AXI4-Stream data to XGMAC
tx_axis_tkeep[7:0]	IN	AXI4-Stream Data Control to XGMAC
tx_axis_tvalid	IN	AXI4-Stream Data Valid input to XGMAC
tx_axis_tuser	IN	AXI4-Stream user signal used to indicate explicit underrun
tx_axis_tlast	IN	AXI4-Stream signal to XGMAC indicating End of Ethernet Packet
tx_axis_tready	IN	AXI4-Stream acknowledge signal from XGMAC to indicate to start the Data transfer
tx_ifg_delay[7:0]	IN	Configures Interframe Gap adjustment between packets.

For transmit data `tx_axis_tdata[63:0]` (Table 6-2), the port is logically divided into lane 0 to lane 7, with the corresponding bit of the `tx_axis_tkeep` word signifying valid data on `tx_axis_tdata`.

Table 6-2: **tx_axis_tdata Lanes**

Lane/tx_axis_tkeep bit	tx_axis_tdata 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

Normal Frame Transmission

The timing of a normal frame transfer is shown in Figure 6-1. When the client wants to transmit a frame, it asserts the `tx_axis_tvalid` and places the data and control in `tx_axis_tdata` and `tx_axis_tkeep` in the same clock cycle. After the core asserts `tx_axis_tready` to acknowledge the first beat of data, on the next and subsequent clock edges, the client must provide the remainder of the data for the frame to the core. The end of packet is indicated to the core by `tx_axis_tlast` asserted for 1 cycle. The bits of `tx_axis_tkeep` are set appropriately if the packet ends at a non- 64 bit boundary. For example, in Figure 6-1, the first packet ends at Lane 3 and any data after that is ignored.

After `tx_axis_tlast` is deasserted, any data and control is deemed invalid until `tx_axis_tvalid` is next asserted.

If custom preamble is enabled, `tx_axis_tready` signal might not be deasserted at the end of the frame, in order to read the custom preamble into the core for the following frame.

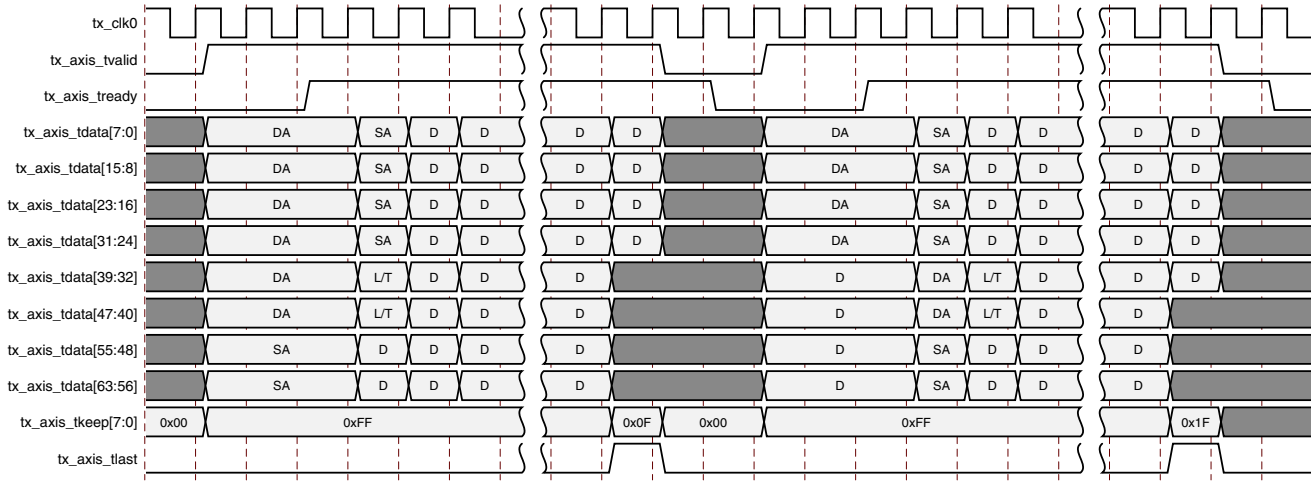

Figure 6-1: **Frame Transmission**

In-Band Ethernet Frame Fields

For maximum flexibility in switching applications, the Ethernet frame parameters (destination address, source address, length/type and optionally FCS) are encoded within the same data stream that the frame payload is transferred on, rather than on separate ports. This is illustrated in the timing diagrams. The destination address must be supplied with the first byte in lane 0 and so on. Similarly, the first byte of the source address must be supplied in lane 6 of the first transfer.The length/type field is similarly encoded, with the first byte placed into lane 4. The definitions of the abbreviations used in the timing diagrams are described in Table 6-3.

Table 6-3: **Abbreviations Used in Timing Diagrams**

Abbreviation	Definition
DA	Destination address
SA	Source address
L/T	Length/type field
FCS	Frame check sequence (CRC)

Padding

When fewer than 46 bytes of data are supplied by the client to the MAC core, the transmitter module adds padding up to the minimum frame length, unless the MAC core is configured for in-band FCS passing. In the latter case, the client must also supply the padding to maintain the minimum frame length. When in-band FCS is enabled, if the client does not provide a frame with at least 46 bytes of data, the frame is terminated correctly but not padded.

Transmission with In-Band FCS Passing

If the MAC core is configured to have the FCS field passed in by the client on the AXI4-Stream Transmit interface, the transmission timing is as shown in Figure 6-2. In this case, it is the responsibility of the client to ensure that the frame meets the Ethernet minimum frame length requirements; the MAC core does not perform any padding of the payload. If the client transmits a frame less than the Ethernet minimum length requirement, then the frame is not counted in the statistics.

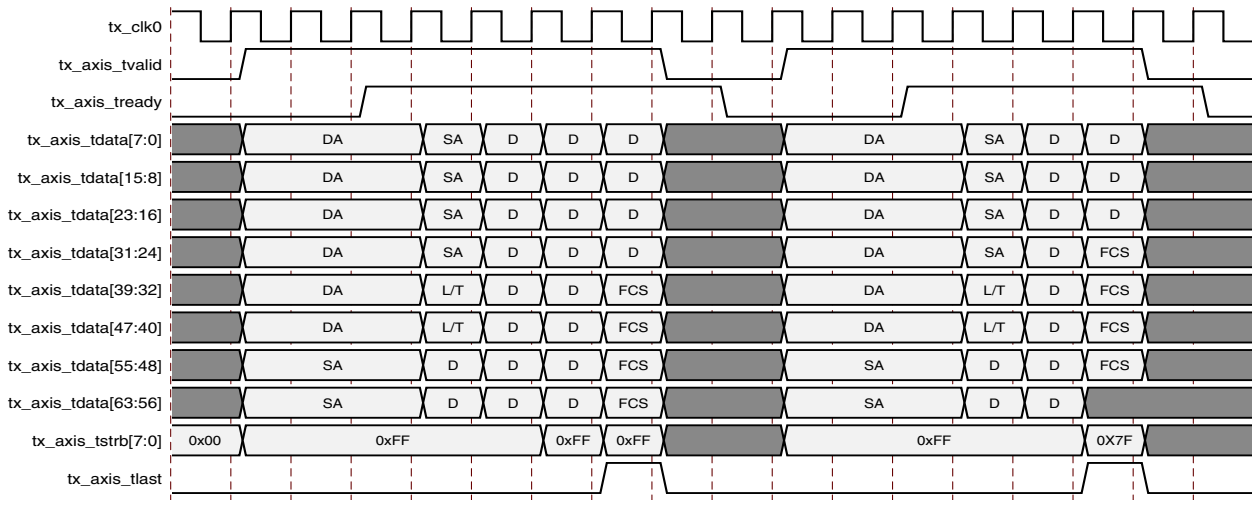

Figure 6-2: **Transmission with In-Band FCS Passing**

Aborting a Transmission

The aborted transfer of a packet on the client interface is called an underrun. This can happen, for instance, if a FIFO in the AXI Transmit client interface empties before a frame is completed. This is indicated to the core in one of two ways:

1. An explicit underrun, in which a Frame Transfer is aborted by asserting `tx_axis_tuser` High while `tx_axis_tvalid` is High and data transfer is continuing. (See Figure 6-3)
   An underrun packet must have the DA, SA, L/T fields in it. This is true even if Custom Preamble is enabled for transmission.
2. An implicit underrun, in which a Frame Transfer is aborted by deasserting `tx_axis_tvalid` without asserting `tx_axis_tlast`. (See Figure 6-4)

Figure 6-3 and Figure 6-4 each show an underrun frame followed by a complete frame. When either of the two scenarios occurs during a frame transmission, the MAC core inserts error codes into the XGMII data stream to flag the current frame as an errored frame and then the MAC core falls back to idle transmission. The `tx_mac_underun` signal shown on the diagram is an internal signal. It remains the responsibility of the client to re-queue the aborted frame for transmission, if necessary.

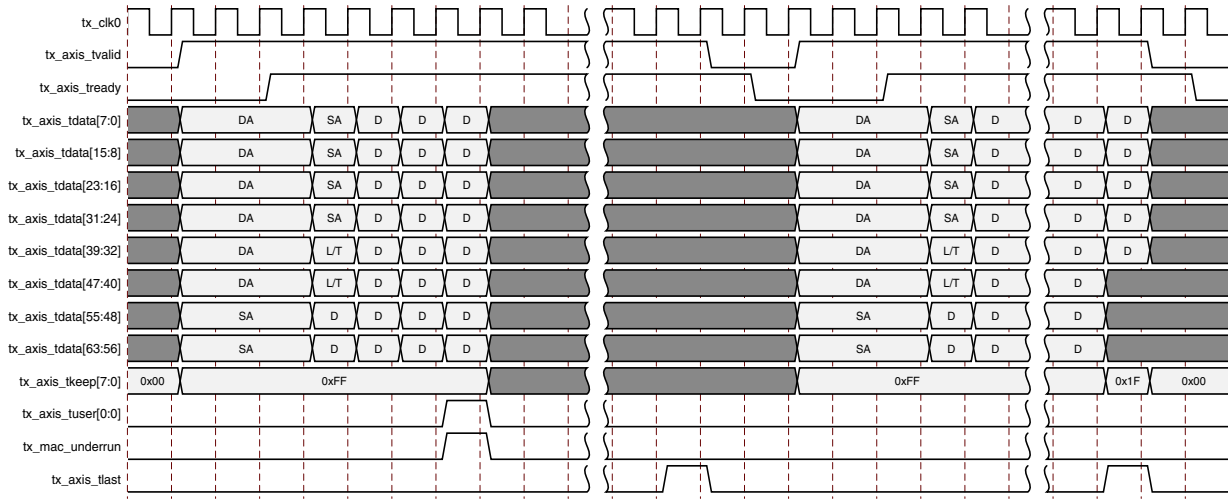

Figure 6-3: **Frame Transfer Abort with tx_axis_tuser asserted**

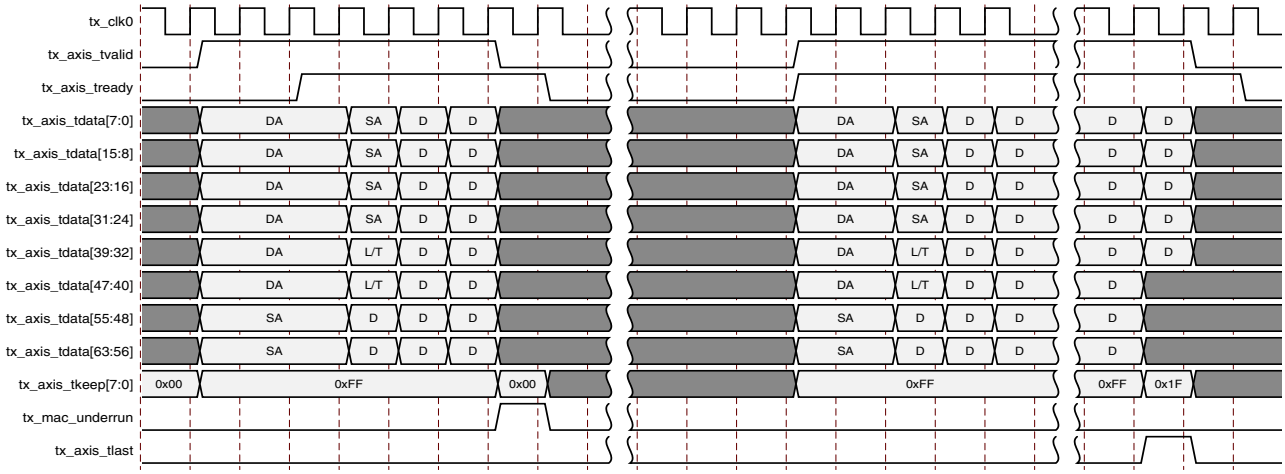

Figure 6-4: **Frame Transfer Abort with tx_axis_tvalid deasserted**

Note: Aborting a frame transfer using the mechanism shown in Figure 6-4 is not fully AXI4-compliant, as no TLAST is asserted to complete the first frame. If AXI4-compliance is important, use the scheme of Figure 6-3.

Back-to-Back Continuous Transfers

Continuous data transfer on Transmit AXI4-Stream interface is possible, as the signal `tx_axis_tvalid` can remain continuously High, with packet boundaries defined solely by `tx_axis_tlast` asserted for the end of the Ethernet packet. However, the MAC core can defer the `tx_axis_tready` acknowledgement signal to comply with the inter-packet gap requirements on the XGMII side of the core.

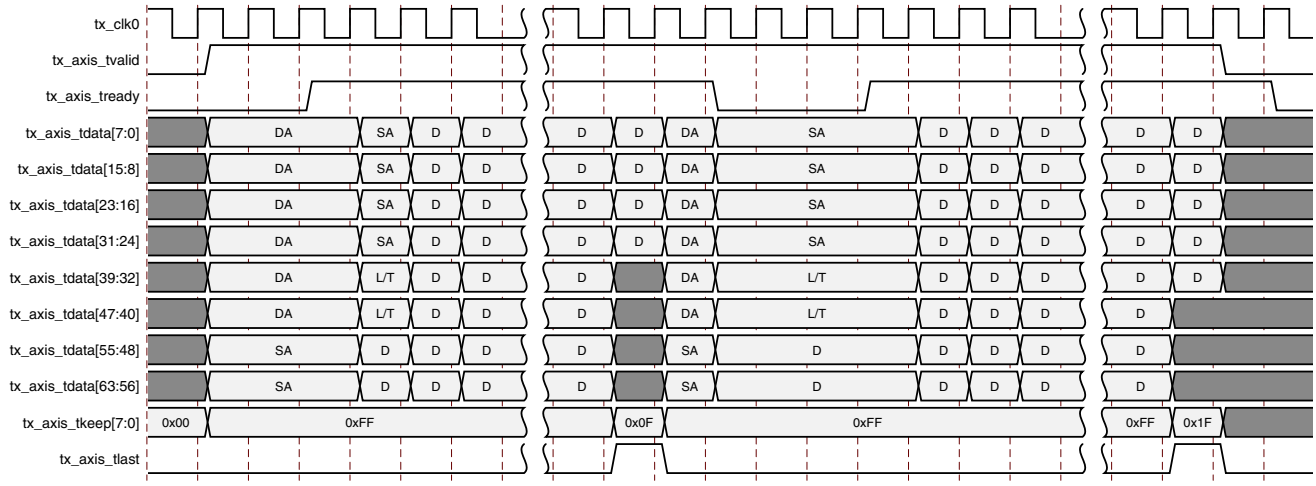

Figure 6-5: **Back-to-Back Continuous Transfer on Transmit Client Interface**

Transmission of Custom Preamble

You can elect to use a custom preamble field. If this function is selected (using a configuration bit, see Configuration Registers in Chapter 7), the standard preamble field can be substituted for custom data. The custom data must be supplied on `tx_axis_tdata[63:8]` in the first column when `tx_axis_tvalid` is first asserted High. Transmission of Custom Preamble can happen in both continuous and non-continuous mode of `tx_axis_tvalid`. Figure 6-6 shows a frame presented at the Transmit Client Interface with a custom preamble where P1 to P7 denote the custom data bytes when `tx_axis_tvalid` is deasserted after `tx_axis_tlast`. Figure 6-7 illustrates the transmission of a custom preamble when `tx_axis_tvalid` remains asserted after `tx_axis_tlast` is asserted.

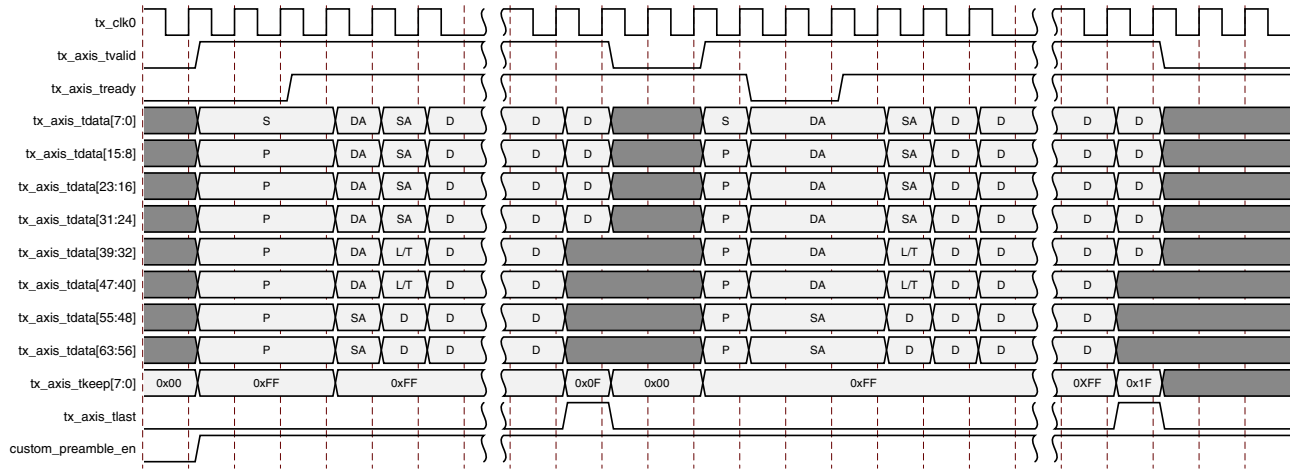

Figure 6-6: **Transmission of Custom Preamble in the Non-Continuous Case**

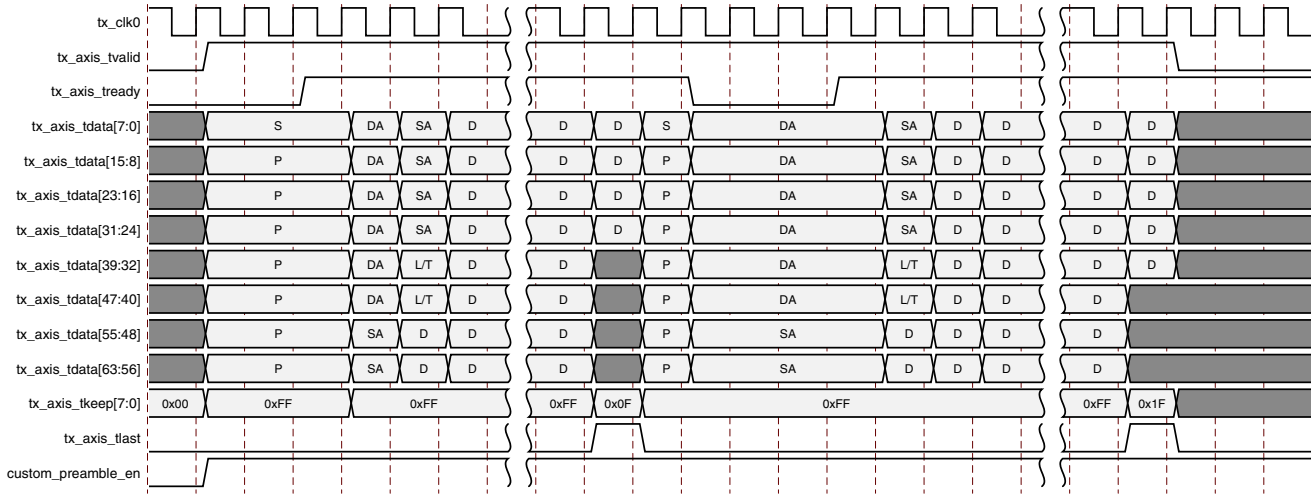

Figure 6-7: **Transmission of Custom Preamble in the Continuous Case**

The MAC core substitutes the IEEE standard preamble with that supplied by the client logic. Figure 6-8 shows the transmission of a frame with custom preamble (P1 to P7) at the XGMII interface.

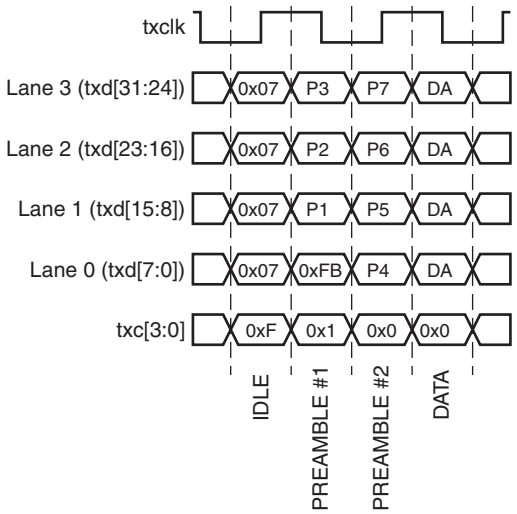

Figure 6-8: **XGMII Frame Transmission of Custom Preamble**

VLAN Tagged Frames

Transmission of a VLAN tagged frame (if enabled) is shown in Figure 6-9. The handshaking signals across the interface do not change; however, the VLAN type tag 81-00 must be supplied by the client to signify that the frame is VLAN tagged. The client also supplies the two bytes of Tag Control Information, V1 and V2, at the appropriate times in the data stream. Additional information about the contents of these two bytes is available in [Ref 6].

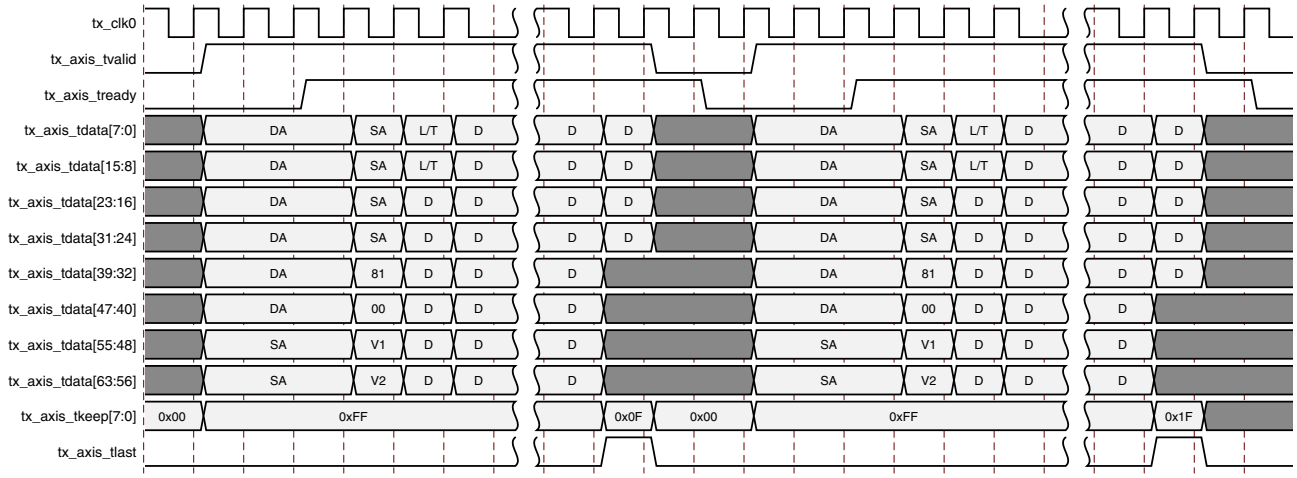

Figure 6-9: **VLAN Tagged Frame Transmission**

Transmitter Maximum Permitted Frame Length

The maximum legal length of a frame specified in [Ref 6] is 1518 bytes for non-VLAN tagged frames. VLAN tagged frames might be extended to 1522 bytes. When jumbo frame handling is disabled and the client attempts to transmit a frame which exceeds the maximum legal length, the MAC core inserts an error code to corrupt the current frame and the frame is truncated to the maximum legal length. When jumbo frame handling is enabled, frames which are longer than the legal maximum are transmitted error free.

If required, a custom Maximum Transmission Unit (MTU) can be programmed into the core. This allows the transmission of frames up to the programmed MTU size by the core, rather than the 1518/1522 byte limit. The programmed MTU must be equal to or greater than 1518 bytes.

Any Frame transmitted greater than the MTU Frame Size, if Jumbo Frame is disabled, is signaled as a bad frame; error codes are inserted and the frame is truncated.

For details on enabling and disabling jumbo frame handling, see Configuration Registers in Chapter 7.

Note: There are interactions between the configuration bits affecting frame length handling that the user should be aware of. Firstly, if Jumbo Enable and MTU Frame Transfer Enable are enabled at the same time, the Jumbo Enable takes precedence. Secondly, if VLAN Enable and MTU Frame Transfer Enable are both turned on, then MTU frame length rules apply.

Interframe Gap Adjustment

You can elect to vary the length of the interframe gap. If this function is selected (using a configuration bit, see Configuration Registers in Chapter 7), the MAC exerts back pressure to delay the transmission of the next frame until the requested number of XGMII columns has elapsed. The number of XGMII columns is controlled by the value on the `tx_ifg_delay` port. The minimum interframe gap of three XGMII columns (12 bytes) is always maintained. Figure 6-10 shows the MAC operating in this mode.

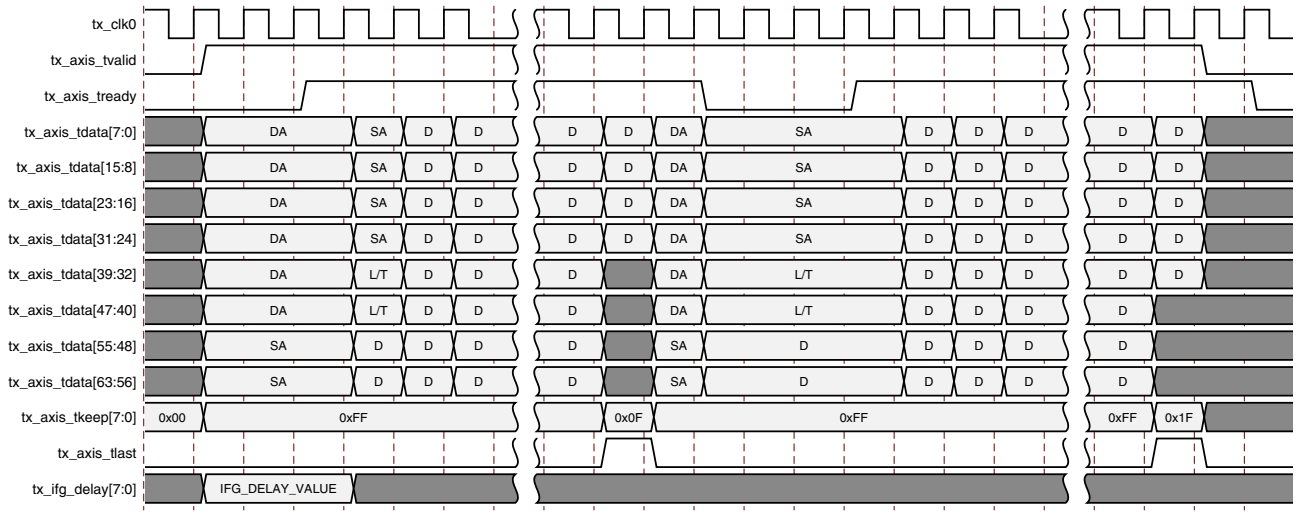

Figure 6-10: **Interframe Gap Adjustment**

Deficit Idle Count (DIC)

The Transmit side XGMAC supports Interframe Gap obtained through Deficit Idle Count to maintain the effective data rate of 10 Gb/s as described in [Ref 6]. This feature is supported even when the AXI4-Stream sends Ethernet packets with In Band FCS or without FCS. It is also supported when Custom Preamble is enabled for transmission. However, the requirement from the Transmit Streaming Interface is that to maintain the effective data rate of 10 Gb/s through the IPG adjustment with DIC, `axis_tx_tvalid` must be maintained continuously High.

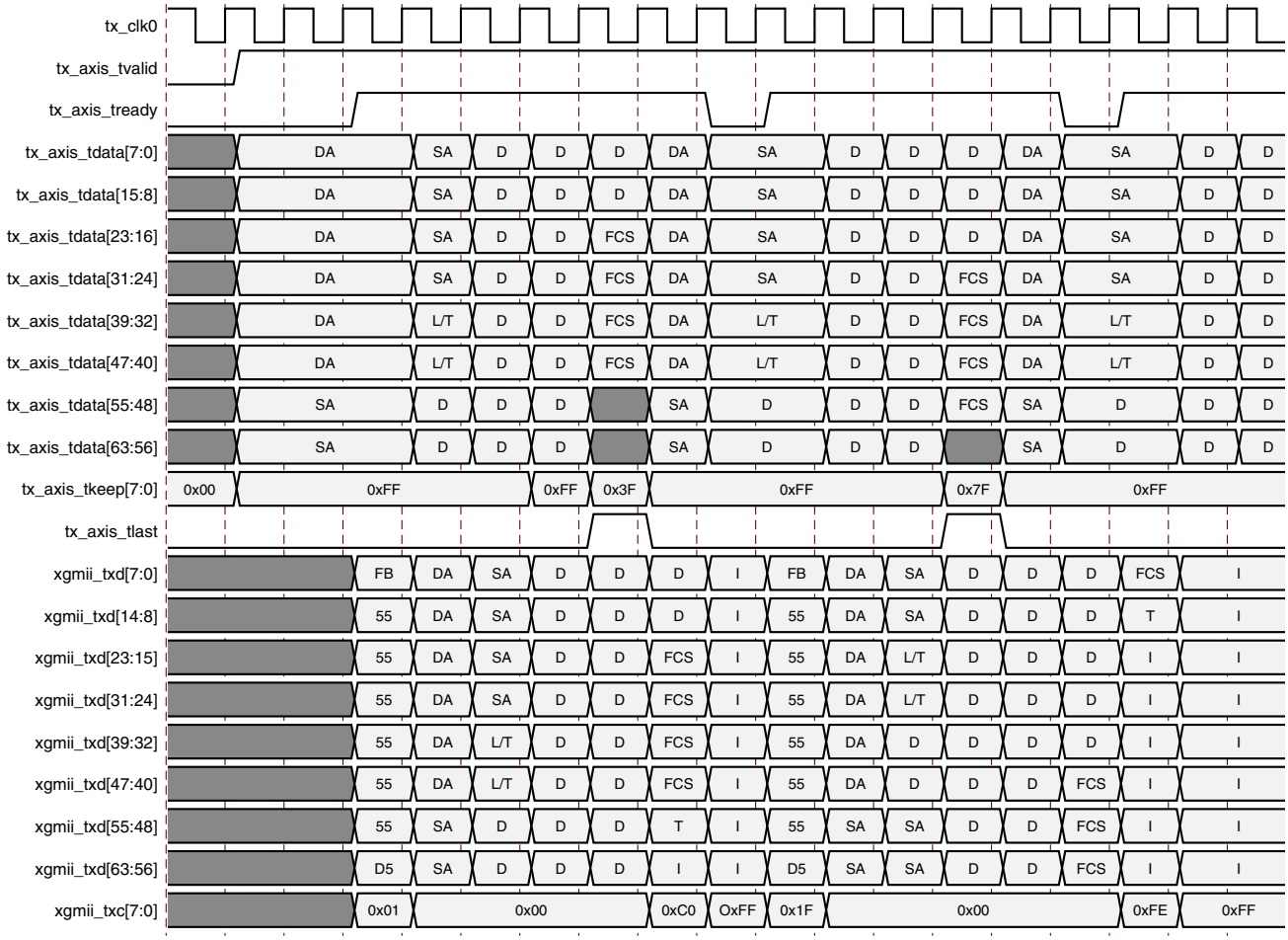

Figure 6-11: **Back-to-Back Continuous Transfer on Transmit XGMII Interface**

Transmission of Frames During Local/Remote Fault Reception

When a local or remote fault has been received, the core might not transmit frames if Fault Inhibit has been disabled (using a configuration bit, see Configuration Registers in Chapter 7). When Fault Inhibit is disabled, the Reconciliation Sublayer transmits ordered sets as presented in [Ref 6]; that is, when the RS is receiving Local Fault ordered sets, it transmits Remote Fault ordered sets. When receiving Remote Fault ordered sets, it transmits idle code words. If the management interface is included with the core, the status of the local and remote fault register bits can be monitored (bits 28 and 29 of the Reconciliation Sublayer configuration word, address 0x300) and when they are both clear, the core is ready to accept frames for transmission. If the management interface is not

included with the core, the status of the local and remote fault register bits can be monitored on bits 0 and 1 of the status vector.

Note: Any frames presented at the client interface prior to both register bits being clear are dropped silently by the core.

When Fault Inhibit mode is enabled, the core transmits data normally regardless of received Local Fault or Remote Fault ordered sets.

Interfacing to the Receive AXI4-Stream Interface

Normal Frame Reception

The client-side interface on receive side of XGMAC supports the AXI4-Stream interface. It has a 64-bit datapath with eight control bits to delineate bytes within the 64-bit port. Additionally, there are signals to indicate to the user logic the validity of the previous frame received. Table 6-4 defines the signals.

Table 6-4: **Receive Client-Side Interface Port Description**

Name	Direction	Description
rx_axis_aresetn	IN	AXI4-Stream active-Low reset for Receive path XGMAC
rx_axis_tdata	OUT	AXI4-Stream Data from XGMAC to upper layer
rx_axis_tkeep	OUT	AXI4-Stream Data Control from XGMAC to upper layer
rx_axis_tvalid	OUT	AXI4-Stream Data Valid from XGMAC
rx_axis_tuser	OUT	AXI4-Stream User Sideband Interface from XGMAC 1 indicates a good packet has been received. 0 indicates a bad packet has been received.
rx_axis_tlast	OUT	AXI4-Stream signal from XGMAC indicating an end of packet

For the receive data port `rx_axis_tdata[63:0]` (Table 6-5), the port is logically divided into lane 0 to lane 7, with the corresponding bit of the `rx_axis_tkeep` word signifying valid data on the `rx_axis_tdata`.

Table 6-5: **rx_axis_tdata Lanes**

Lane/rx_axis_tkeep bit	rx_axis_tdata bits
0	7:0
1	15:8
2	23:16
3	31:24
4	39:32
5	47:40
6	55:58
7	63:56

The timing of a normal inbound frame transfer is represented in Figure 6-12. The client must be prepared to accept data at any time; there is no buffering within the MAC to allow for latency in the receive client. When frame reception begins, data is transferred on consecutive clock cycles to the receive client. The `rx_axis_mac_tlast` and `rx_axis_mac_tuser` signals are asserted, along with the final bytes of the transfer, only after all frame checks are completed. This is after the FCS field has been received.'The MAC asserts the `rx_axis_tuser` signal to indicate that the frame was successfully received and that the frame should be analyzed by the client. This is also the end of packet signaled by `tx_axis_tlast` asserted for 1 cycle.

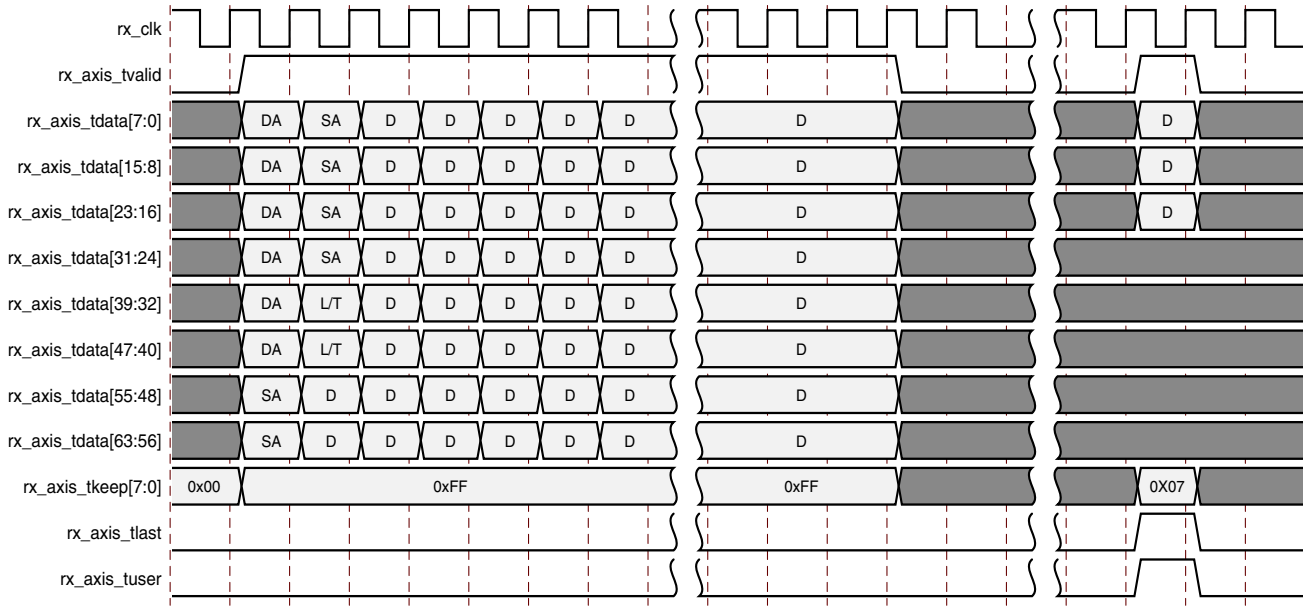

Figure 6-12: **Reception of a Good Frame**

Timing for a Good or a Bad Frame

As can be seen in the Figure 6-12, there is always a gap of up to seven clocks, by which the indication of a good frame or a bad frame is signaled through `rx_axis_tuser` set to '1' or a '0' and `rx_axis_tlast` asserted. This status is only indicated when all frame checks are completed. This can be up to seven clock cycles after the last valid data is presented when the Length/Type field in the frame is valid; for example, this can result from padding at the end of the Ethernet frame. If the Length/Type field in the frame is incorrect and the frame is longer than indicated, then it is a bad frame and hence `rx_axis_tlast` might be deasserted significantly earlier than seven clock cycles after the end of valid data.Although good frame reception is illustrated, the same timing applies to a bad frame. Either the good frame or bad frame signaled through `rx_axis_tuser` and `rx_axis_tlast` is, however, always asserted before the next frame data begins to appear on `rx_axis_tdata`.

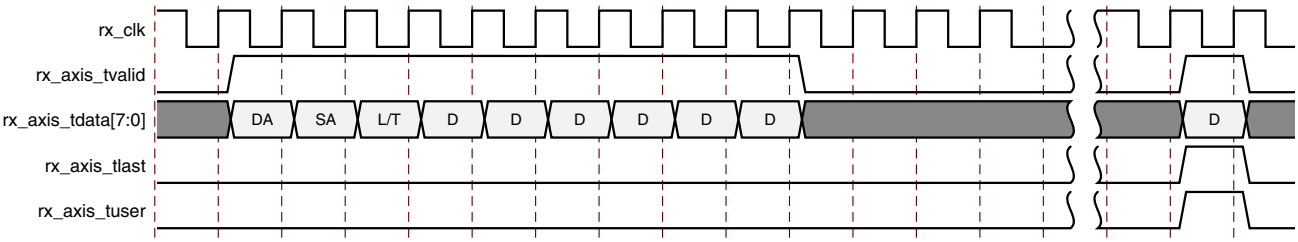

Figure 6-13: **Timing of a Good Frame**

Frame Reception with Errors

The case of an unsuccessful frame reception (for example, a runt frame or a frame with an incorrect FCS) can be seen in Figure 6-14. In this case, the bad frame is received and the signal `rx_axis_tuser` is deasserted to the client at the end of the frame. It is then the responsibility of the client to drop the data already transferred for this frame.

The following conditions cause the assertion of `rx_axis_tlast` along with `rx_axis_tuser` = '0' signifying a bad_frame:

- FCS errors occur.
- Packets are shorter than 64 bytes (undersize or fragment frames).
- Jumbo frames are received when jumbo frames are not enabled.
- Frames of length greater than the MTU Size programmed are received, MTU Size Enable Frames are enabled, and jumbo frames are not enabled.
- The length/type field is length, but the real length of the received frame does not match the value in the length/type field (when length/type checking is enabled).
- The length/type field is length, in which the length value is less than 46. In this situation, the frame should be padded to minimum length. If it is not padded to exactly minimum frame length, the frame is marked as bad (when length/type checking is enabled).
- Any control frame that is received is not exactly the minimum frame length unless Control Frame Length Check Disable is set.
- The XGMII data stream contains error codes.
- A valid pause frame, addressed to the MAC, is received when flow control is enabled.

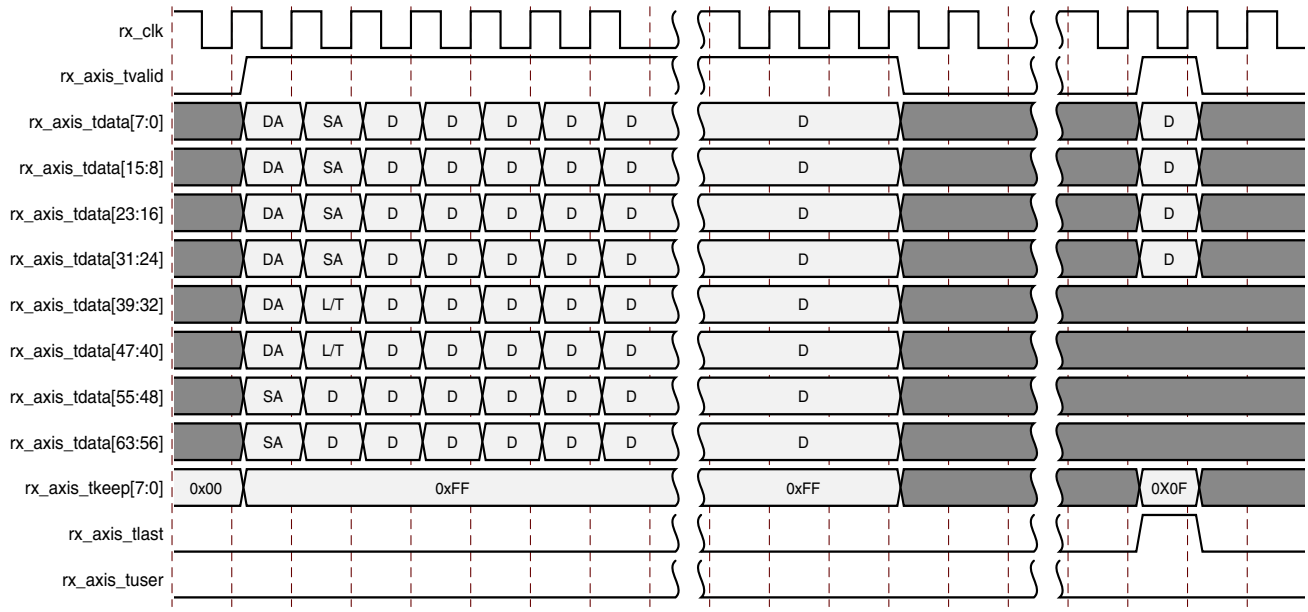

Figure 6-14: **Frame Reception with Error**

Reception with In-Band FCS Passing

Figure 6-15 illustrates the MAC Core configured to pass the FCS field to the client (see Configuration Registers in Chapter 7). In this case, any padding inserted into the frame to meet the Ethernet minimum frame length specifications is left intact and passed to the client. Although the FCS is passed up to the client, it is also verified by the MAC core, and `rx_axis_tuser` is '0' when `rx_axis_tlast` is asserted if the FCS check fails.

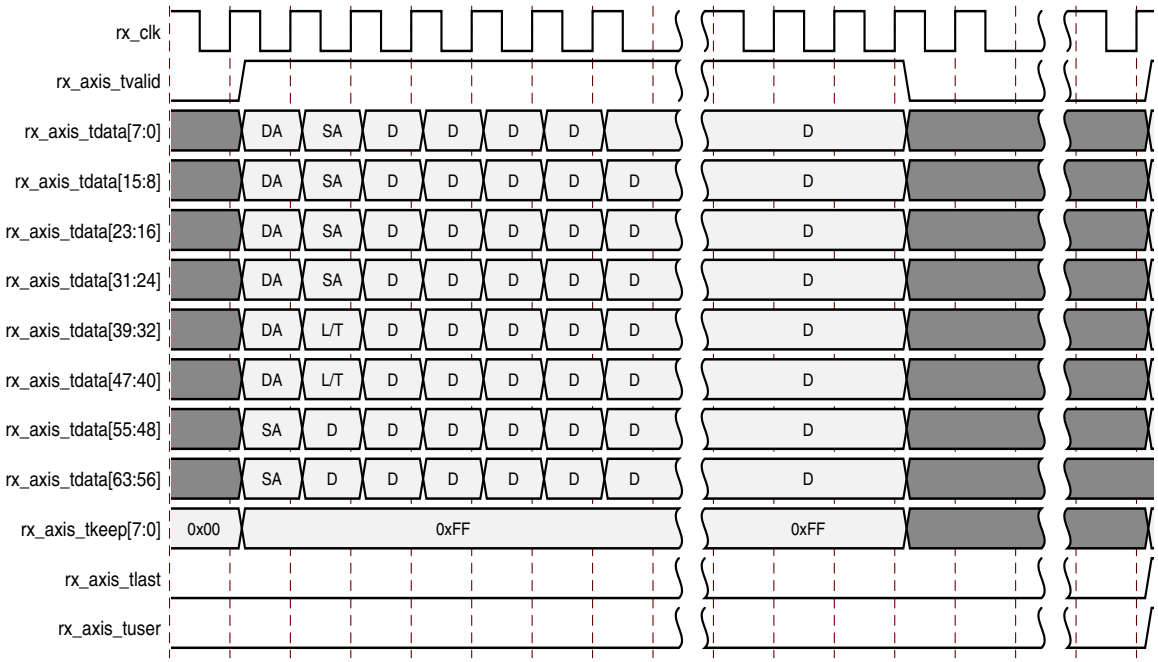

Figure 6-15: **Frame Reception with In-Band FCS Passing**

Reception of Custom Preamble

You can elect to use a custom preamble field. If this function is selected (using a configuration bit, see Configuration Registers in Chapter 7), the preamble field can be recovered from the received data and presented on the Client AXI4-Stream receive Interface. If this mode is enabled, the custom preamble data is present on `rx_axis_tdata[63:8]`. The `rx_axis_tkeep` output is asserted to frame the custom preamble. Figure 6-16 shows the reception of a frame with custom preamble.

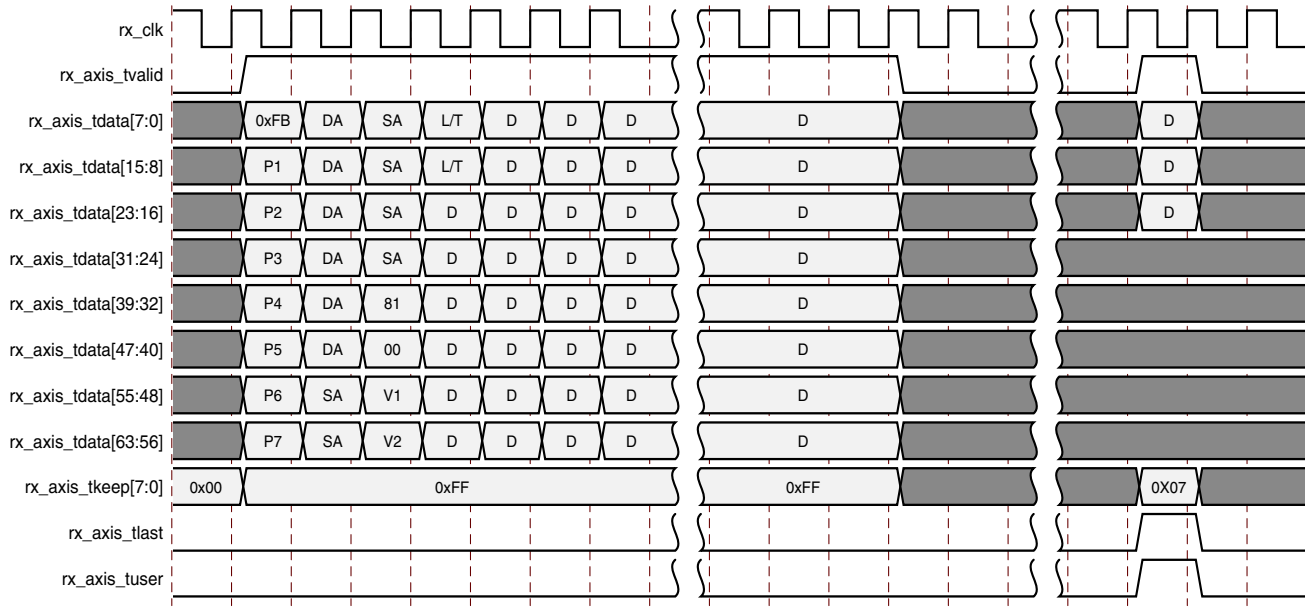

Figure 6-16: **Frame Reception with Custom Preamble**

VLAN Tagged Frames

The reception of a VLAN tagged frame (if enabled) is represented in Figure 6-17. The VLAN frame is passed to the client so that the frame can be identified as VLAN tagged; this is followed by the Tag Control Information bytes, V1 and V2. More information on the interpretation of these bytes can be found in [Ref 6]. All VLAN tagged frames are treated as Type frames, that is, any padding is treated as valid and passed to the client.

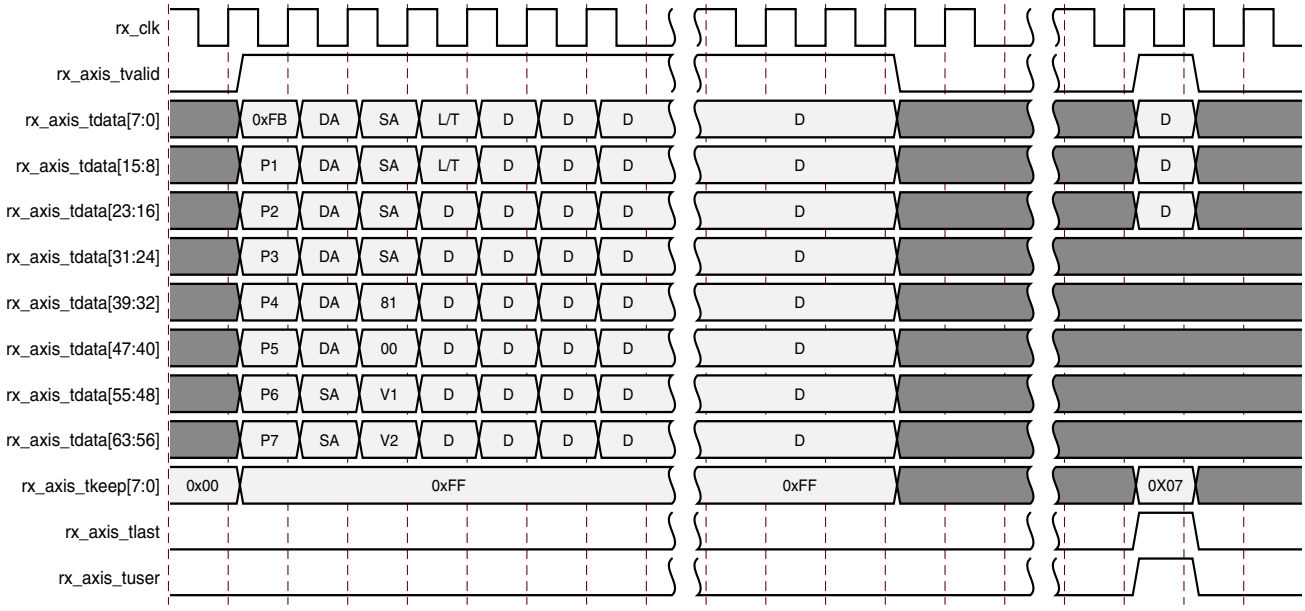

Figure 6-17: **Frame Reception with VLAN Tagged Frames**

Receiver Maximum Permitted Frame Length

The maximum legal length of a frame specified in [Ref 6] is 1518 bytes for non- VLAN tagged frames. VLAN tagged frames might be extended to 1522 bytes. When jumbo frame handling is disabled and the core receives a frame which exceeds the maximum legal length, a bad frame is indicated by `rx_axis_tuser` being '0' when `rx_axis_tlast` is asserted. When jumbo frame handling is enabled, frames which are longer than the legal maximum are received in the same way as shorter frames.

If required, a custom Maximum Transmission Unit (MTU) can be programmed into the core. This allows the reception of frames up to the programmed MTU size by the core, rather than the 1518/1522 byte limit. The programmed MTU must be equal to or greater than 1518 bytes.

Any Frame received greater than the MTU Frame Size, if Jumbo Frame is disabled is signaled as a bad frame.

For details on enabling and disabling jumbo Frame handling and MTU Frame handling, see Configuration Registers in Chapter 7.

Length/Type Field Error Checks

Enabled

Default operation is with the length/type error checking enabled (see Configuration Registers in Chapter 7). In this mode the following checks are made on all received frames. If either of these checks fail, the frame is marked as bad.

- A value in the length/type field which is greater than or equal to decimal 46, but less than 1536, is checked against the actual data length received.
- A value in the length/type field that is less than decimal 46, (a length interpretation), the frame data length is checked to see if it has been padded to exactly 46 bytes (so that the resultant total frame length is 64 bytes).

Furthermore, if padding is indicated (the length/type field is less than decimal 46) and client-supplied FCS passing is disabled, the length value in the length/type field is used to deassert `rx_axis_tkeep[]` after the indicated number of data bytes so that the padding bytes are removed from the frame. SeeReception with In-Band FCS Passing.

Disabled

When the length/type error checking is disabled and the length/type field has a length interpretation, the MAC does not check the length value against the actual data length received as detailed previously. A frame containing only this error is marked as good. However, if the length/type field is less than decimal 46 then the MAC marks a frame as bad if it is not the minimum frame size of 64 bytes.

If padding is indicated and client-supplied FCS passing is disabled, then a length value in the length/type field is not used to deassert `rx_axis_tkeep[]`. Instead, `rx_axis_tkeep[]` is deasserted before the start of the FCS field, and any padding is not removed from the frame.

Sending and Receiving Flow Control Frames

The flow control block is designed to clause 31 of the *IEEE 802.3-2008* standard [Ref 6]. See Overview of Flow Control in Chapter 8 for a description of Flow Control. The MAC can be configured to send pause frames and to act on their reception. These two behaviors can be configured asymmetrically; see Configuration Registers in Chapter 7.

Transmitting a Pause Frame

The client sends a flow control frame by asserting `pause_req` while the pause value is on the `pause_val` bus. These signals are synchronous with respect to `tx_clk0`. The timing of this can be seen in Figure 6-18.

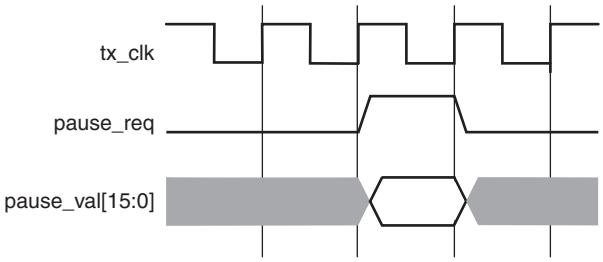

Figure 6-18: **Transmitting a Pause Frame**

If the MAC core is configured to support transmit flow control, this action causes the MAC core to transmit a pause control frame on the link, with the pause parameter set to the value on `pause_val` in the cycle when `pause_req` was asserted. This does not disrupt any frame transmission in progress but does take priority over any pending frame transmission. This frame is transmitted even if the transmitter is in the paused state itself.

Receiving a Pause Frame

When an error-free frame is received by the MAC core, these checks are made:

- The destination address field is matched against the MAC control multicast address or the configured source address for the MAC (see Configuration Registers in Chapter 7).
- The length/type field is matched against the MAC Control type indication, 88-08
- The opcode field contents are matched against the Pause opcode

If any of these checks are false or MAC receiver flow control is disabled, the frame is ignored by the flow control logic and passed up to the client.

If the frame passes all of these checks, is of minimum legal size, and MAC receiver flow control is enabled, the pause value parameter in the frame is used to inhibit transmitter operation for the time defined in the Ethernet specification. This inhibit is implemented using the same back pressure scheme shown in Figure 6-5. Because the received pause frame has been acted on, it is passed to the client with `rx_bad_frame` asserted to indicate that it should be dropped. If Simplex Split with Receive Only is implemented then the pause frame is dropped without being acted on.

Reception of any frame for which the length/type field is the MAC Control type indication 88-08 but is not the legal minimum length is considered an invalid Control frame. It is ignored by the flow control logic and passed to the client with `rx_bad_frame` asserted.

The PHY-Side Interface

External XGMII versus Internal 64-bit Interfaces

At customization time, you have the choice of selecting a 32-bit DDR XGMII PHY Interface or No Interface, which is a 64-bit SDR interface intended for internal connection. In either case, the core netlist is the same; only the example design changes, with the I/O registers and associated constraints targeting DDR or SDR operation respectively.

It is important to remember that although a frame to be transmitted should always be presented to the MAC core with the start in lane 0; due to internal realignment the start of the frame /S/ codeword might appear on the PHY side interface in lane 0 or lane 4. Likewise, the /S/ codeword might legally arrive at the RX PHY interface in either lane 0 or lane 4, but the MAC always presents it to the client logic with start of frame in lane 0.

XILINX®

Chapter 7

Interfacing to the Core: Management Interface

This chapter describes the interfaces available for dynamically setting and querying the configuration and status of the 10-Gigabit Ethernet MAC core. There are two interfaces available for configuration; depending on the core customization, only one is available in a particular core instance.

In addition, the statistics counters and vectors are described in this chapter as well as the use of the MDIO interface.

The Management Interface

The Management Interface is an AXI4-Lite Interface.

This interface is used for:

- Configuring the MAC core
- Configuring the interrupts
- Accessing statistics information for use by high layers, for example, SNMP
- Providing access through the MDIO interface to the management registers located in the PHY attached to the MAC core

The ports of the Management Interface are shown in Table 7-1.

Table 7-1: **Management Interface Port Description**

Name	Direction	Description
s_axi_aclk	IN	AXI4-Lite clock. Range between 10 MHz and 156.25 MHz
s_axi_aresetn	IN	Asynchronous active-Low reset
s_axi_awaddr[31:0]	IN	Write address Bus
s_axi_awvalid	IN	Write address valid
s_axi_awready	OUT	Write address acknowledge
s_axi_wdata[31:0]	IN	Write data bus
s_axi_wvalid	IN	Write data valid
s_axi_wready	OUT	Write data acknowledge
s_axi_bresp[1:0]	OUT	Write transaction response. A value of b00 indicates OKAY and a value of b10 indicates SLVERR.

Table 7-1: **Management Interface Port Description *(Cont'd)***

Name	Direction	Description
s_axi_bvalid	OUT	Write response valid
s_axi_bready	IN	Write response acknowledge
s_axi_araddr[31:0]	IN	Read address Bus
s_axi_arvalid	IN	Read address valid
s_axi_arready	OUT	Read address acknowledge
s_axi_rdata[31:0]	OUT	Read data output
s_axi_rresp[1:0]	OUT	Read data response. A value of b00 indicates OKAY and a value of b10 indicates SLVERR.
s_axi_rvalid	OUT	Read data/response valid
s_axi_rready	IN	Read data acknowledge

Note: Only 11[10:0] out of the 32 address bits are used for decoding the read/write address.

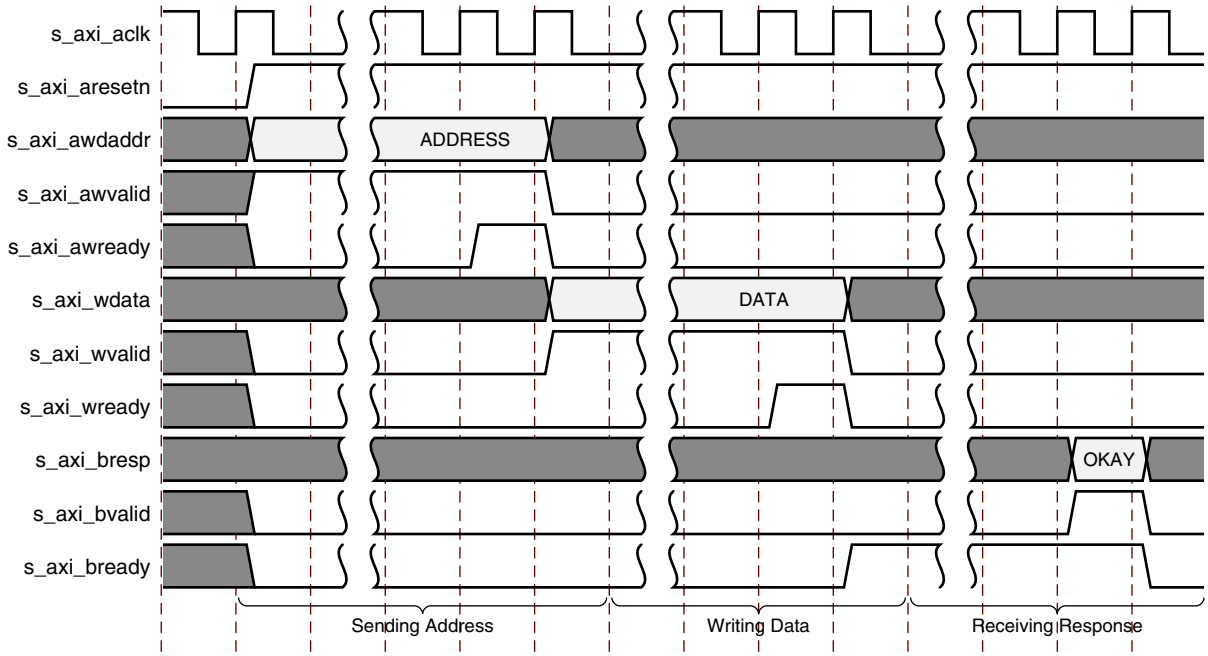

Figure 7-1: **Management Register Write Timing**

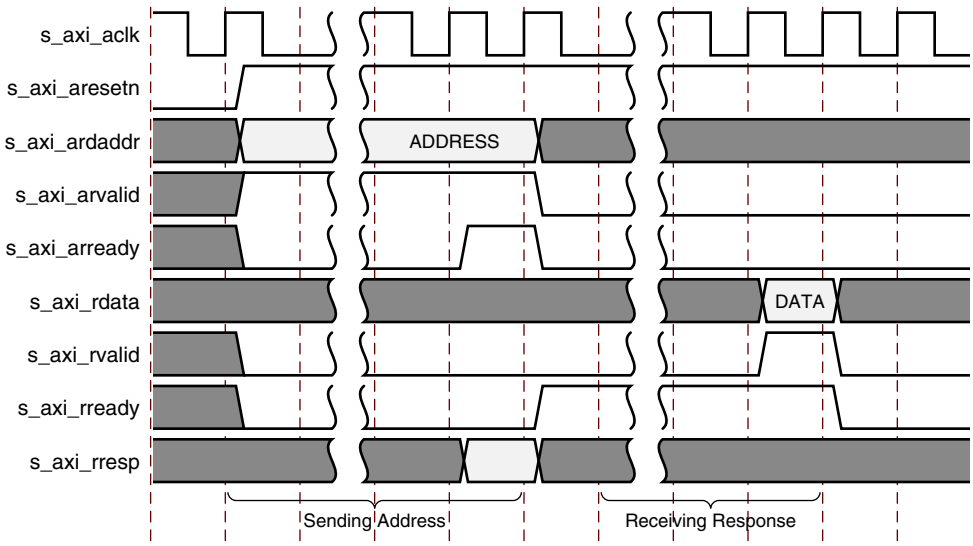

Figure 7-2: **Management Register Read Timing**

Statistics Counters

During operation, the MAC core collects statistics on the success and failure of various operations for processing by network management entities elsewhere in the system. These statistics are accessed through the Management Interface. A list of statistics is shown in Table 7-2.

As per [Ref 6], sub-clause 5.2.1, these statistic counters are wraparound counters and do not have a reset function. They do not reset upon being read and only return to zero when they naturally wrap around or when the device is reconfigured.

All Statistics Counters are Read Only, Write attempts to Statistics Counters are acknowledged with a SLVERR on the AXI4-Lite Bus.

Read of MSW of a particular counter is allowed only if the previous transaction was addressed to the LSW of the same counter, otherwise the MSW read operation is acknowledged with a SLVERR on the AXI4-Lite Bus. This restriction is to avoid the rollover of LSW counter into MSW counter between the read transactions.

Table 7-2: **Statistics Counters**

Address (Hex)	Name	Description
0x200	Received bytes - LSW	A count of bytes of frames that are received (destination address to frame check sequence inclusive).
0x204	Received bytes - MSW	
0x208	Transmitted bytes - LSW	A count of bytes of frames that are transmitted (destination address to frame check sequence inclusive).
0x20C	Transmitted bytes - MSW	
0x210	Undersize frames received - LSW	A count of the number of frames that were less than 64 bytes in length but were otherwise well formed.
0x214	Undersize frames received - MSW	
0x218	Fragment frames received – LSW	A count of the number of packets received that were less than 64 bytes in length and had a bad frame check sequence field.
0x21C	Fragment frames received – MSW	
0x220	64 byte frames received OK – LSW	A count of error-free frames received that were 64 bytes in length.
0x224	64 byte frames received OK – MSW	
0x228	65-127 byte frames received OK – LSW	A count of error-free frames received that were between 65 and 127 bytes in length inclusive.
0x22C	65-127 byte frames received OK – MSW	
0x230	128-255 byte frames received OK – LSW	A count of error-free frames received that were between 128 and 255 bytes in length inclusive.
0x234	128-255 byte frames received OK – MSW	
0x238	256-511 byte frames received OK – LSW	A count of error-free frames received that were between 256 and 511 bytes in length inclusive.
0x23C	256-511 byte frames received OK – MSW	
0x240	512-1023 byte frames received OK – LSW	A count of error-free frames received that were between 512 and 1023 bytes in length inclusive.
0x244	512-1023 byte frames received OK – MSW	
0x248	1024-MaxFrameSize byte frames received OK – LSW	A count of error-free frames received that were between 1024 bytes and the maximum legal frame size as specified in [Ref 6].
0x24C	1024-MaxFrameSize byte frames received OK – MSW	
0x250	Oversize frames received OK – LSW	A count of otherwise error-free frames received that exceeded the maximum legal frame length specified in [Ref 6].
0x254	Oversize frames received OK – MSW	
0x258	64 byte frames transmitted OK – LSW	A count of error-free frames transmitted that were 64 bytes in length.
0x25C	64 byte frames transmitted OK – MSW	
0x260	65-127 byte frames transmitted OK – LSW	A count of error-free frames transmitted that were between 65 and 127 bytes in length.
0x264	65-127 byte frames transmitted OK – MSW	
0x268	128-255 byte frames transmitted OK – LSW	A count of error-free frames transmitted that were between 128 and 255 bytes in length.
0x26C	128-255 byte frames transmitted OK – MSW	
0x270	256-511 byte frames transmitted OK – LSW	A count of error-free frames transmitted that were between 256 and 511 bytes in length.
0x274	256-511 byte frames transmitted OK – MSW	

Table 7-2: **Statistics Counters *(Cont'd)***

0x278	512-1023 byte frames transmitted OK – LSW	A count of error-free frames transmitted that were between 512 and 1023 bytes in length.
0x27C	512-1023 byte frames transmitted OK – MSW	
0x280	1024-MaxFrameSize byte frames transmitted OK – LSW	A count of error-free frames transmitted that were between 1024 bytes and the maximum legal frame length specified in [Ref 6].
0x284	1024-MaxFrameSize byte frames transmitted OK	
0x288	Oversize frames transmitted OK – LSW	A count of otherwise error-free frames transmitted that exceeded the maximum legal frame length specified in [Ref 6].
0x28C	Oversize frames transmitted OK – MSW	
0x290	Frames received OK – LSW	A count of error free frames received.
0x294	Frames received OK – MSW	
0x298	Frame Check Sequence errors – LSW	A count of received frames that failed the CRC check and were at least 64 bytes in length.
0x29C	Frame Check Sequence errors – MSW	
0x2A0	Broadcast frames received OK – LSW	A count of frames that were successfully received and were directed to the broadcast group address.
0x2A4	Broadcast frames received OK – MSW	
0x2A8	Multicast frames received OK – LSW	A count of frames that were successfully received and were directed to a non-broadcast group address.
0x2AC	Multicast frames received OK – MSW	
0x2B0	Control frames received OK – LSW	A count of error-free frames received that contained the MAC Control type identifier in the length/type field.
0x2B4	Control frames received OK – MSW	
0x2B8	Length/Type out of range – LSW	A count of error-free frames received that were at least 64 bytes in length where the length/type field contained a length value that did not match the number of MAC client data bytes received. The counter also increments for frames in which the length/type field indicated that the frame contained padding but where the number of MAC client data bytes received was greater than 64 bytes (minimum frame size).
0x2BC	Length/Type out of range – MSW	
0x2C0	VLAN tagged frames received OK – LSW	A count of error-free frames received with VLAN tags. This counter only increments when the receiver has VLAN operation enabled.
0x2C4	VLAN tagged frames received OK – MSW	
0x2C8	PAUSE frames received OK – LSW	A count of error-free frames received that contained the MAC Control type identifier 88-08 in the length/type field, contained a destination address that matched either the MAC Control multicast address or the configured source address of the MAC, contained the Pause opcode and were acted on by the MAC.
0x2CC	PAUSE frames received OK – MSW	
0x2D0	Control frames received with unsupported opcode – LSW	A count of error-free frames received that contained the MAC Control type identifier 88-08 in the length/type field but were received with an opcode other than the Pause opcode.
0x2D4	Control frames received with unsupported opcode – MSW	

Table 7-2: **Statistics Counters *(Cont'd)***

0x2D8	Frames transmitted OK – LSW	A count of error-free frames transmitted.
0x2DC	Frames transmitted OK – MSW	
0x2E0	Broadcast frames transmitted OK – LSW	A count of error-free frames transmitted to the broadcast address.
0x2E4	Broadcast frames transmitted OK – MSW	
0x2E8	Multicast frames transmitted OK – LSW	A count of error-free frames transmitted to group addresses other than the broadcast address.
0x2EC	Multicast frames transmitted OK – MSW	
0x2F0	Underrun errors – LSW	A count of frames that would otherwise be transmitted by the core but could not be completed due to the assertion of underrun during the frame transmission. This does not count frames which are less than 64 bytes in length.
0x2F4	Underrun errors – MSW	
0x2F8	Control frames transmitted OK – LSW	A count of error-free frames transmitted that contained the MAC Control Frame type identifier 88-08 in the length/type field.
0x2FC	Control frames transmitted OK – MSW	
0x300	VLAN tagged frames transmitted OK – LSW	A count of error-free frames transmitted that contained a VLAN tag. This counter only increments when the transmitter has VLAN operation enabled.
0x304	VLAN tagged frames transmitted OK – MSW	
0x308	PAUSE frames transmitted OK – LSW	A count of error-free pause frames generated and transmitted by the core in response to an assertion of pause_req.
0x30C	PAUSE frames transmitted OK – MSW	

Configuration Registers

After the core is powered up and reset, the client can reconfigure some of the core parameters from their defaults, such as flow control support and WAN/LAN connections. Configuration changes can be written at any time. Both the receiver and transmitter configuration register changes only take effect during interframe gaps. The exceptions to this are the configurable soft resets, which take effect immediately. Configuration of the MAC core is performed through a register bank accessed through the Management Interface. The configuration registers available in the core are detailed in Table 7-3.

Table 7-3: **Configuration Registers**

Address (Hex)	Description
0x400	Receiver Configuration Word 0
0x404	Receiver Configuration Word 1
0x408	Transmitter Configuration
0x40C	Flow Control Configuration
0x410	Reconciliation Sublayer Configuration
0x414	Receiver MTU Configuration Word
0x418	Transmitter MTU Configuration Word
0x4F8	Version Register (Read Only)
0x4FC	Capability Register (Read Only)

The contents of each configuration register are shown in Tables 7-4 through Table 7-8.

Table 7-4: **Receiver Configuration Word 0**

Bit	Default Value	Description
31:0	All 0s	**Pause frame MAC address [31:0]** This address is used by the MAC to match against the destination address of any incoming flow control frames. It is also used by the flow control block as the source address (SA) for any outbound flow control frames. This address does not have any affect on frames passing through the main transmit and receive datapaths of the MAC. The address is ordered so the first byte transmitted or received is the lowest positioned byte in the register; for example, a MAC address of AA-BB-CC-DD-EE-FF would be stored in Address[47:0] as 0xFFEEDDCCBBAA.

Table 7-5: **Receiver Configuration Word 1**

Bit	Default Value	Description
15:0	All 0s	**Pause frame MAC address [47:32]**. See description in Table 7-4.
23:16	N/A	Reserved
24	0	**Control Frame Length Check Disable.** When this bit is set to '1', the core does not mark MAC Control frames as 'bad' if they are greater than minimum frame length.
25	0	**Length/Type Error Check Disable.** When this bit is set to 1, the core does not perform the length/type field error checks as described in Length/Type Field Error Checks in Chapter 6. When this bit is set to 0, the length/type field checks are performed; this is normal operation.
26	0	**Receiver Preserve Preamble Enable**. When this bit is set to 1, the MAC receiver preserves the preamble field of the received frame. When it is 0, the preamble field is discarded as specified in [Ref 6].
27	0	**VLAN Enable.** When this bit is set to 1, VLAN tagged frames are accepted by the receiver.
28	1	**Receiver Enable**. If set to 1, the receiver block is operational. If set to 0, the block ignores activity on the physical interface RX port.
29	0	**In-band FCS Enable**. When this bit is 1, the MAC receiver passes the FCS field up to the client as described in Reception with In-Band FCS Passing in Chapter 6. When it is 0, the client is not passed to the FCS. In both cases, the FCS is verified on the frame.
30	0	**Jumbo Frame Enable**. When this bit is set to 1, the MAC receiver accepts frames that are greater than the maximum legal frame length specified in [Ref 6]. When this bit is 0, the MAC only accepts frames up to the legal maximum.
31	0	**Receiver reset**. When this bit is set to 1, the receiver is reset. The bit then automatically reverts to 0. This reset also sets all of the receiver configuration registers to their default values.

Table 7-6: **Transmitter Configuration Word**

Bit	Default Value	Description
22:0	N/A	Reserved
23	0	**Transmitter Preserve Preamble Enable**. When this bit is set to 1, the MAC transmitter preserves the custom preamble field presented on the Client Interface. When it is 0, the standard preamble field specified in [Ref 6] is transmitted.
24	0	**Deficit Idle Count Enable**. When this bit is set to 1, the core reduces the IFG as described in *IEE 803.2ae-2008* 46.3.1.4 Option 2 to support the maximum data transfer rate. When this bit is set to 0, the core always stretches the IFG to maintain start alignment. This bit is cleared and has no effect if Interframe Gap Adjust is enabled.
25	0	**Interframe Gap Adjust Enable**. When this bit is set to 1, the core reads the value on the port tx_ifg_delay at the start of a frame transmission and adjust the interframe gap accordingly. See Interframe Gap Adjustment in Chapter 6. When this bit is set to 0, the transmitter outputs the minimum Inter Frame Gap. This bit has no effect when bit 26 (LAN/WAN mode) is set to 1.
26	0	**WAN Mode Enable.** When this bit is set to 1, the transmitter automatically inserts extra idles into the interframe gap (IFG) to reduce the average data rate to that of the OC-192 SONET payload rate (WAN mode). When this bit is set to 0, the transmitter uses normal Ethernet interframe gaps (LAN mode). When the transmitter is in WAN mode, jumbo frames should be limited to 16384 bytes maximum
27	0	**VLAN Enable.** When this bit is set to 1, the transmitter allows the transmission of VLAN tagged frames.
28	1	**Transmitter Enable.** When this bit is 1, the transmitter is operational. When it is 0, the transmitter is disabled.
29	0	**In-band FCS Enable.** When this bit is 1, the MAC transmitter expects the FCS field to be passed in by the client as described in Transmission with In-Band FCS Passing in Chapter 6. When this bit is 0, the MAC transmitter appends padding as required, computes the value for the FCS field and appends it to the frame.
30	0	**Jumbo Frame Enable**. When this bit is set to 1, the MAC transmitter sends frames that are greater than the maximum legal frame length specified in [Ref 6]. When this bit is 0, the MAC only sends frames up to the legal maximum.
31	0	**Transmitter Reset**. When this bit is set to 1, the transmitter is reset. The bit then automatically reverts to 0. This reset also sets all of the transmitter configuration registers to their default values.

Table 7-7: **Flow Control Configuration Word**

Bit	Default Value	Description
28:0	N/A	Reserved
29	1	**Flow Control Enable (RX)**. When this bit is 1, received flow control frames inhibit the transmitter operation as described in Receiving a Pause Frame in Chapter 6. When this bit is 0, received flow control frames are always passed up to the client.
30	1	**Flow Control Enable (TX).** When this bit is 1, asserting the PAUSE_REQ signal sends a flow control frame out from the transmitter. When this bit is 0, asserting the PAUSE_REQ signal has no effect.
31	N/A	Reserved

Table 7-8: **Reconciliation Sublayer Configuration Word**

Bit	Default Value	Description
26:0	N/A	Reserved
27	0	**Fault Inhibit**. When this bit is set to 0, the Reconciliation Sublayer transmits ordered sets as laid out in [Ref 6]; that is, when the RS is receiving Local Fault ordered sets, it transmits Remote Fault ordered sets. When it is receiving Remote Fault ordered sets, it transmits idles code words. When this bit is set to 1, the reconciliation sublayer always transmits data presented to it by the MAC, regardless of whether fault ordered sets are being received.
28	N/A	**Local Fault Received**. If this bit is 1, the RS layer is receiving local fault sequence ordered sets. Read-only.
29	N/A	**Remote Fault Received**. If this bit is 1, the RS layer is receiving remote fault sequence ordered sets. Read-only.
30	N/A	**Transmit DCM Locked**. If this bit is 1, the Digital Clock Management (DCM) block for the transmit-side clocks (GTX_CLK, XGMII_TX_CLK, TX_CLK) is locked. If this bit is 0, the DCM is not locked. Read-only.
31	N/A	**Receive DCM Locked**. If this bit is 1, the Digital Clock Management (DCM) block for the receive-side clocks (XGMII_RX_CLK, RX_CLK) is locked. If this bit is 0, the DCM is not locked. Read-only.

Table 7-9: **Receiver MTU Configuration Word**

Bit	Default Value	Description
14:0	0x05EE	**RX MTU Size**. This value is used as the maximum frame size allowed as described in Receiver Maximum Permitted Frame Length in Chapter 6 when RX MTU Enable is set to 1. Only values of 1518 or greater are legal for RX MTU size and the core does not enforce this size on write. Ensure that only legal values are written to this register for correct core operation.
15	N/A	Reserved

Table 7-9: **Receiver MTU Configuration Word *(Cont'd)***

16	0	**RX MTU Enable**. When this bit is set to 1, the value in RX MTU Size is used as the maximum frame size allowed as described in Receiver Maximum Permitted Frame Length in Chapter 6. When set to 0 frame handling depends on the other configuration settings.
31:17	N/A	Reserved

Table 7-10: **Transmitter MTU Configuration Word**

Bit	Default Value	Description
14:0	0x05EE	**TX MTU Size**. This value is used as the maximum frame size allowed as described in Transmitter Maximum Permitted Frame Length in Chapter 6 when TX MTU Enable is set to 1. Only values of 1518 or greater are legal for TX MTU size and the core does not enforce this size on write. Ensure that only legal values are written to this register for correct core operation.
15	N/A	Reserved
16	0	**TX MTU Enable**. When this bit is set to 1, the value in TX MTU Size is used as the maximum frame size allowed as described in Transmitter Maximum Permitted Frame Length in Chapter 6. When set to 0 frame handling depends on the other configuration settings.
31:17	N/A	Reserved

Table 7-11: **Version Register**

Bit	Default Value	Description
7:0	All 0s	**Patch Level**. This field indicates the patch status of the core. (When this value is 0x00 it indicates a non patched version, when 0x01 indicates Rev 1 etc.)
15:8	N/A	Reserved
23:16	0x01	**Minor Revision**. This field indicates the minor revision of the core.
31:24	0x0B	**Major Revision**. This field indicates the major revision of the core.

Table 7-12: **Capability Register**

Bit	Default Value	Description
0	0	**Line rate 10Mbit**. This bit indicates that the core has a capability to support the 10 Mbit line rate.
1	0	**Line rate 100Mbit**. This bit indicates that the core has a capability to support the 100 Mbit line rate.
2	0	**Line rate 1Gbit**. This bit indicates that the core has a capability to support the 1 Gbit line rate.
3	1	**Line rate 10Gbit**. This bit indicates that the core has a capability to support the 10Gbit line rate.
7:4	N/A	Reserved
8	1	**Statistics Counter**. This bit indicates that the core has statistics counters.
31:9	N/A	Reserved

MDIO Interface

The Management Interface is also used to access the MDIO Interface of the MAC core; this interface is used to access the Managed Information Block (MIB) of the PHY components attached to the MAC core. A list of MDIO Registers is shown in Table 7-13.

Table 7-13: **MDIO Configuration Registers**

Address (Hex)	Description
0x500	MDIO Configuration word 0
0x504	MDIO Configuration word 1
0x508	MDIO TX Data
0x50c	MDIO RX Data (Read-only)

The contents of each configuration register are shown in Table 7-14 through Table 7-17.

Table 7-14: **MDIO Configuration Word 0**

Bit	Default Value	Description
5:0	All 0s	**Clock Divide**. Used as a divider value to generate MDC signal at 2.5 MHz. See MDIO Interface.
6	0	**MDIO Enable**. When this bit is 1, the MDIO interface can be used to access attached PHY devices. When this bit is 0, the MDIO interface is disabled and the MDIO signal remains inactive.
31:7	N/A	Reserved

Table 7-15: **MDIO Configuration Word 1**

Bit	Default Value	Description
6:0	N/A	Reserved
7	1	**MDIO Ready**. When this bit is 1 MDIO master is ready for an MDIO transaction, when 0 MDIO master is busy in a transaction and goes to 1 when the pending transaction is complete. This bit is Read-Only.
10:8	N/A	Reserved
11	0	**Initiate**. If a 1 is written to this bit when MDIO Ready is 1 an MDIO transaction is initiated. This bit goes to '0' automatically when the pending transaction completed.
13:12	N/A	Reserved
15:14	0	**TX OP**. Op code for the MDIO transaction. For more details see the MDIO transactions Figure 7-5 through Figure 7-8.
20:16	All 0s	**DEVAD**. Device address for the MDIO transaction
23:21	N/A	Reserved
28:24	All 0s	**PRTAD**. Port address for the MDIO transaction
31:29	N/A	Reserved

Table 7-16: **MDIO TX Data**

Bit	Default Value	Description
15:0	All 0s	**MDIO TX Data**. MDIO Write data, can be address of the device based on the OP code.
31:16	N/A	Reserved

Table 7-17: **MDIO RX Data**

Bit	Default Value	Description
15:0	All 0s	**MDIO RX Data**. MDIO Read data.
31:16	N/A	Reserved

The MDIO Interface supplies a clock to the external devices, MDC. This clock is derived from the `s_axi_aclk` signal, using the value in the Clock Divide[5:0] configuration register.

The frequency of MDC is given by this equation:

$$f_{MDC} = \frac{f_{\text{HOST\_CLK}}}{(1 + \text{Clock Divide}[5:0]) \times 2}$$

The frequency of MDC given by this equation should not exceed 2.5 MHz to comply with the specification for this interface, [Ref 6]. To prevent MDC from being out of specification, the Clock Divide[5:0] value powers up at 000000, and while this value is in the register, it is impossible to enable the MDIO Interface.

MDIO Transaction initiation and completion are shown in Figure 7-3.

When MDC, PRTAD, DEVAD and OP are programmed and MDIO is enabled, if MDIO Ready bit in the MDIO configuration register is 1, the MDIO transaction can be initiated by writing a '1' to the initiate bit (bit 11 of MDIO Configuration word 1).

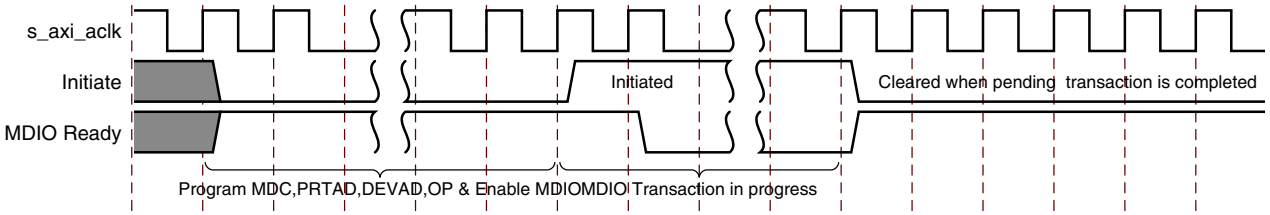

Figure 7-3: **MDIO Initiate**

The ports of the MDIO interface itself are shown in Table 7-18.

Table 7-18: **MDIO Interface Ports**

Name	Direction	Description
mdc	Output	MDIO Clock
mdio_in	Input	MDIO Input
mdio_out	Output	MDIO Output
mdio_tri	Output	MDIO 3-state. 1 disconnects the output driver from the MDIO bus

The bidirectional data signal MDIO is implemented as three unidirectional signals. These can be used to drive a 3-state buffer either in the FPGA SelectIO™ interface buffer on in a separate device. Figure 7-4 illustrates the used of a SelectIO interface 3-state buffer as the bus interface.

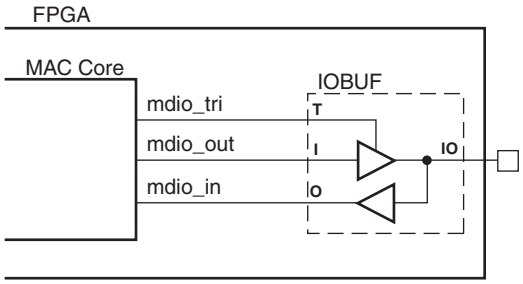

Figure 7-4: **Using a SelectIO Interface Tristate Buffer to Drive MDIO**

There are four different transaction types for MDIO, and they are described in the next four sections. In these sections, these abbreviations apply:

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

Set Address Transaction

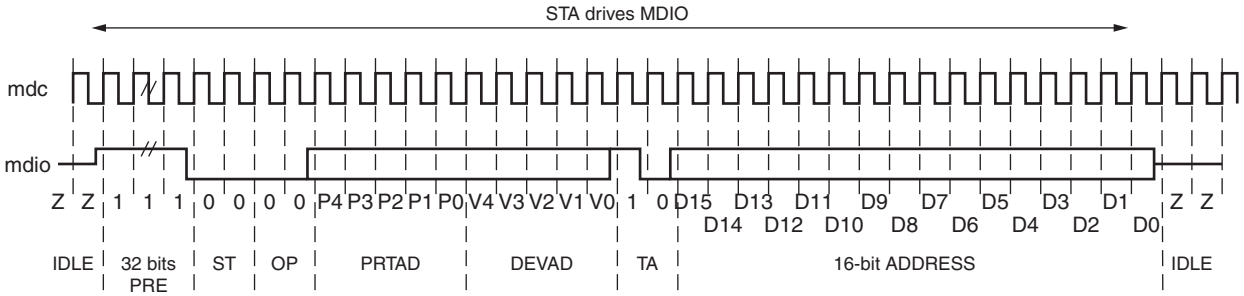

Figure 7-5: **MDIO Set Address Transaction**

Figure 7-5 shows an Address transaction; this is defined by OP=00. This is used to set the internal 16-bit address register of the PHY device for subsequent data transactions. This is called the "current address" in the following sections.

Write Transaction

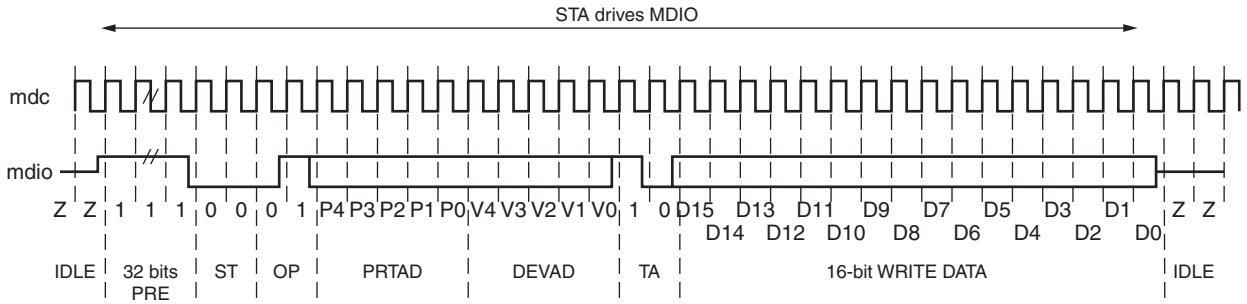

Figure 7-6: **MDIO Write Transaction**

Figure 7-6 shows a Write transaction; this is defined by OP=01. The PHY device takes the 16-bit word in the data field and writes it to the register at the current address.

Read Transaction

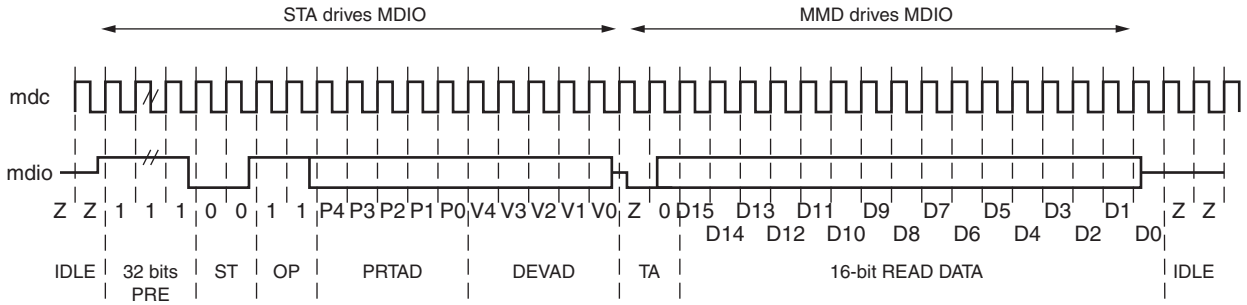

Figure 7-7: **MDIO Read Transaction**

Figure 7-7 shows a Read transaction; this is defined by OP=11. The PHY device returns the 16-bit word from the register at the current address.

Post-read-increment-address Transaction

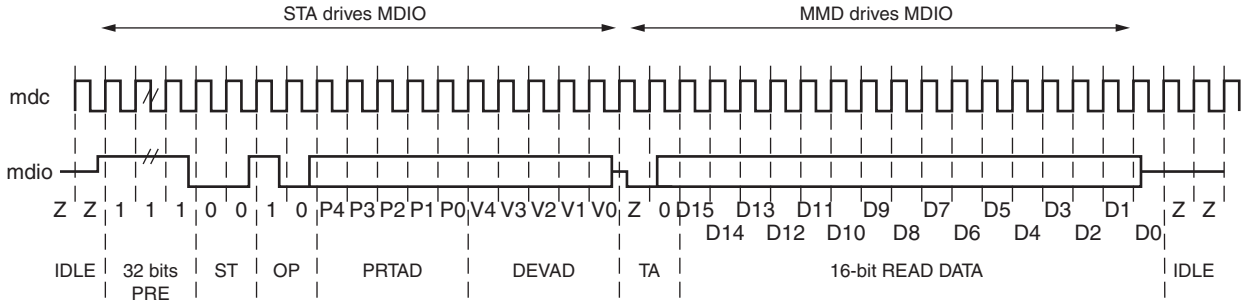

Figure 7-8: **MDIO Read-and-increment Transaction**

Figure 7-8 shows a Post-read-increment-address transaction; this is defined by OP=10. The PHY device 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.

For details of the register map of PHY layer devices and a fuller description of the operation of the MDIO Interface itself, see [Ref 6].

Using the AXI4-Lite Interface to access PHY registers over MDIO

The AXI4-Lite interface is used to access the MDIO ports on the core and access PHY registers either in devices external to the FPGA, or, in the case of XAUI, RXAUI and 10-Gigabit Ethernet PCS/PMA, PHY registers in an associated soft core on the same FPGA. Because the MDIO interface is a relatively slow two-wire interface, MDIO accesses can take many AXI4-Lite cycles to complete.

Prior to any MDIO accesses taking place, the MDIO Configuration Word 0 register must be written to with a valid Clock Divide value and the MDIO Enable bit set.

The target for PHY register accesses is set by the value of the PRTAD and DEVAD fields in the MDIO Configuration Word 1 register. Each port should have a unique 5-bit port address set on each PHY on that port (internal or external).

To write to a PHY register, first the register address must be set, then a second transaction performed to write the value from that address. This is done by setting the target port and device addresses in MDIO Configuration Word 1, setting the target register address in the MDIO TX Data register, setting the TX OP field of MDIO Configuration Word 1 to ADDRESS and starting the transaction; then setting the MDIO Tx Data register to the data to be written, the TX OP field to WRITE and starting a follow-up transaction.

To read from a PHY register, first the register address must be set, then a second transaction performed to read the value from that address. This is done by setting the target port and device addresses in MDIO Configuration Word 1, setting the target register address in the MDIO TX Data register, setting the TX OP field of MDIO Configuration Word 1 to ADDRESS and starting the transaction; then setting the TX OP field to READ and starting a follow-up transaction, and reading the result from the MDIO RX Data register.

If successive registers in the same PHY address space are to be read, a special read mode of the protocol can be used. First, the read address should be set as above, but for the first read operation, the Post-Read-Increment-Address opcode should be written into the relevant field of the MDIO Configuration Word1. This returns the read value as above, and also has the side effect of moving the read address to the next register value in the PHY. Thus, repeating the same opcode sequentially returns data from consecutive register addresses in the PHY. Table 7-19 provides an example of a PHY register write using MDIO, to a XAUI configured as a DTE XS on port 0.

Table 7-19: **Example of a PHY Register Write using MDIO**

Register	Access	Value	Activity
MDIO TX Data	Write	0x00000019	Address of XAUI test control register.
MDIO Configuration Word 1	Write	0x00050800	Initiate the Address transaction by setting the DEVAD (5), PRTAD (0), OP(00) and Initiate bit.
MDIO Configuration Word 1	Read	0x00050080	Poll bit 7 (MDIO Ready) until it becomes 1. The Initiate bit returns to 0.
MDIO TX Data	Write	0x00000006	Turn on transmit test pattern, mixed frequency.
MDIO Configuration Word 1	Write	0x00054800	Initiate the Write transaction by setting the DEVAD (5), PRTAD (0), OP(01) and Initiate bit.
MDIO Configuration Word 1	Read	0x00054080	Initiate the Write transaction by setting the DEVAD (5), PRTAD (0), OP(01) and Initiate bit.

Table 7-20 provides an example of a PHY register read using MDIO, to a PCS on port 7.

Table 7-20: **Example of a PHY Register Read using MDIO**

Register	Access	Value	Activity
MDIO TX Data	Write	0x00000001	Address of PCS status 1 register.
MDIO Configuration Word 1	Write	0x07030800	Initiate the Address transaction by setting the DEVAD (3), PRTAD (7), OP(00) and Initiate bit.
MDIO Configuration Word 1	Read	0x07030080	Poll bit 7 (MDIO Ready) until it becomes 1. The Initiate bit returns to 0.
MDIO Configuration Word 1	Write	0x0703C800	Initiate the Read transaction by setting the DEVAD (5), PRTAD (0), OP(11) and Initiate bit.
MDIO Configuration Word 1	Read	0x0703C080	Poll bit 7 (MDIO Ready) until it becomes 1. The Initiate bit returns to 0.
MDIO RX Data	Read	0x00000006	Read the status value back from the RX data register.

Interrupt Registers

Interrupt block is implemented in the 10-Gigabit Ethernet MAC core to assert an interrupt when a pending MDIO transaction is completed. Interrupt registers are shown in Table 7-21.

Table 7-21: **Interrupt Registers**

Address (Hex)	Default Value	Description
0x600	0x00	**Interrupt Status Register.** Indicates the status of an interrupt. Any asserted interrupt can be cleared by directly writing a 0 to the concerned bit location.
0x610	0x00	**Interrupt Pending Register.** Indicates the pending status of an interrupt. Writing a 1 to any bit of this register clears that particular interrupt. Bits in this register are set only when the corresponding bits in IER & ISR are set.
0x620	0x00	**Interrupt Enable Register.** Indicates the enable state of an interrupt. Writing a 1 to any bit enables that particular interrupt.
0x630	0x00	**Interrupt Acknowledge Register. (Write only)** Writing a 1 to any bit of this register clears that particular interrupt.

Bit '0' of all the interrupt registers is used to indicate the MDIO transaction complete interrupt. Bits [31:1] are Reserved.

The Configuration and Status Vector

If the optional Management interface is omitted from the core, all of relevant configuration and status signals are brought out of the core. These signals are bundled into the `configuration_vector` and `status_vector` signals. The bit mapping of the signals are defined in Table 7-22 and Table 7-23. See the corresponding entry in the configuration register tables for the full description of each signal.

You can change the configuration vector signals at any time; however, with the exception of the reset signals and the flow control configuration signals, they do not take effect until the current frame has completed transmission or reception. It is recommended that the configuration vector input signals are driven synchronous from the appropriate clock domain, as detailed in Table 7-22 and Table 7-23.

Table 7-22: **tx_configuration_vector Bit Definitions**

Bit(s)	Description
0	**Transmitter Reset**. When this bit is 1, the MAC transmitter is held in reset. This signal is an input to the reset circuit for the transmitter block. See Reset Circuits in Chapter 10 special for details.
1	**Transmitter Enable**. When this bit is set to 1, the transmitter is operational. When set to 0, the transmitter is disabled.
2	**Transmitter VLAN Enable**. When this bit is set to 1, the transmitter allows the transmission of VLAN tagged frames.
3	**Transmitter In-Band FCS Enable**. When this bit is 1, the MAC transmitter expects the FCS field to be pass in by the client as described in Transmission with In-Band FCS Passing in Chapter 6. When it is 0, the MAC transmitter appends padding as required, compute the FCS and append it to the frame.
4	**Transmitter Jumbo Frame Enable**. When this bit is 1, the MAC transmitter allows frames larger than the maximum legal frame length specified in [Ref 6] to be sent. When set to 0, the MAC transmitter only allows frames up to the legal maximum to be sent.
5	**Transmit Flow Control Enable**. When this bit is 1, asserting the pause_req signal causes the MAC core to send a flow control frame out from the transmitter as described in Transmitting a Pause Frame in Chapter 6. When this bit is 0, asserting the pause_req signal has no effect.
6	Reserved
7	**Transmitter Preserve Preamble Enable**. When this bit is set to 1, the MAC transmitter preserves the custom preamble field presented on the Client Interface. When it is 0, the standard preamble field specified in [Ref 6] is transmitted.
8	**Transmitter Interframe Gap Adjust Enable**. When this bit is 1, the transmitter reads the value of the tx_ifg_delay port and set the interframe gap accordingly. If it is set to 0, the transmitter inserts a minimum interframe gap. This bit is ignored if bit 53 (Transmitter LAN/WAN Mode) is set to 1.
9	**Transmitter LAN/WAN Mode.** When this bit is 1, the transmitter automatically inserts idles into the Inter Frame Gap to reduce the average data rate to that of the OC-192 SONET payload rate (WAN mode). When this bit is 0, the transmitter uses standard Ethernet interframe gaps (LAN mode).

Table 7-22: **tx_configuration_vector Bit Definitions *(Cont'd)***

Bit(s)	Description
10	**Deficit Idle Count Enable**. When this bit is set to 1, the core reduces the IFG as described in [Ref 6], 46.3.1.4 Option 2 to support the maximum data transfer rate. When this bit is set to 0, the core always stretches the IFG to maintain start alignment. This bit is cleared and has no effect if LAN Mode and In-band FCS are both enabled or if Interframe Gap Adjust is enabled.
13:11	Reserved
14	**TX MTU Enable**. When this bit is set to 1, the value in TX MTU Size is used as the maximum frame size allowed as described in Transmitter Maximum Permitted Frame Length in Chapter 6. When set to 0 frame handling depends on the other configuration settings.
15	Reserved
31:16	**RX MTU Size**. This value is used as the maximum frame size allowed as described in Receiver Maximum Permitted Frame Length in Chapter 6 when RX MTU Enable is set to '1'.
79:32	**Transmitter Pause Frame Source Address[47:0].** This address is used by the MAC core as the source address for any outbound flow control frames. This address does not have any effect on frames passing through the main transmit datapath of the MAC. The address is ordered such that the first byte transmitted or received is the least significant byte in the register; for example, a MAC address of AA-BB-CC-DD-EE-FF is stored in byte [79:32] as 0xFFEEDDCCBBAA.

Table 7-23: **rx_configuration_vector Bit Definitions**

Bit(s)	Description
0	**Receiver Reset**. When this bit is 1, the MAC receiver is held in reset. This signal is an input to the reset circuit for the receiver block. See Reset Circuits in Chapter 10 special for details.
1	**Receiver Enable**. When this bit is set to 1, the receiver is operational. When set to 0, the receiver is disabled.
2	**Receiver VLAN Enable**. When this bit is set to 1, the receiver allows the reception of VLAN tagged frames.
3	**Receiver In-Band FCS Enable**. When this bit is 1, the MAC receiver passes the FCS field up to the client as described in Reception with In-Band FCS Passing in Chapter 6. When it is 0, the MAC receiver does not pass the FCS field. In both cases, the FCS field is verified on the frame.
4	**Receiver Jumbo Frame Enable**. When this bit is 0, the receiver does not pass frames longer than the maximum legal frame size specified in [Ref 6]. When it is 1, the receiver does not have an upper limit on frame size.
5	**Receive Flow Control Enable**. When this bit is 1, received flow control frames inhibit the transmitter operation as described in Receiving a Pause Frame in Chapter 6. When it is 0, received flow frames are passed up to the client.
6	Reserved

Table 7-23: **rx_configuration_vector Bit Definitions *(Cont'd)***

Bit(s)	Description
7	**Receiver Preserve Preamble Enable**. When this bit is set to 1, the MAC receiver preserves the preamble field on the received frame. When it is 0, the preamble field is discarded as specified in [Ref 6].
8	**Receiver Length/Type Error Disable**. When this bit is set to 1, the core does not perform the length/type field error check as described in Length/Type Field Error Checks in Chapter 6. When this bit is 0, the length/type field checks are performed; this is normal operation.
9	**Control Frame Length Check Disable**. When this bit is set to 1, the core does not mark control frames as 'bad' if they are greater than the minimum frame length.
10	**Reconciliation Sublayer Fault Inhibit**. When this bit is 0, the reconciliation sublayer transmits ordered sets as laid out in [Ref 6]; that is, when the RS is receiving local fault ordered sets, it transmits Remote Fault ordered sets. When it is receiving Remote Fault ordered sets, it transmits idle code words. When this bit is 1, the Reconciliation Sublayer always transmits the data presented to it by the MAC, regardless of whether fault ordered sets are being received.
13:11	Reserved
14	**RX MTU Enable**. When this bit is set to 1, the value in RX MTU Size is used as the maximum frame size allowed as described in Receiver Maximum Permitted Frame Length in Chapter 6. When set to 0 frame handling depends on the other configuration settings.
15	Reserved
31:16	**RX MTU Size**. This value is used as the maximum frame size allowed as described in Receiver Maximum Permitted Frame Length in Chapter 6 when RX MTU Enable is set to 1.
79:32	**Receiver Pause Frame Source Address[47:0].** This address is used by the MAC core to match against the Destination address of any incoming flow control frames. This address does not have any effect on frames passing through the main receive datapath of the MAC. The address is ordered such that the first byte transmitted or received is the least significant byte in the register; for example, a MAC address of AA-BB-CC-DD-EE-FF is stored in byte [47:0] as 0xFFEEDDCCBBAA.

Table 7-24: **status_vector Bit Definitions**

Bit(s)	Description
0	**Local Fault Received**. If this bit is 1, the RS layer is receiving local fault sequence ordered sets. Read-only.
1	**Remote Fault Received**. If this bit is 1, the RS layer is receiving remote fault sequence ordered sets. Read-only.

Statistics Vectors

Transmit

The statistics for the frame transmitted are contained within the `tx_statistics_vector`. The vector is synchronous to the transmitter clock, `tx_clk0` and is driven following frame transmission. The bit field definition for the vector is defined in Table 7-25. All bit fields, with the exception of `byte_valid`, are valid only when the `tx_statistics_valid` is asserted. This is illustrated in Figure 7-9. `byte_valid` is significant on every `tx_clk0` cycle.

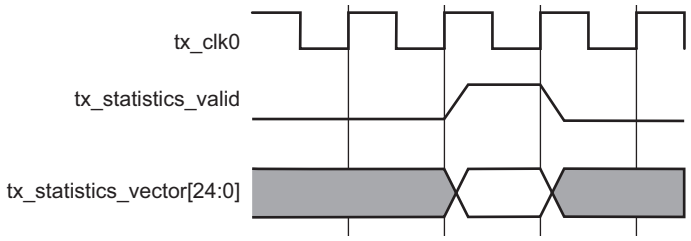

Figure 7-9: **Transmitter Statistics Output Timing**

Table 7-25: **Transmit Statistics Vector Bit Description**

Bit	Name	Description
25	pause_frame_transmitted	Asserted if the previous frame was a pause frame that was initiated by the MAC in response to a pause_req assertion.
24 to 21	bytes_valid	The number of MAC frame bytes transmitted on the last clock cycle (DA to FCS inclusive). This can be between 0 and 8. This is valid on every clock cycle, it is not validated by tx_statistics_valid. The information for the bytes_valid field is sampled at a different point in the transmitter pipeline than the rest of the tx_statistics_vector bits.
20	vlan_frame	Asserted if the previous frame contained a VLAN identifier in the length/type field and transmitter VLAN operation is enabled.
19 to 5	frame_length_count	The length of the previously transmitted frame in bytes. The count stays at 32767 for any jumbo frames larger than this value.
4	control_frame	Asserted if the previous frame had the special MAC Control Type code 88-08 in the length/type field.
3	underrun_frame	Asserted if the previous frame transmission was terminated due to an underrun error.
2	multicast_frame	Asserted if the previous frame contained a multicast address in the destination address field.
1	broadcast_frame	Asserted if the previous frame contained the broadcast address in the destination address field.
0	successful_frame	Asserted if the previous frame was transmitted without error.

Receive

The statistics for the frame received are contained within the `rx_statistics_vector`. The vector is driven synchronously by the receiver clock, `rx_clk0`, following frame reception. The bit field definition for the vector is defined in Table 7-26.

All bit fields, with the exception of `bytes_valid`, are valid only when `rx_statistics_valid` is asserted. This is illustrated in Figure 7-10. `bytes_valid` is significant on every `rx_clk0` cycle.

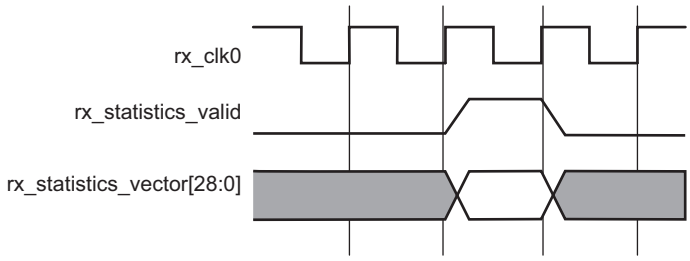

Figure 7-10: **Receiver Statistics Output Timing**

Table 7-26: **Receive Statistics Vector Description**

Bits	Name	Description
29	Length/Type Out of Range	Asserted if the length/type field contained a length value that did not match the number of MAC client data bytes received. Also High if the length/type field indicated that the frame contained padding but the number of client data bytes received was not equal to 64 bytes (minimum frame size).
28	bad_opcode	Asserted if the previous frame was error free, contained the special Control Frame identifier in the length/type field but contained an opcode that is unsupported by the MAC (any opcode other than Pause).
27	flow_control_frame	Asserted if the previous frame was error free, contained the Control Frame type identifier 88-08 in the length/type field, contained a destination address that matched either the MAC Control multicast address or the configured source address of the MAC, contained the Pause opcode and was acted on by the MAC.
26 to 23	bytes_valid	The number of MAC frame bytes received on the last clock cycle (DA to FCS inclusive). This can be between 0 and 8. This is valid on every clock cycle, it is not validated by `rx_statistics_valid`. The information for the bytes_valid field is sampled at a different point in the transmitter pipeline than the rest of the `rx_statistics_vector` bits.
22	vlan_frame	Asserted if the previous frame contained a VLAN tag in the length/type field and VLAN operation was enabled in the receiver.
21	out_of_bounds	Asserted if the previous frame exceeded the maximum legal frame length specified in [Ref 6]. This is only asserted if jumbo frames are disabled.

Table 7-26: **Receive Statistics Vector Description *(Cont'd)***

Bits	Name	Description
20	control_frame	Asserted if the previous frame contained the MAC Control Frame identifier 88-08 in the length/type field.
19 to 5	frame_length_count	The length in bytes of the previous received frame. The count stays at 32767 for any Jumbo frames larger than this value. If jumbo frames are disabled, the count stays at 1518 for non-VLAN frames and 1522 for VLAN frames.
4	multicast_frame	Asserted if the previous frame contained a multicast address in the destination address field.
3	broadcast_frame	Asserted if the previous frame contained the broadcast address in the destination address field.
2	fcs_error	Asserted if the previous frame received had an incorrect FCS value or the MAC detected error codes during frame reception.
1	bad_frame	Asserted if the previous frame received contained errors.
0	good_frame	Asserted if the previous frame received was error free.

Chapter 8

Using Flow Control

This chapter describes the operation of the flow control logic of the core. The flow control block is designed to clause 31 of the *IEEE 802.3-2008* standard. The MAC can be configured to transmit pause requests and to act on their reception; these modes of operation can be independently enabled or disabled. See Configuration Registers in Chapter 7.

Overview of Flow Control

Flow Control Requirement

Figure 8-1 illustrates the requirement for Flow Control. The MAC at the right side of the figure has a reference clock slightly faster than the nominal 156.25 MHz, and the MAC at the left side of the figure has a reference clock slightly slower than the nominal 156.25 MHz. This results in the MAC on the left not being able to match the full line rate of the MAC on the right (due to clock tolerances). The left MAC is illustrated as performing a loopback implementation, which results in the FIFO filling up over time. Without Flow Control, this FIFO eventually fills and overflows, resulting in the corruption or loss of Ethernet frames. Flow Control is one solution to this issue.

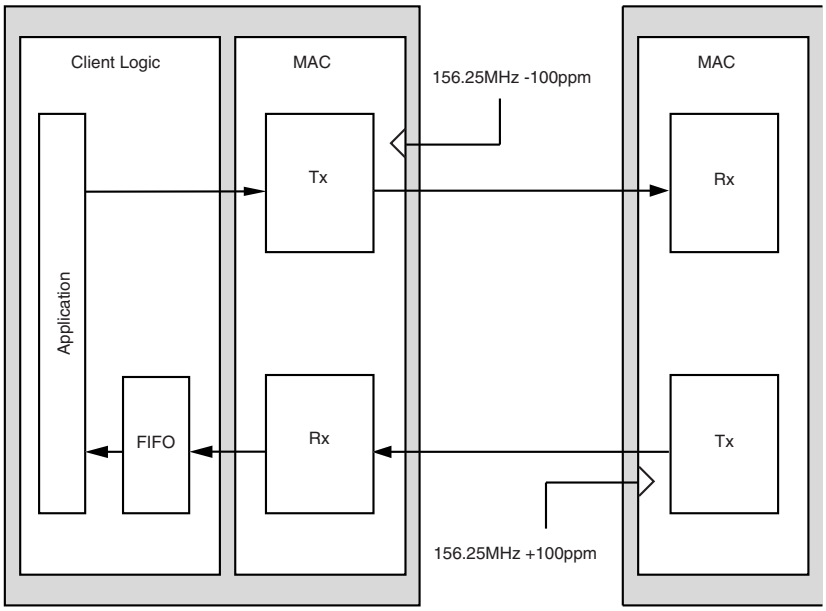

Figure 8-1: **The Requirement for Flow Control**

Flow Control Basics

A MAC might transmit a pause control frame to request that its link partner cease transmission for a defined period of time. For example, the MAC at the left side of Figure 8-1 can initiate a pause request when its client FIFO (illustrated) reaches a nearly full state.

A MAC should respond to received pause control frames by ceasing transmission of frames for the period of time defined in the received pause control frame. For example, the MAC at the right side of Figure 8-1 might cease transmission after receiving the pause control frame transmitted by the left-hand MAC. In a well designed system, the right MAC would cease transmission before the client FIFO of the left MAC overflowed. This provides time for the FIFO to be emptied to a safe level before normal operation resumes and safeguards the system against FIFO overflow conditions and frame loss.

Pause Control Frames

Control frames are a special type of Ethernet frame defined in clause 31 of the *IEEE 802.3-2008* standard. Control frames are identified from other frame types by a defined value placed into the length/type field (the MAC Control Type code). Control frame format is illustrated in Figure 8-2.

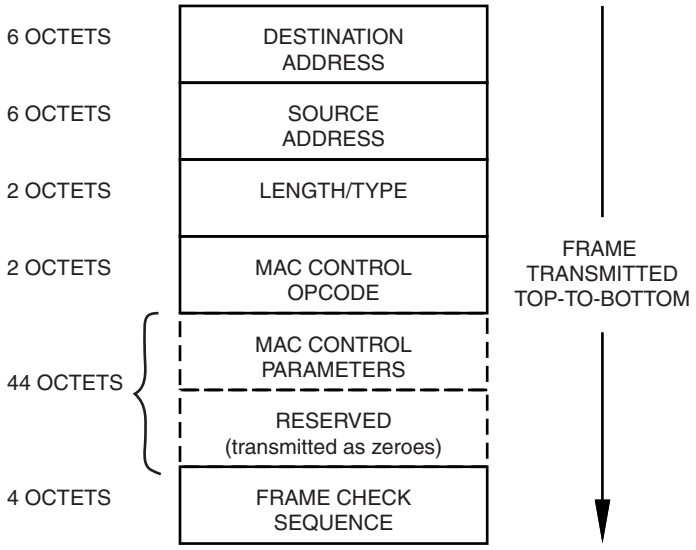

Figure 8-2: **MAC Control Frame Format**

A pause control frame is a special type of control frame, identified by a defined value placed into the MAC Control OPCODE field.

Note: MAC Control OPCODES other than for Pause (Flow Control) frames have recently been defined for Ethernet Passive Optical Networks.

The MAC Control Parameter field of the pause control frame contains a 16-bit field which contains a binary value directly relating to the duration of the pause. This defines the number of *pause_quantum* (512 bit times of the particular implementation). For 10-Gigabit Ethernet, a single *pause_quantum* corresponds to 51.2 ns.

Flow Control Operation of the 10-Gigabit MAC

Transmitting a Pause Control Frame

Core-Initiated Pause Request

If the MAC core is configured to support transmit flow control, the client can initiate a pause control frame by asserting the `pause_req` signal. Figure 8-3 displays this timing.

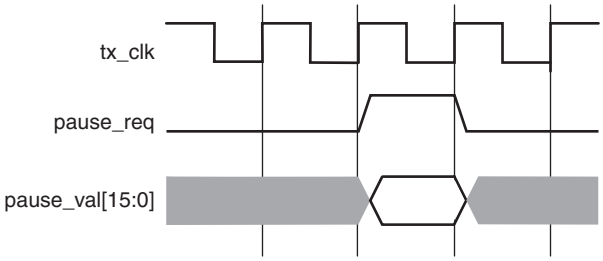

Figure 8-3: **Pause Request Timing**

This action causes the core to construct and transmit a pause control frame on the link with the MAC Control frame parameters (see Figure 8-2):

- The destination address used is an *IEEE 802.3-2008* globally assigned multicast address (which any Flow Control capable MAC responds to).
- The source address used is the configurable Pause Frame MAC Address (see Configuration Registers in Chapter 7).
- The value sampled from the `pause_val[15:0]` port at the time of the `pause_req` assertion is encoded into the MAC control parameter field to select the duration of the pause (in units of *pause_quantum*).

If the transmitter is currently inactive at the time of the pause request, then this pause control frame is transmitted immediately. If the transmitter is currently busy, the current frame being transmitted is allowed to complete; the pause control frame then follows in preference to any pending client supplied frame.

A pause control frame initiated by this method is transmitted even if the transmitter itself has ceased transmission in response to receiving an inbound pause request.

Note: Only a single pause control frame request is stored by the transmitter. If the `pause_req` signal is asserted numerous times in a short time period (before the control pause frame transmission has had a chance to begin), then only a single pause control frame is transmitted. The `pause_val[15:0]` value used is the most recent value sampled.

Client-Initiated Pause Request

For maximum flexibility, flow control logic can be disabled in the core (see Configuration Registers in Chapter 7) and alternatively implemented in the client logic connected to the core. Any type of control frame can be transmitted through the core through the client interface using the same transmission procedure as a standard Ethernet frame (see Normal Frame Transmission in Chapter 6).

Receiving a Pause Control Frame

Core-Initiated Response to a Pause Request

An error-free control frame is a received frame matching the format of Figure 8-2. It must pass all standard receiver frame checks (for example, FCS field checking); in addition, the control frame received must be exactly 64-bytes in length (from destination address through to the FCS field inclusive: this is minimum legal Ethernet MAC frame size and the defined size for control frames).

Any control frame received that does not conform to these checks contains an error, and it is passed to the receiver client with the `rx_bad_frame` signal asserted.

Pause Frame Reception Disabled

When pause control reception is disabled (see Configuration Registers in Chapter 7), an error free control frame is received through the client interface with the `rx_good_frame` signal asserted. In this way, the frame is passed to the client logic for interpretation (see Transmitting a Pause Frame in Chapter 6).

Pause Frame Reception Enabled

When pause control reception is enabled (see Configuration Registers in Chapter 7) and an error-free frame is received by the MAC core, the frame decoding functions are performed:

1. The destination address field is matched against the *IEEE 802.3-2008* globally assigned multicast address or the configurable Pause Frame MAC Address (see Configuration Registers in Chapter 7).
2. The length/type field is matched against the MAC Control Type code.
3. The opcode field contents are matched against the Pause opcode.

If any of these checks are false, the frame is ignored by the flow control logic and passed up to the client logic for interpretation by marking it with `rx_good_frame` asserted. It is then the responsibility of the MAC client logic to decode, act on (if required) and drop this control frame.

If all these checks are true, the 16-bit binary value in the MAC Control Parameters field of the control frame is then used to inhibit transmitter operation for the required number of *pause_quantum*. This inhibit is implemented by delaying the assertion of `tx_ack` at the transmitter client interface until the requested pause duration has expired. Because the received pause frame has been acted upon, it is passed to the client with `rx_bad_frame` asserted to indicate to the client that can now be dropped.

Note: Any frame in which the length/type field contains the MAC Control Type should be dropped by the receiver client logic. All control frames are indicated by `rx_statistic_vector` bit 19 (see Receive in Chapter 7).

Client-Initiated Response to a Pause Request

For maximum flexibility, flow control logic can be disabled in the core (see Configuration Registers in Chapter 7) and alternatively implemented in the client logic connected to the core. Any type of error free control frame is then passed through the core with the `rx_good_frame` signal asserted. In this way, the frame is passed to the client for interpretation. It is then the responsibility of the client to drop this control frame and to act on it by ceasing transmission through the core, if applicable.

Flow Control Implementation Example

This explanation is intended to describe a simple (but crude) example of a Flow Control implementation to introduce the concept.

Consider the system illustrated in Figure 8-1. The MAC on the left-hand side of the figure cannot match the full line rate of the right-hand MAC due to clock tolerances. Over time, the FIFO illustrated fills and overflows. The aim is to implement a Flow Control method which, over a long time period, reduces the full line rate of the right-hand MAC to average that of the lesser full line rate capability of the left-hand MAC.

Method

1. Choose a FIFO nearly full occupancy threshold (7/8 occupancy is used in this description but the choice of threshold is implementation specific). When the occupancy of the FIFO exceeds this occupancy, initiate a single pause control frame with 0xFFFF used as the *pause_quantum* duration (0xFFFF is placed on `pause_val[15:0]`). This is the maximum pause duration. This causes the right-hand MAC to cease transmission and the FIFO of the left-hand MAC starts to empty.
2. Choose a second FIFO occupancy threshold (3/4 is used in this description but the choice of threshold is implementation specific). When the occupancy of the FIFO falls below this occupancy, initiate a second pause control frame with 0x0000 used as the *pause_quantum* duration (0x0000 is placed on `pause_val[15:0]`). This indicates a zero pause duration, and upon receiving this pause control frame, the right-hand MAC immediately resumes transmission (it does not wait for the original requested pause duration to expire). This pause control frame can therefore be considered a "pause cancel" command.

Operation

Figure 8-4 illustrates the FIFO occupancy over time.

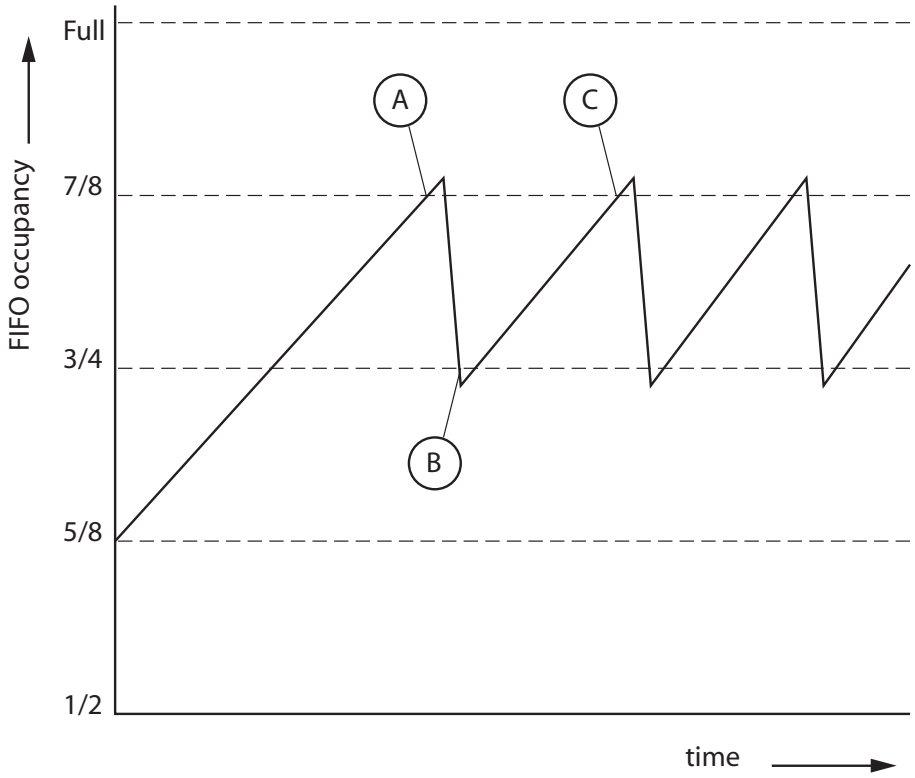

Figure 8-4: **Flow Control Implementation Triggered from FIFO Occupancy**

1. The average FIFO occupancy of the left-hand MAC gradually increases over time due to the clock tolerances. At point A, the occupancy has reached the threshold of 7/8 occupancy. This triggers the maximum duration pause control frame request.
2. Upon receiving the pause control frame, the right-hand MAC ceases transmission.
3. After the right-hand MAC ceases transmission, the occupancy of the FIFO attached to the left-hand MAC rapidly empties. The occupancy falls to the second threshold of 3/4 occupancy at point B. This triggers the zero duration pause control frame request (the pause cancel command).
4. Upon receiving this second pause control frame, the right-hand MAC resumes transmission.
5. Normal operation resumes and the FIFO occupancy again gradually increases over time. At point C, this cycle of Flow Control repeats.

XILINX®

Chapter 9

Constraining the Core

This chapter describes how to constrain a design containing the 10-Gigabit Ethernet MAC core. This is illustrated by the User Constraint File (UCF) delivered with the core at generation time. See Chapter 13, Detailed Example Design for a complete description of the Xilinx® CORE Generator™ tool output files.

Not all constraints are relevant for a particular implementation of the core; consult the UCF created with the core instance to see exactly what constraints are relevant.

Device, Package, and Speed Grade Selection

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

```
# set the part and package
CONFIG PART = xc6vlx240tff1156-1;
```

The 10-Gigabit Ethernet MAC core can be implemented in the following devices with the following speed grades:

- Virtex®-6 family devices, speed grade of -1 or faster
- Spartan®-6 family devices, speed grade of -3 or faster (no XGMII interface or WAN mode)

Clock Frequencies, Clock Management, and Placement

The core can have up to three clock domains; the transmit clock domain, derived from the `gtx_clk` signal, the receive clock domain, derived from the `xgmii_rx_clk` signal, and the `host_clk` domain.

```
############################################################
# Clock/period constraints                                 #
############################################################
# Main transmit clock/period constraints
NET "gtx_clk" TNM_NET = "xgmactxclk";
TIMESPEC "TS_xgmactx" = PERIOD "xgmactxclk" 6400 ps HIGH 50 %;

# Group the driven storage elements together for use in FROM-TO
constraints.
NET "tx_clk0" TNM_NET = "xgmactxgrp";
TIMESPEC "TS_clk0" = PERIOD "xgmactxgrp" 6400 ps HIGH 50 %;
```

This section sets the period of the transmit clock.

```
# Main receive clock/period constraints
NET "xgmii_rx_clk" TNM_NET = "xgmacrxclk";
```

```
TIMESPEC "TS_xgmacrx"    = PERIOD "xgmacrxclk" 6400 ps HIGH 50 %;

# Group the driven storage elements together for use in FROM-TO
constraints.
NET "fifo_block/xgmac_block/*_if/rx_clk0" TNM_NET = "xgmacrxgrp";
TIMESPEC "TS_rxclk0" = PERIOD "xgmacrxgrp" 6400 ps HIGH 50 %;
```

This section sets the period of the receive clock.

```
# The management clock can run at same speed as RX & TX.
NET "*s_axi_aclk_int" TNM_NET = "xgmacmangementgrp";
TIMESPEC "TS_s_axi_aclk_clk" = PERIOD "xgmacmangementgrp" 6400 ps HIGH
50 %;
```

This sets the period of the s_axi_clk.

```
# False paths on an internal counter load
INST
"*xgmac_core/BU2/U0/G_TX.txgen/tx_controller_inst/ifg_control_inst/sta
te_*" TNM = "xgmac_ifg_false_paths_src_1";
INST
"*xgmac_core/BU2/U0/G_TX.txgen/tx_controller_inst/ifg_control_inst/eof
_during_pad" TNM = "xgmac_ifg_false_paths_src_1";
INST
"*xgmac_core/BU2/U0/G_TX.txgen/tx_controller_inst/ifg_control_inst/ifg
_counter/count_*" TNM = "xgmac_ifg_false_paths_dst_1";
NET
"*xgmac_core/BU2/U0/G_TX.txgen/tx_controller_inst/ifg_control_inst/ifg
_count_init<*>" TPTHRU = "xgmac_ifg_false_paths_thru_1";
INST
"*xgmac_core/BU2/U0/G_TX.txgen/tx_controller_inst/ifg_control_inst/ifg
_counter/Mcount_count_cy<?>" TPTHRU = "xgmac_ifg_false_paths_thru_2";
TIMESPEC "TS_xgmac_ifg_false_paths_thru_1" = FROM
"xgmac_ifg_false_paths_src_1" THRU "xgmac_ifg_false_paths_thru_1" THRU
"xgmac_ifg_false_paths_thru_2" TO "xgmac_ifg_false_paths_dst_1" TIG;
```

Ignore some false paths in the design.

Flow Control Constraints

```
############################################################
# Flow control clock crossing timing constraint
INST
"*xgmac_core/BU2/U0/G_RX.rxgen/rx_pause_control_i/good_frame_to_tx"
TNM = "flow_grp";
INST
"*xgmac_core/BU2/U0/G_RX.rxgen/rx_pause_control_i/pause_value_to_tx_*"
TNM = "flow_grp";
INST
"*xgmac_core/BU2/U0/G_RX.rxgen/rx_pause_control_i/pause_req_to_tx" TNM
= "flow_grp";
TIMESPEC "TS_flow" = FROM "flow_grp" TO "xgmactxgrp" 6400 ps
```

These constraints cover paths that cross from the receive clock domain into the transmit clock domain in the flow control group.DATAPATHONLY;

Management Constraints

```
############################################################
#  MANAGEMENT CONSTRAINTS                                  #
#  Please do not edit these constraints.                   #
############################################################

### Configuration and status registers ###
# Clock domain crossings into and out of the configuration/status
registers
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/conf/fc_out_*" TNM =
"xgmac_config_regs";
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/conf/rx0_out_*" TNM =
"xgmac_config_regs";
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/conf/rx1_out_*" TNM =
"xgmac_config_regs";
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/conf/tx_out_*" TNM =
"xgmac_config_regs";
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/conf/rs_out_*" TNM =
"xgmac_config_regs";
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/conf/rs_local_fail_reg"
TNM = "xgmac_status_regs";
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/conf/rs_remote_fail_reg"
TNM = "xgmac_status_regs";
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/conf/tx_dcm_lock_reg"
TNM = "xgmac_status_regs";
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/conf/rx_dcm_lock_reg"
TNM = "xgmac_status_regs";


TIMESPEC "TS_config_to_tx" = FROM "xgmac_config_regs" TO "xgmactxgrp"
TIG;
TIMESPEC "TS_config_to_rx" = FROM "xgmac_config_regs" TO "xgmacrxgrp"
TIG;

# False paths from Reconciliation sublayer to the management status regs
INST "*xgmac_core/BU2/U0/rsgen/local_fail_reg" TNM =
"xgmac_fault_src_grp";
INST "*xgmac_core/BU2/U0/rsgen/remote_fail_reg" TNM =
"xgmac_fault_src_grp";
TIMESPEC "TS_rs_tig" = FROM "xgmac_fault_src_grp" TO
"xgmac_status_regs" TIG;
```

These constraints cover the clock-domain crossings from the management clock domain into the transmit and receive clock domains and from the reconciliation sublayer back into the management clock domain.

Statistics Counters

```
### Statistics ###
# Cover the clock domain crossing into and out of the management clock
domain; needs
# to be limited to a single tx/rx clock period to guarantee the
statistics
# data is stable at the management-side registers on a read.
INST
"*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/rx_data_reclock_*"
TNM = "xgmac_stats_rx_to_host_sources";
TIMESPEC "TS_stats_rx_to_host" = FROM "xgmac_stats_rx_to_host_sources"
TO "xgmacmangementgrp" TIG;

INST
"*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/tx_data_reclock_*"
TNM = "xgmac_stats_tx_to_host_sources";
TIMESPEC "TS_stats_tx_to_host" = FROM "xgmac_stats_tx_to_host_sources"
TO "xgmacmangementgrp" TIG;

INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/request_rx_int"
TNM = "xgmac_stats_host_rx_sources";
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/request_tx_int"
TNM = "xgmac_stats_host_tx_sources";
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/request_rx_reg1"
TNM = "xgmac_stats_rx_sources";
INST "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/request_tx_reg1"
TNM = "xgmac_stats_tx_sources";
TIMESPEC "TS_stats_host_to_tx" = FROM "xgmac_stats_host_rx_sources" TO
"xgmac_stats_tx_sources" TIG;
TIMESPEC "TS_stats_host_to_rx" = FROM "xgmac_stats_host_tx_sources" TO
"xgmac_stats_rx_sources" TIG;

NET
"*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/rx_clk_quarter_small"
TNM_NET = "stats_rx_slowgrp1";
NET "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/rx_we_reg"
TNM_NET = "stats_rx_slowgrp2";
NET
"*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/rx_carry_reset_reg<*>
" TNM_NET = "stats_rx_slowgrp2";
TIMESPEC TS_slow_rx1 = FROM "stats_rx_slowgrp1" TO "stats_rx_slowgrp1"
TS_xgmacrx * 2.0;
TIMESPEC TS_slow_rx2 = FROM "stats_rx_slowgrp2" TO "stats_rx_slowgrp1"
TS_xgmacrx * 2.0;

NET
"*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/tx_clk_quarter_small"
TNM_NET = "stats_tx_slowgrp1";
NET "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/tx_we_reg"
TNM_NET = "stats_tx_slowgrp2";
NET
"*xgmac_core/BU2/U0/G_MANAGEMENT.managen/i1.stat/tx_carry_reset_reg<*>
" TNM_NET = "stats_tx_slowgrp2";

TIMESPEC TS_slow_tx1 = FROM "stats_tx_slowgrp1" TO "stats_tx_slowgrp1"
TS_xgmactx * 2.0;
TIMESPEC TS_slow_tx2 = FROM "stats_tx_slowgrp2" TO "stats_tx_slowgrp1"
TS_xgmactx * 2.0;
```

This section constrains the statistic counter logic. As well as clock domain crossings from management clock to and from transmit and receive clock domains, there are multicycle paths covered in these constraints.

MDIO Interface

```
# The constraint on the clock period for the MDIO block is half the MDC
# minimum period of 400 ns; the turnaround phase of a read operation
leads
# to a half-cycle operation in the middle of the transaction.
# Value below is correct for a 133 MHz bus2ip_clk - 200 ns / 7518 ps =
26.60
NET "*xgmac_core/BU2/U0/G_MANAGEMENT.managen/mdio_master_i/mdc_ce" TNM
= "mdc_grp";
TIMESPEC "TSmdc" = FROM "mdc_grp" TO "mdc_grp" TS_s_axi_aclk_clk * 26;
```

This section constrains the low-speed MDIO logic.

I/O Constraints

```
############################################################
# I/O constraints                                          #
############################################################

#  Ensure that XGMII DDR registers are placed in IOBs
#  and utilize the HSTL_I voltage standard.
INST "*txd_ddr*"                    IOB             = "TRUE";
NET  "xgmii_txd<*>"                IOSTANDARD      = "HSTL_I";
INST "*txc_ddr*"                    IOB             = "TRUE";
NET  "xgmii_txc<*>"                IOSTANDARD      = "HSTL_I";
INST "*tx_clk_ddr"                  IOB         = "TRUE";
NET  "xgmii_tx_clk"                IOSTANDARD = "HSTL_I";

#  Ensure that XGMII DDR registers are placed in IOBs
#  and utilize the HSTL_I voltage standard.
NET  "xgmii_rx_clk"                IOSTANDARD  = "HSTL_I";
NET  "xgmii_rx_clk"                IOBDELAY    = "NONE";
NET  "xgmii_rxd<*>"                IOSTANDARD   = "HSTL_I";
NET  "xgmii_rxd<*>"                IOBDELAY     = "NONE";
INST "*xgmii_if/rxd_loop[*].rxd_ddr" IOB             = "TRUE";

#  XGMII RXC Constraints.
NET "xgmii_rxc<*>" IOSTANDARD   = "HSTL_I";
NET "xgmii_rxc<*>" IOBDELAY     = "NONE\";

INST "*xgmii_if/rxc_loop[*].rxc_ddr"  IOB = "TRUE";

#  DDR Setup and Hold
INST "xgmii_rxd<*>" TNM = IN_XGMII;
INST "xgmii_rxc<?>" TNM = IN_XGMII;

# Define setup respect to the clock
TIMEGRP "IN_XGMII" OFFSET = IN  480 ps BEFORE "xgmii_rx_clk" RISING;
TIMEGRP "IN_XGMII" OFFSET = IN  480 ps BEFORE "xgmii_rx_clk" FALLING;
```

These constraints set the I/O standards used for the SelectIO™ interface pads and ensure that the DDR registers are placed in the I/O buffer. This section is only used in an XGMII implementation.

Chapter 10

Special Design Considerations

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

Reset Circuits

Internally, the core is divided up into clock/reset domains, which group together elements with the common clock and reset signals. The reset circuitry for one of these domains is illustrated in Figure 10-1.

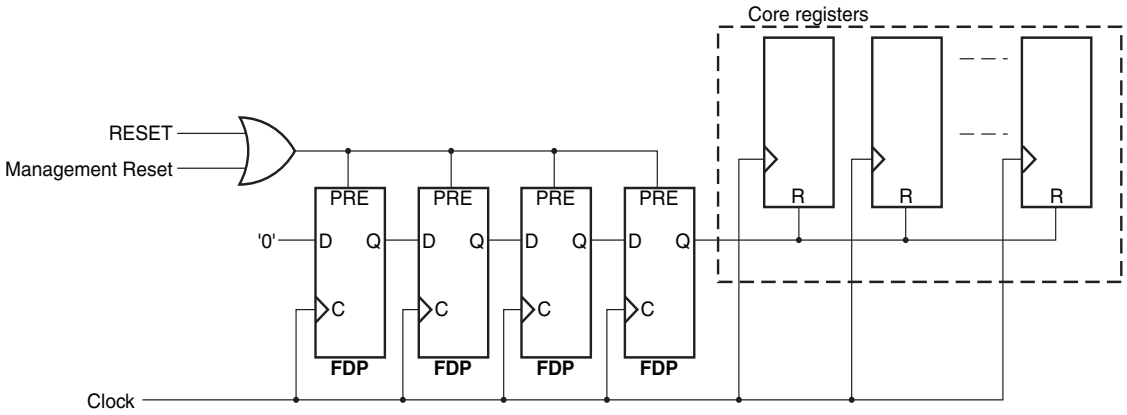

Figure 10-1: **Reset Circuit for a Single Clock/Reset Domain**

Multiple Core Instances

In a large design, it might be necessary or desirable to have more than one instance of the 10-Gigabit Ethernet MAC core on a single FPGA. One possible clock scheme for two instance with XGMII interfaces is shown in Figure 10-2.

The transmit clock `tx_clk0` can be shared among multiple core instances as illustrated, resulting in a common transmitter clock domain across the device.

A common receiver clock domain is not possible; each core derives an independent receiver clock from its XGMII interface as shown.

Although not illustrated, if the optional Management Interface is used, `host_clk` can also be shared between cores.The `host_clk` signal consumes another BUFG global clock buffer resource.

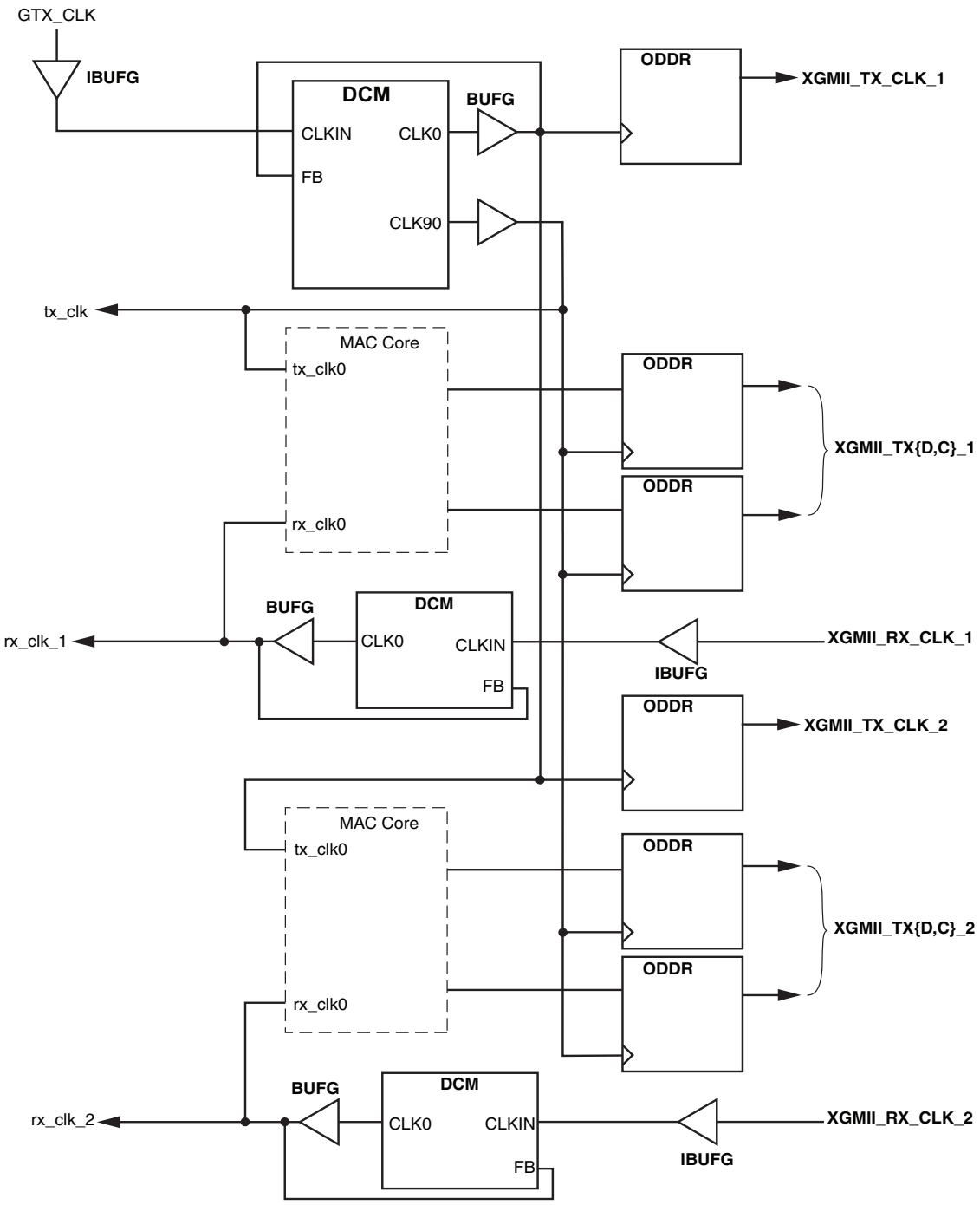

Figure 10-2: **Clock Management, Multiple Instances of the Core with XGMII**

Clock management for multiple cores with the 64-bit SDR interface is similar to that for the XGMII interface.

Pin Location Considerations for XGMII Interface

The MAC core allows for a flexible pinout of the XGMII and the exact pin locations are left to the designer. In doing so, codes of practice and device restrictions must be followed.

I/Os should be grouped in their own separate clock domains. XGMII contains two of these:

- `xgmii_rxd[31:0]` and `xgmii_rxc[3:0]`, which are centered with respect to `xgmii_rx_clk`
- `xgmii_txd[31:0]` and `xgmii_txc[3:0]`, which are centered with respect to `xgmii_tx_clk`

Interfacing to the Xilinx XAUI Core

The 10-Gigabit Ethernet MAC core can be integrated with the Xilinx® XAUI core in a single device to provide the PHY interface for the MAC.

A description of the latest available IP Update containing the XAUI core and instructions on obtaining and installing the IP Update can be found on the Xilinx XAUI core product page at:

www.xilinx.com/products/intellectual-property/XAUI.htm

A data sheet and other documentation for the XAUI core can also be found at this URL.

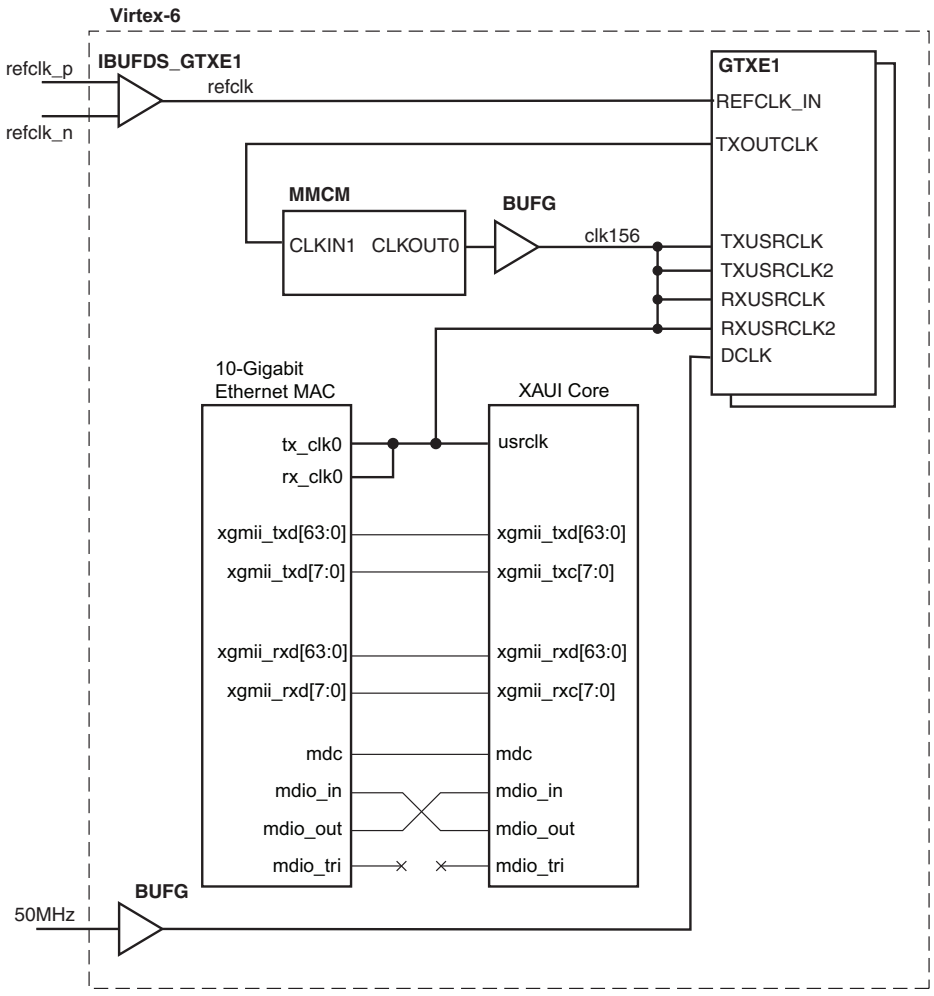

Figure 10-3: **10-Gigabit Ethernet MAC Core Integrated with XAUI Core - Virtex-6 FPGAs**

Figure 10-3 illustrates the connections and clock management logic required to interface the 10-Gigabit Ethernet MAC core to the XAUI core in Virtex®-6 FPGAs. This shows that:

- Direct connections are made between the PHY-side interface of the 10-Gigabit Ethernet MAC and the client-side interface of the XAUI core.
- If the 10-Gigabit Ethernet MAC core instance has been customized with the Management Interface, then the MDIO port can be connected directly to the XAUI core MDIO port to access the embedded configuration and status registers.
- Both the transmit and receive sides of the XAUI core operate on a single clock domain. This single clock is used as the 156.25 MHz system clock for both cores and the transmitter and receiver logic in the 10-Gigabit Ethernet MAC core now operate in a single unified clock domain.

Note: This final point indicates that some simplification to the UCF for the 10-Gigabit Ethernet MAC core is possible. The constraints that refer to clock-domain crossings from the transmit clock domain to the receive clock domain and vice-versa can be safely removed (although these do not cause harm if left).

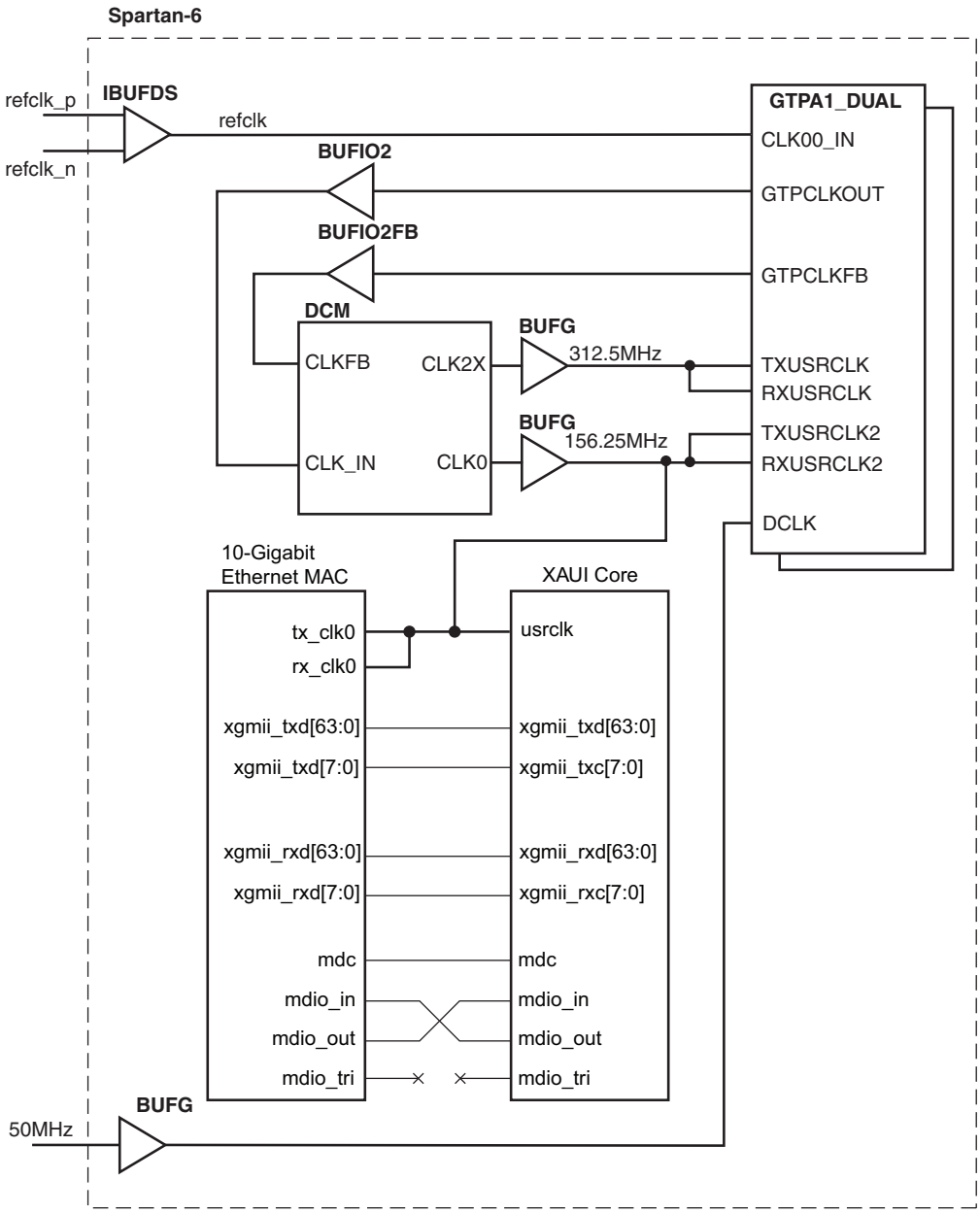

Figure 10-4: **10-Gigabit Ethernet MAC Core Integrated with XAUI Core - Spartan 6 FPGA**

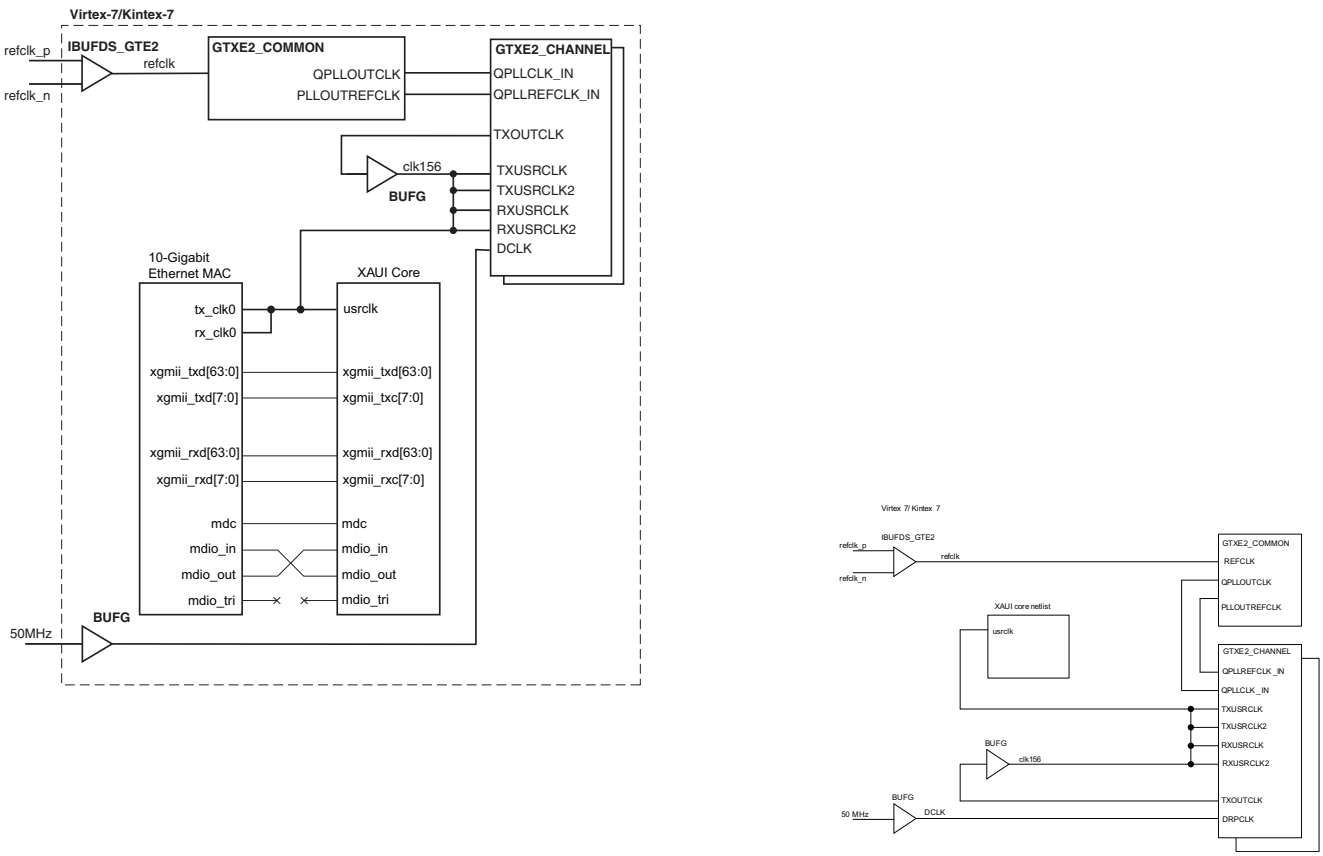

Figure 10-5: **10-Gigabit Ethernet MAC Core Integrated with XAUI Core - Virtex-7 and Kintex-7 FPGAs**

Figure 10-4 shows the MAC core integrated with the XAUI core on Spartan®-6 devices. Figure 10-5 shows the MAC core integrated with the XAUI core on Virtex-7 and Kintex™-7 FPGAs. All of the points made with respect to the Virtex-6 FPGA implementation apply here also.

For details on clocks and transceiver placement using the XAUI core, see [Ref 3].

Interfacing with the RXAUI Core

The 10-Gigabit Ethernet MAC core can be integrated with the Xilinx RXAUI core in a single device to provide the PHY interface for the MAC.

A description of the latest available IP Update containing the RXAUI core and instructions on obtaining and installing the IP Update can be found on the Xilinx RXAUI core product page at:

www.xilinx.com/products/intellectual-property/RXAUI.htm

A data sheet and other documentation for the RXAUI core can also be found at this URL.

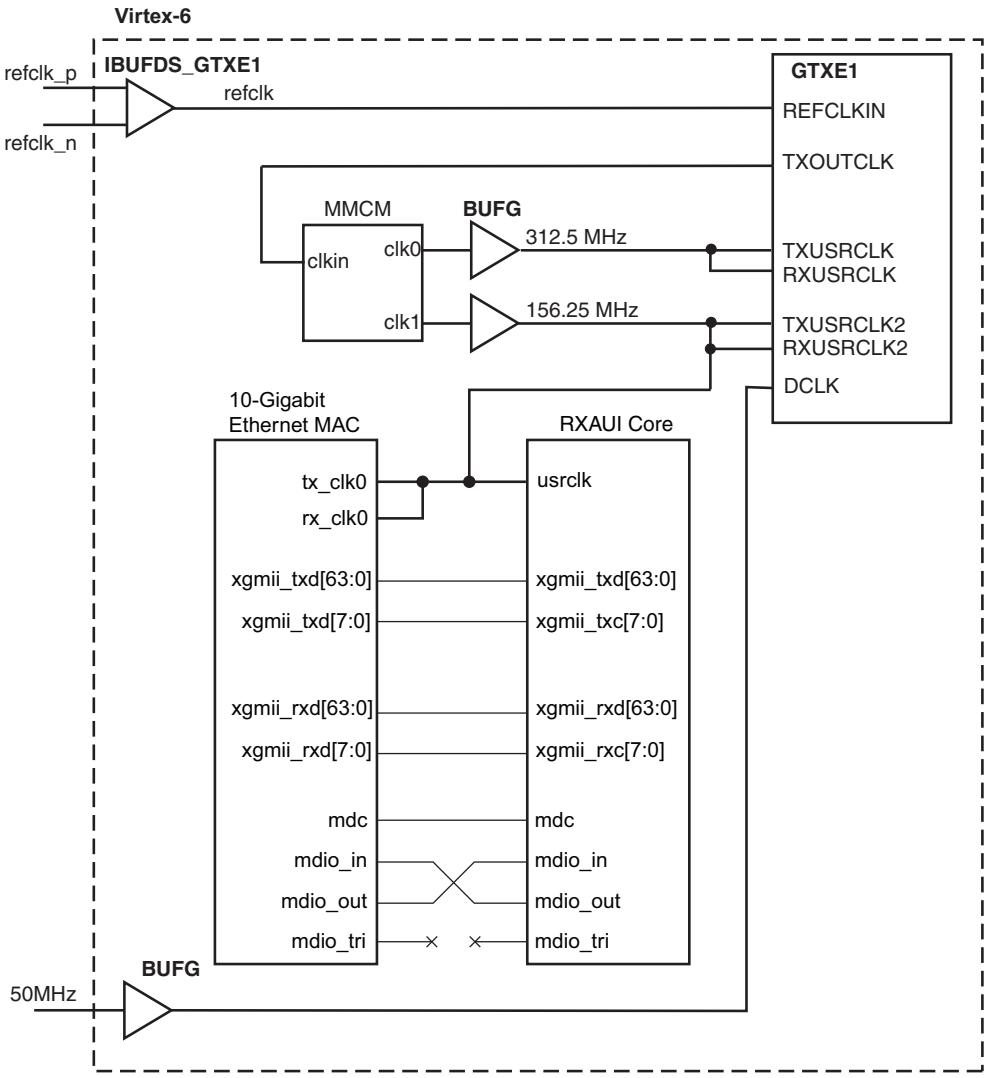

Figure 10-6: **10-Gigabit Ethernet MAC Core Integrated with RXAUI Core - Virtex 6 FPGA**

Figure 10-6 illustrates the connections and clock management logic required to interface the 10-Gigabit Ethernet MAC core to the RXAUI core in Virtex-6 FPGAs. This shows that:

- Direct connections are made between the PHY-side interface of the 10-Gigabit Ethernet MAC and the client-side interface of the RXAUI core.
- If the 10-Gigabit Ethernet MAC core instance has been customized with the Management Interface, then the MDIO port can be connected directly to the RXAUI core MDIO port to access the embedded configuration and status registers.
- Both the transmit and receive client interfaces of the RXAUI core operate on a single clock domain. This single clock is used as the 156.25 MHz system clock for both cores and the transmitter and receiver logic in the 10-Gigabit Ethernet MAC core now operate in a single unified clock domain.

Note: This final point indicates that some simplification to the UCF for the 10-Gigabit Ethernet MAC core is possible. The constraints that refer to clock-domain crossings from the transmit clock domain to the receive clock domain and vice-versa can be safely removed (although these do not cause harm if left).

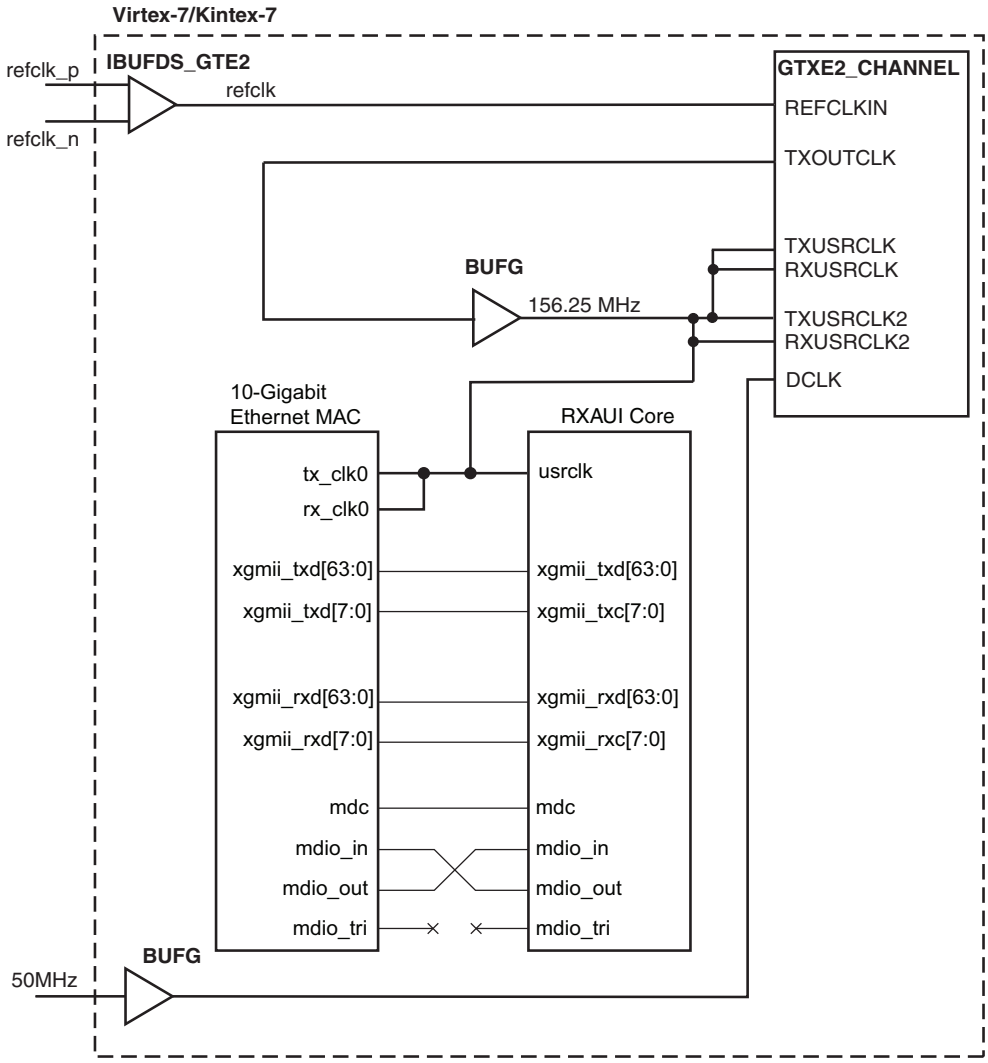

Figure 10-7: **10-Gigabit Ethernet MAC Core Integrated with RXAUI Core - Virtex-7 and Kintex-7 FPGAs**

Figure 10-7 shows the MAC core integrated with the RXAUI core on Virtex-7 and Kintex-7 FPGAs. All of the points made with respect to the Virtex-6 FPGA implementation apply here also.

For details on clocks and transceiver placement using the RXAUI core, see [Ref 4].

Interfacing to the 10-Gigabit Ethernet PCS/PMA Core

The 10-Gigabit Ethernet MAC core can be integrated with the Xilinx 10-Gigabit Ethernet PCS/PMA core in a single device to provide the PHY interface for the MAC.

A description of the latest available IP Update containing the 10-Gigabit Ethernet PCS/PMA core and instructions on obtaining and installing the IP Update can be found on the Xilinx 10-Gigabit Ethernet PCS/PMA Product Page at:

www.xilinx.com/products/intellectual-property/10GBASE-R.htm

A data sheet and other documentation for the PCS/PMA core can also be found at this URL.

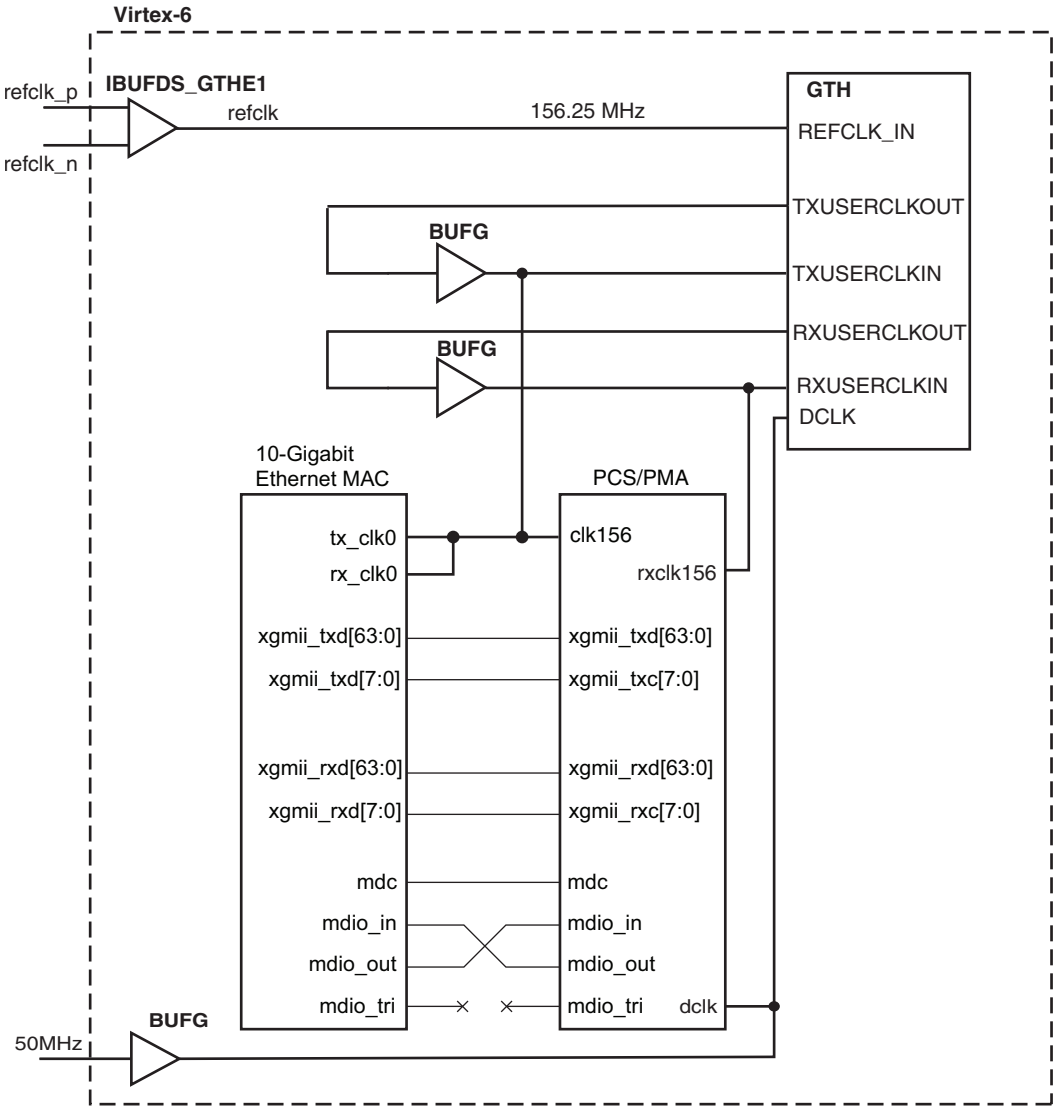

Figure 10-8: **10-Gigabit Ethernet MAC Core Integrated with PCS/PMA Core - Virtex 6 FPGA**

Figure 10-8 illustrates the connections and clock management logic required to interface the 10-Gigabit Ethernet MAC core to the PCS/PMA core in Virtex-6 FPGAs. This shows that:

- Direct connections are made between the PHY-side interface of the 10-Gigabit Ethernet MAC and the client-side interface of the PCS/PMA core.
- If the 10-Gigabit Ethernet MAC core instance has been customized with the Management Interface, then the MDIO port can be connected directly to the PCS/PMA core MDIO port to access the embedded configuration and status registers.
- Both the transmit and receive client interfaces of the PCS/PMA core operate on a single clock domain. This single clock is used as the 156.25 MHz system clock for both cores and the transmitter and receiver logic in the 10-Gigabit Ethernet MAC core now operate in a single unified clock domain.

Note: This final point indicates that some simplification to the UCF for the 10-Gigabit Ethernet MAC core is possible. The constraints that refer to clock-domain crossings from the transmit clock domain to the receive clock domain and vice-versa can be safely removed (although these do not cause harm if left).

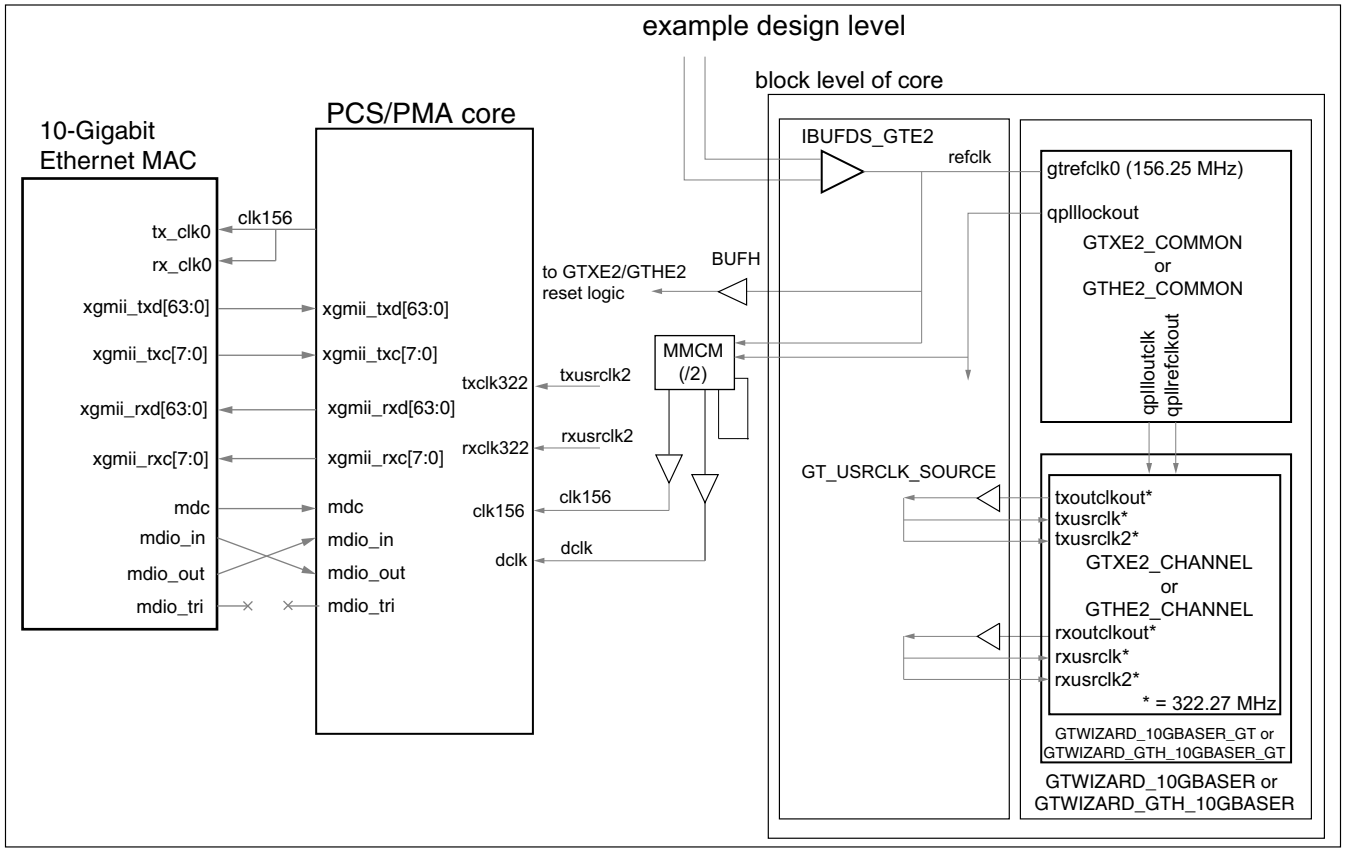

Figure 10-9: **10-Gigabit Ethernet MAC Core Integrated with PCS/PMA Core - Virtex-7 and Kintex-7 FPGAs**

Figure 10-9 shows the MAC core integrated with the PCS/PMA core on Virtex-7 and Kintex-7 FPGAs. All of the points made with respect to the Virtex-6 FPGA implementation apply here also.

For details on clocks and transceiver placement using the PCS/PMA core, see [Ref 2].

Behavior of the Evaluation Core in Hardware

When the core is generated with a Full System Hardware Evaluation, the core can be tested in the target device for several hours before ceasing to function.

Symptoms of the hardware evaluation timeout include:

- The transmitter failing to assert TX_AXIS_TREADY in response to TX_AXIS_TVALID.
- The receiver failing to recognize frames in the inbound data stream.
- After the timeout occurs, the core can be reactivated by reconfiguring the FPGA.

XILINX®

Chapter 11

Implementing Your Design

This chapter describes how to simulate and implement your design containing the 10-Gigabit Ethernet MAC core.

Synthesis

XST: VHDL

In the CORE Generator™ tool project directory, there is a *xgmac_component_name*.vho file that is a component and instantiation template for the core. Use this template to help instance the 10-Gigabit Ethernet MAC core into your VHDL source.

After 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 [Ref 1] for details on creating project and synthesis script files and running the xst program.

XST: Verilog

In the Xilinx® CORE Generator™ project directory, there is a module declaration for the 10-Gigabit Ethernet MAC core at:

project_directory/*component_name*/implement/*component_name*_mod.v

Use this module to help instance the 10-Gigabit Ethernet MAC core into your Verilog source.

After 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

  project_directory/*component_name*/implement/*component_name*_mod.v

  as the first file 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 [Ref 1] for details on creating project and synthesis script files, and running the xst program.

Implementation

Generating the Xilinx Netlist

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

```
$ ngdbuild -sd path_to_xgmac_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 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.

VHDL

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

Verilog

```
$ netgen -sim -ofmt verilog -pcf top_level_module_name.pcf \
    -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 [Ref 5], also included in your Xilinx software installation.

Other Implementation Information

For details on the use of the Xilinx implementation tool flow including command line switches and options, consult the software manuals that came with the ISE® Design Suite.

Chapter 12

Quick Start Example Design

The quick start instructions let you quickly generate a 10-Gigabit Ethernet MAC core, run the design through implementation with the Xilinx tools, and simulate the example design using the provided demonstration test bench. For detailed information about the example design, see Chapter 13, Detailed Example Design.

Introduction

The 10-Gigabit Ethernet MAC example design consists of the following.

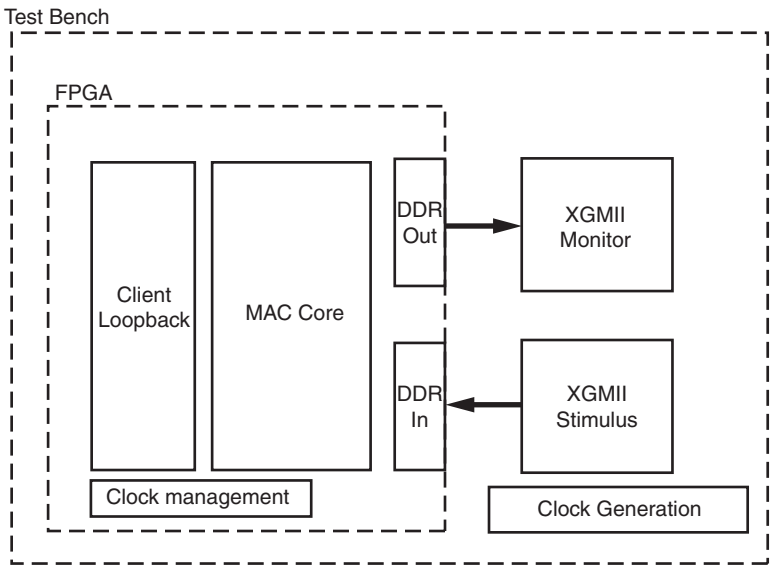

Figure 12-1: **Example Design and Test Bench**

- The 10-Gigabit Ethernet MAC core netlist
- An example HDL wrapper/top level
- A *ping* client-side loopback circuit including asynchronous FIFO that returns received frames on the transmit port
- A demonstration test bench to exercise the example design
- An AXI4-Lite-to-IPIF bridge for the management interface of the core.

The 10-Gigabit Ethernet MAC example design has been tested with Xilinx® ISE® Design Suite v14.1, Mentor Graphics ModelSim, Cadence Incisive Enterprise Simulator (IES), and Synopsys VCS and VCS MX. For the supported version of the tools see the ISE Design Suite 14: Release Notes Guide.

Generating the Core

To begin working with the example design, start by generating the core with the default settings.

To generate the core:

1. Start the Xilinx CORE Generator™ tool.

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

2. Create a new project.
3. In the Project Options dialog box, select a silicon family that supports the 10-Gigabit Ethernet MAC core for example, Virtex®-6 FPGAs.
4. In the Generate section of the Project Options, select either VHDL or Verilog, and then select Other for the Vendor.
5. Locate the 10-Gigabit Ethernet MAC core in the taxonomy tree, displayed under Communications & Networking/Ethernet; then double-click the core to open the main GUI screen.
6. A dialog box warning of the limitations of the Simulation Only Evaluation license might appear; click OK to continue. The 10-Gigabit Ethernet MAC customization screen appears.

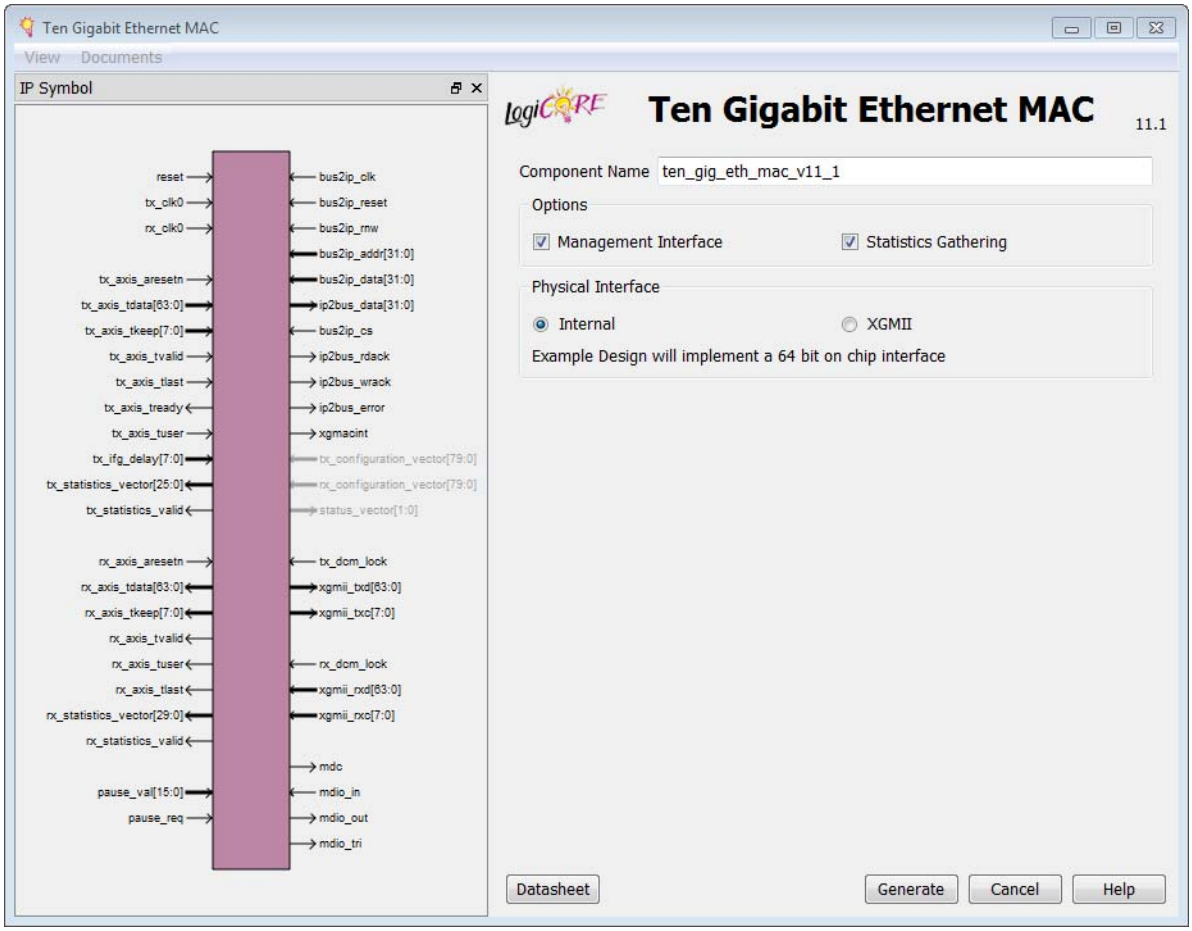

Figure 12-2: **10-Gigabit Ethernet MAC Customization Screen**

7. In the Component Name field, enter a name for the core instance. For this example, the name *quickstart* is used.
8. Accept the default settings; then click Finish to generate the core.

The core and its supporting files, including the example design, are generated in the project directory. For a detailed description of the design example files and directories, see Chapter 13, Detailed Example Design.

Implementation

After the core is successfully generated, the netlist and example design HDL wrapper can be processed through the Xilinx implementation toolset. Included in the generated outputs are several scripts to assist in this processing.

To implement the example design:

Open a command prompt or shell in your project directory, then enter these commands for either Linux or Windows platforms:

Linux

```
% cd quickstart/implement
% ./implement.sh
```

Windows

```
> cd quickstart\implement
> implement.bat
```

This starts a script that synthesizes the example design HDL wrapper, builds, maps, and places-and-routes the example design, and then creates gate-level netlist HDL files in both VHDL and Verilog with associated timing information (SDF) files. Finally these files are copied into the test area directories. This process occurs only when using the Full System Hardware Evaluation or Full licenses.

Simulation

The example design provided with the 10-Gigabit Ethernet MAC core provides a complete environment which allows you to simulate the core and view the outputs. Scripts are provided for pre- and post-layout simulation. The simulation model is either in VHDL or Verilog depending on the CORE Generator Design Entry project option.

Setting up for Simulation

The Xilinx UNISIM and SIMPRIM libraries must be mapped into the simulator. If the UNISIM and SIMPRIM libraries are not set up for your environment, see [Ref 5] for help on compiling Xilinx simulation models and setting up the simulator environment.

Pre-implementation Simulation

To run a functional simulation of the example design:

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

   ```
   quickstart/simulation/functional
   ```

2. Launch the simulation script:

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

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

Post-implementation Simulation

To run a timing simulation of the example design:

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

   ```
   quickstart/simulation/timing
   ```

2. Launch the simulation script:

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

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

XILINX®

Chapter 13

Detailed Example Design

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

- <project directory>
  Top-level project directory; name is user-defined.
  - <project directory>/<component name>
    Core release notes file
    - <component name>/doc
      Product documentation
    - <component name>/example design
      - example_design/axi_ipif
        Files for the AXI-Lite to IPIF Bridge instanced in the <component_name>_block
      - example_design/fifo
        Files for the FIFO instanced in the client_loopback example design
    - <component name>/implement
      Implementation script files
      - implement/results
        Results directory, created after implementation scripts are run, and contains implement script results
    - <component name>/simulation
      Simulation scripts
      - simulation/functional
        Functional simulation files
      - simulation/timing
        Timing simulation files

Directory and File Contents

The 10-Gigabit Ethernet MAC core directories and their associated files are defined in the following sections.

Note: The implement and timing simulation directories are only present when the core is generated with a Full or Hardware Evaluation license.

<project directory>

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

Table 13-1: **Project Directory**

Name	Description
<project_dir>	
<component_name>.ngc	Binary Xilinx implementation netlist. Describes how the core is to be implemented. Used as input to the Xilinx Implementation Tools.
<component_name>.v[hd]	VHDL structural simulation model. File used to support VHDL functional simulation of a core. The VHDL model passes customized parameters to the generic core simulation model.
<component_name>.xco	As an output file, the XCO file is a log file which records the settings used to generate a particular core. An XCO file is generated by the CORE Generator tool for each core that it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator tool.
<component_name>.xcp	As an output file, similar to the XCO file, except that it does not specify project-specific settings such as target architecture and output products.
<component_name>_flist.txt	Text file listing all of the output files produced when customized core was generated in the CORE Generator tool.
<component_name>.{veo|vho}	VHDL or Verilog instantiation template. This can be copied into the user design.

Back to Top

<project directory>/<component name>

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

Table 13-2: **Component Name Directory**

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

Back to Top

<component name>/doc

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

Table 13-3: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
ds813_ten_gig_eth_mac.pdf	10-Gigabit Ethernet MAC Data Sheet
ug773_ten_gig_eth_mac.pdf	10-Gigabit Ethernet MAC User Guide

Back to Top

<component name>/example design

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

Table 13-4: **Example Design Directory**

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_example_design.v[hd]	Example design level VHDL or Verilog file for the example design. The fifo_block module is instanced along with the address swap block, if required.
<component_name>_example_design.ucf	UCF for the core and the example design.
<component_name>_fifo_block.v[hd]	FIFO and Block level VHDL/Verilog file. For cores without a simplex split, this instances the AXI4-Stream FIFO in addition to the block level.
<component_name>_block.v[hd]	Block level VHDL/Verilog file. This instances the appropriate interface block and the MAC core.
address_swap.v[hd]	Address swap VHDL or Verilog file. This connects to the AXI4-Stream interface and swaps the destination and source addresses of frame.

Table 13-4: **Example Design Directory *(Cont'd)***

Name	Description
xgmii_if.v[hd]	XGMII interface VHDL or Verilog file. This contains the DDR registers to implement the external XGMII interface. See 10-Gigabit Ethernet MAC with External XGMII Interface, page 107.
physical_if.v[hd]	PHY interface VHDL or Verilog file. This contains the SDR registers to implement the 64-bit interface. See 10-Gigabit Ethernet MAC with 64-bit SDR Interface, page 109.
client_loopback.v[hd]	Client loopback design example instanced in the FIFO and block level.

Back to Top

example_design/axi_ipif

This directory contains files for the AXI4-Lite to IPIF Bridge instanced in the <component_name>_block.

Table 13-5: **axi_ipif Directory**

Name	Description
<project_dir>/<component_name>/example_design/axi_ipif	
address_decoder.v[hd]	Address decoder VHDL or Verilog file.
axi_lite_ipif.v[hd]	AXI4-Lite-to-IPIF bridge VHDL or Verilog file. Top level for the function.
axi_lite_ipif_wrapper.v[hd]	VHDL or Verilog wrapper to map permitted generics/parameters.
counter_f.v[hd]	Parameterizable N bit counter VHDL or Verilog file.
ipif_pkg.vhd	VHDL package containing component declarations for the AXI4-Lite to IPIF bridge.
pselect_f.v[hd]	Peripheral select VHDL or Verilog file.
slave_attachment.v[hd]	AXI4-Lite Slave function VHDL or Verilog file.

Back to Top

example_design/fifo

This directory contains the files for the FIFO instanced in the client_loopback example design.

Table 13-6: **FIFO Directory**

Name	Description
<project_dir>/<component_name>/example_design/fifo	
xgmac_fifo.v[hd]	Top-level of the FIFO.
xgmac_fifo_pack.vhd	Component declarations for the FIFO.
axi_fifo.v[hd]	Generic AXI4-Stream Fifo. This implements the logic to store and forward a good frame and drop a bad frame on AXI4-Stream interface.
fifo_ram.v[hd]	Block RAM wrapper used by both FIFOs.

Back to Top

<component name>/implement

The implement directory contains the core implementation script files.

Table 13-7: **Implement Directory**

Name	Description
<project_dir>/<component_name>/implement	
implement.sh	Linux shell script that processes the example design through the Xilinx tool flow.
implement.bat	Windows batch file that processes the example design through the Xilinx tool flow.
xst.prj	XST project file for the example design; it enumerates all the HDL files that need to be synthesized.
xst.scr	XST script file for the example design.

Back to Top

implement/results

This directory is produced by the implement scripts and is used to run the example design files and the `<component_name>.ngc` file through the Xilinx implementation tools. After these are run, this directory contains the following files for timing simulation.

Table 13-8: **Results Directory**

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

Back to Top

<component name>/simulation

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

Table 13-9: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
demo_tb.v[hd]	VHDL or Verilog demonstration test bench for the 10-Gigabit Ethernet MAC core. More information can either be found in 10-Gigabit Ethernet MAC with External XGMII Interface in Chapter 13 or 10-Gigabit Ethernet MAC with 64-bit SDR Interface in Chapter 13.

Back to Top

simulation/functional

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

Table 13-10: **Functional Directory**

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

Back to Top

simulation/timing

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

Table 13-11: **Timing Directory**

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

Back to Top

Implementation and Test Scripts

Implementation Script

The implementation script is either a shell script or batch file that processes the example design through the Xilinx tool flow. It is located in one of the following directories, depending on the platform:

Linux

project_dir/*component_name*/implement/implement.sh

Windows

project_dir/*component_name*/implement/implement.bat

Full System Hardware Evaluation or Full License Implement Script

If the core is generated with the Full System Hardware Evaluation license or Full license, the implement script performs these steps:

- The example HDL wrapper and client loopback logic is synthesized using XST
- ngdbuild is run to consolidate the core netlist and the wrapper netlist into the NGD file containing the entire design
- The design is mapped to the target technology
- The design is place-and-routed on the target device
- Static timing analysis is performed on the routed design using trce
- A bitstream is generated
- netgen runs on the routed design to generate VHDL and Verilog netlists and timing information in the form of SDF files
- These files are copied into the <component name>/implement/results directory

Simulation Only Evaluation License Implement Script

If the core is generated with the Simulation Only Evaluation license, no implement directory or script is generated.

Simulation Scripts

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

Functional

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

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

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

Timing

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

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

<project_dir>/<component_name>/simulation/timing/simulate_vcs.sh

The scripts perform these tasks:

- Compiles the gate-level netlist
- Compiles the demonstration test bench
- Starts a simulation of the test bench (with timing information if a Full System Evaluation license or Full license is in use)
- Opens a Wave window and adds some interesting signals (wave.do)
- Runs the simulation to completion

10-Gigabit Ethernet MAC with External XGMII Interface

Note: External XGMII interface is not supported on Spartan®-6 FPGA designs.

Example Design and HDL Wrapper

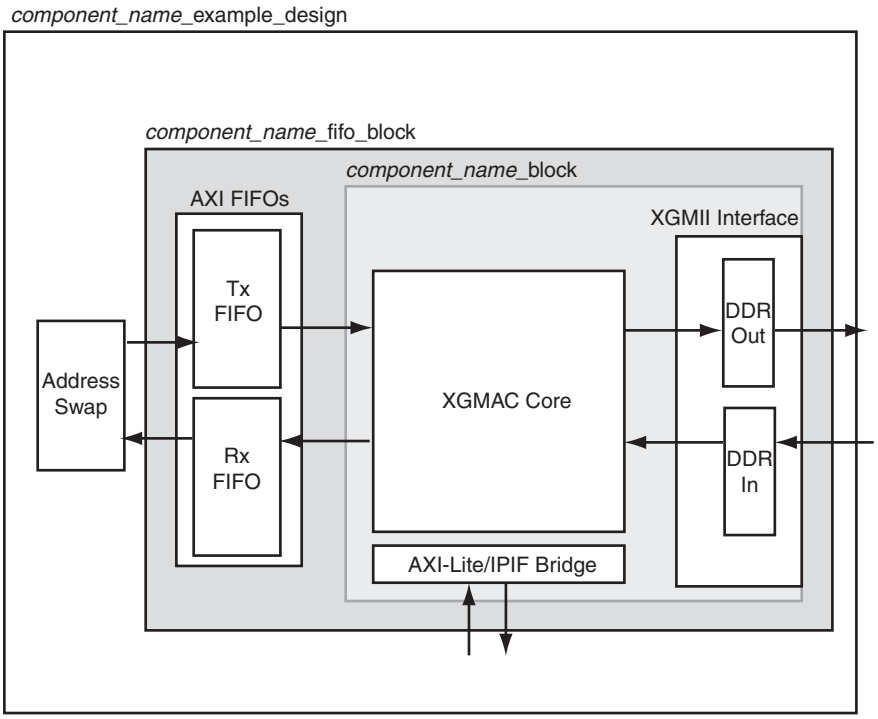

Figure 13-1: **Example Design and HDL Wrapper for 10-Gigabit Ethernet MAC with XGMII Interface**

The example design and HDL wrapper contain the following:

- Global clock buffers and Digital Clock Managers (DCMs) or Multi-Mode Clock Managers (MMCMs)
- HDL sources for client loopback design

The client loopback design performs these functions:

- Drops frame marked as bad by the core
- Crosses clock domain from received clock to transmit clock safely using an asynchronous FIFO

The address swap module performs this function:

- Swaps the destination and source address field in the received Ethernet frame

The XGMII block performs these functions:

- Receiver DCM/MMCM and clock buffer
- DDR logic for the XGMII Interface
- AXI-Lite to IPIF bridge logic

Demonstration Test Bench

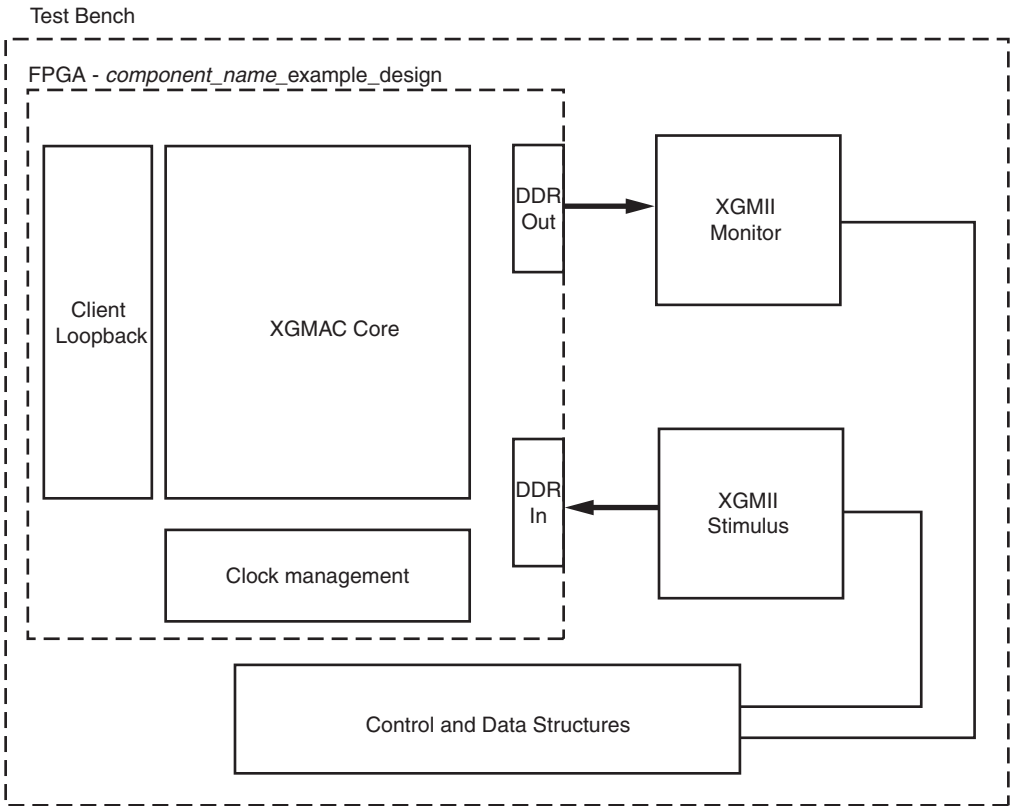

Figure 13-2: **Demonstration Test Bench for 10-Gigabit Ethernet MAC with XGMII Interface**

The demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core. It consists of transactor procedures or tasks that connect to the PHY-side ports of the example design, and a control program that pushes frames of varying length and content through the design and checks the values as they exit the core.

Because the address swap design swaps the destination and source field, the CRC is different on the outbound frame compared to that injected into the receiver. This is taken into account by the test bench when checking the transmitted data.

10-Gigabit Ethernet MAC with 64-bit SDR Interface

Example Design and HDL Wrapper

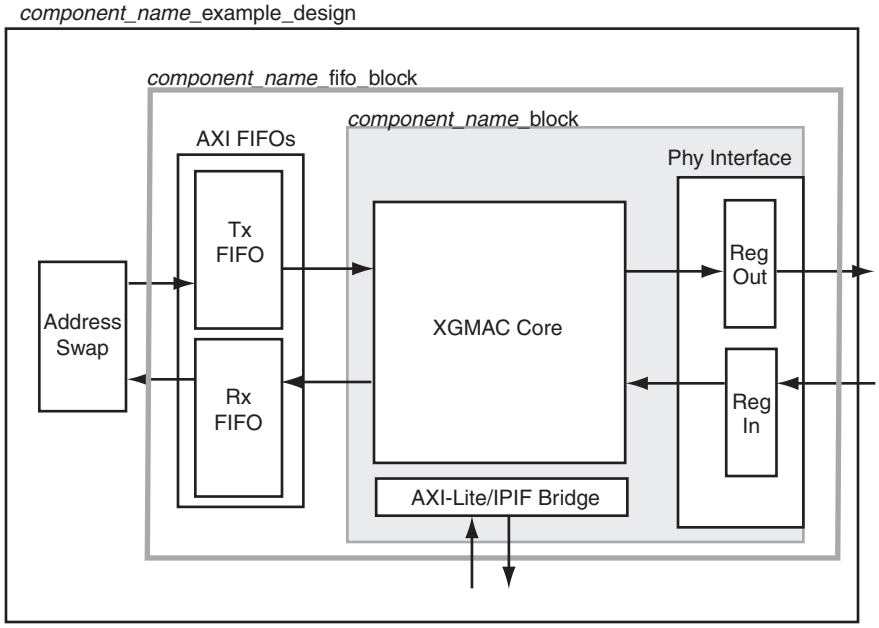

Figure 13-3: **Example Design and HDL Wrapper for 10-Gigabit Ethernet MAC with 64-bit Interface**

The example design and HDL wrappers contain the following:

- Global clock buffers and Digital Clock Managers (DCMs) or Multi-Mode Clock Managers (MMCMs)
- HDL sources for client loopback design

The client loopback design performs these functions:

- Drops frame marked as bad by the 10-Gigabit Ethernet MAC core
- Crosses clock domain from received clock to transmit clock safely using an asynchronous FIFO

The address swap module performs this function:

- Swaps the destination and source address field in the received Ethernet frame

The physical interface block performs these functions:

- Registers for the 64-bit SDR interface
- Receiver DCM/MMCM and clock buffer
- AXI4-Lite to IPIF bridge logic

Demonstration Test Bench

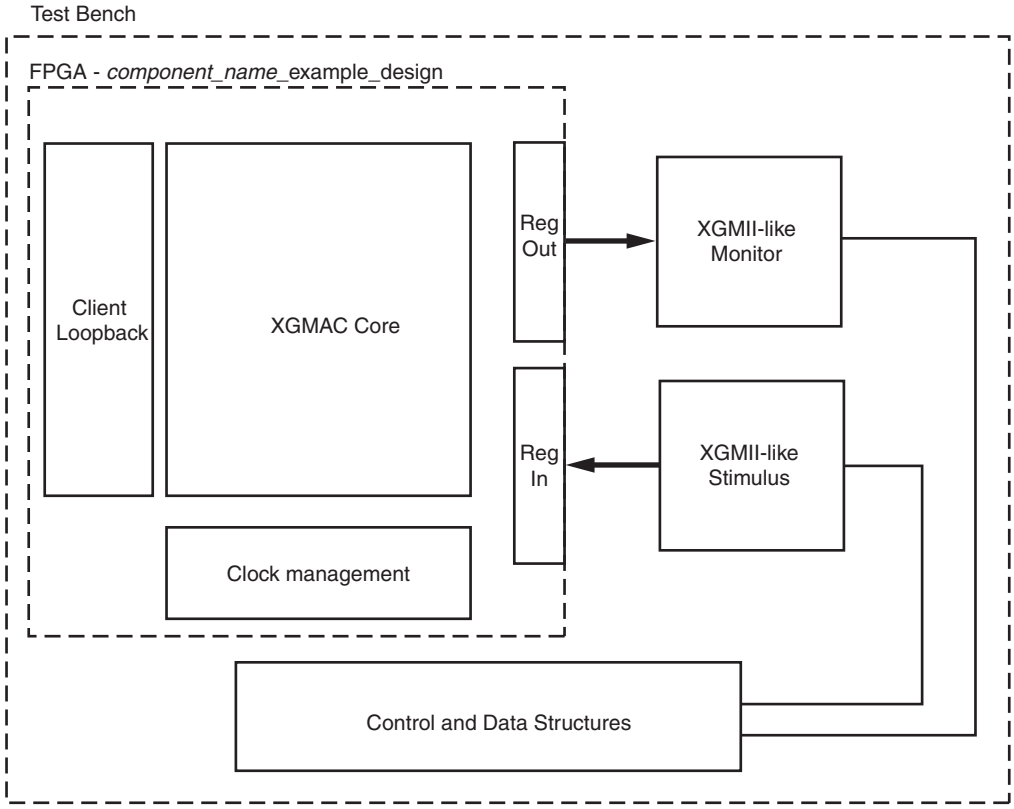

Figure 13-4: **Demonstration Test Bench for 10-Gigabit Ethernet MAC with 64-bit Interface**

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

Because the address swap design swaps the destination and source field, the CRC is different on the outbound frame compared to that injected into the receiver. This is taken into account by the test bench when checking the transmitted data.

AXI4-Lite to IPIF Converter Block

AXI4-Lite to IPIF converter block is used in the example design at block level to convert the AXI4-Lite transactions to IPIF transactions which is the actual management interface of the core. Table 13-12 describes the ports associated with the IPIF interface.

Table 13-12: **IPIF Interface Port Description**

Name	Direction	Description
bus2ip_clk	IN	IPIF Clock.
bus2ip_reset	IN	IPIF Reset.
bus2ip_cs	IN	IPIF Chip select, qualifies the bus2ip_rnw signal.
bus2ip_rnw	IN	IPIF read not write signal. When this signal is 0 along with bus2ip_cs being 1 during clock positive edge Write operation is requested. When this signal is 1 along with bus2ip_cs being 1 during clock positive edge Read operation is requested.
bus2ip_data[31:0]	IN	IPIF Write data to the core.
ip2bus_data[31:0]	OUT	IPIF Read data from the core.
bus2ip_addr[31:0]	IN	Address of the register accessed.
ip2bus_rdack	OUT	Read Transaction acknowledge signal
ip2bus_wrack	OUT	Write Transaction acknowledge signal
ip2bus_error	OUT	Error on invalid transaction, asserted along with the read/write acknowledge signal.

Figure 13-5 explains the timing of IPIF read and write transactions.

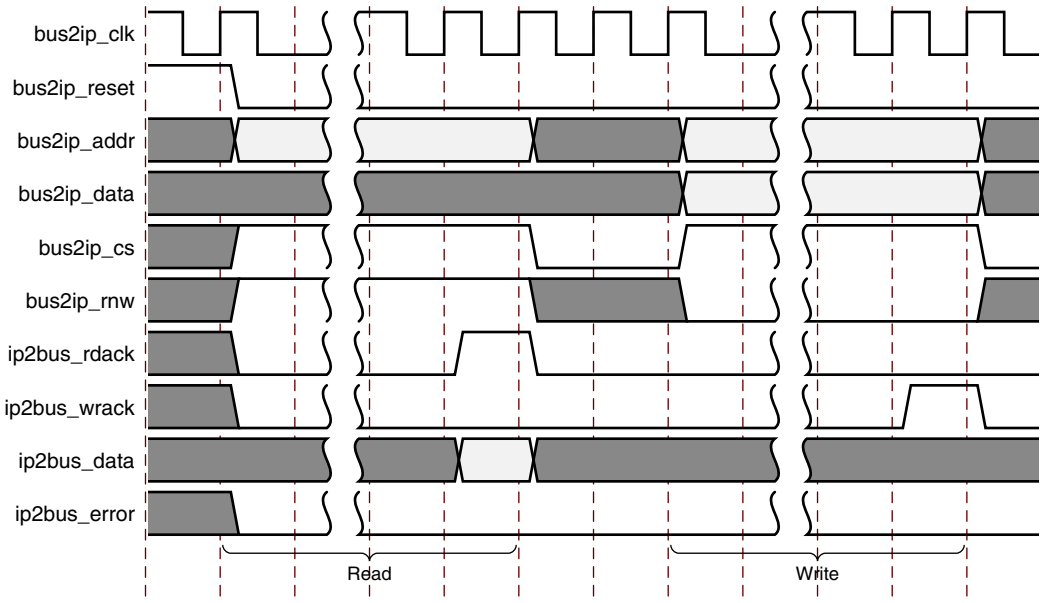

Figure 13-5: **Timing of IPIF Read and Write Transactions**

XILINX®

Appendix A

Verification and Interoperability

The 10-Gigabit Ethernet MAC core has been verified using simulation.

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

- Configuration register access through Management Interface
- Local Fault and Remote Fault handling
- Frame transmission
- Frame reception
- CRC validity
- Handling of CRC errors
- Statistic counter access through Management Interface and validity of counts
- Statistic vector validity
- Initiating MDIO transactions through Management Interface

The core has also been hardware validated on Virtex®-6 and Spartan®-6 based platforms, alongside the Xilinx® XAUI core. The design comprises the MAC, XAUI, a loopback FIFO and test pattern generator all under embedded MicroBlaze™ processor control.

XILINX®

Appendix B

Calculating the DCM Fixed Phase-Shift Value

Requirement for DCM Phase-Shifting

A DCM is used in the receiver 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 receiver clock DCM to skew the clock; this performs static alignment by using the receiver clock DCM to shift the internal version of the receiver 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 receiver 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 families. Xilinx does not recommend attempting to determine the phase-shift setting empirically. In addition to the clock-to-data phase relationship, other factors such as package flight time (package skew) and clock routing delays (internal to the device) affect the clock to data relationship at the sample point (in the IOB) and are difficult to characterize.

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

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, etc.) 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.

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.

XILINX®

Appendix C

Core Latency

These measurements are for the core only; they do not include the latency through the example design FIFO or IOB registers.

Transmit Path Latency

As measured from the input port `tx_axis_tdata` of the AXI4-Stream Transmit interface (until that data appears on `xgmii_txd` on the PHY-side interface), the latency through the core in the transmit direction is 15 clock periods of the `tx_clk0`.

Receive Path Latency

Measured from the `xgmii_rxd` port on the PHY-side Receive interface (until the data appears on the `rx_axis_tdata` port of the receiver side AXI4-Stream interface), the latency through the core in the receive direction is 14 clock periods of `rx_clk0`. This can increase to 15 clock periods if the core needs to modify the alignment of data at the AXI4-Stream Receive interface.

XILINX®

Appendix D

Additional Resources

Xilinx Resources

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

www.xilinx.com/support.

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

www.xilinx.com/company/terms.htm

References

1. XST User Guide (ISE documentation)
2. LogiCORE™ IP Ten Gigabit Ethernet PCS/PMA User Guide (UG692)
3. LogiCORE IP XAUI User Guide (UG150)
4. LogiCORE IP RXAUI User Guide (UG693)
5. Xilinx Synthesis and Simulation Design Guide
6. IEEE Standard 802.3-2008, “Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.”