LogiCORE™ IP 10-Gigabit Ethernet MAC v10.2

User Guide

UG148 March 1, 2011

XILINX®

Revision History

The following table shows the revision history for this document..

Date	Version	Revision
09/30/04	1.0	Initial Xilinx release.
04/28/05	1.1	Update to v6.0 of core, support for Virtex®-4, update to Xilinx® tools 7.1i.
01/18/06	1.2	Update to v7.0 of core, Xilinx tools 8.1i, licensing chapter, and release date.
07/13/06	1.3	Update to Xilinx tools 8.2i, core version 8.0.
09/21/06	1.4	Update to core version 8.1.
02/15/07	1.5	Update to core version 8.2, Xilinx tools 9.1i.
08/08/07	1.6	Update to core version 8.3, Xilinx tools 9.2i.
3/24/08	1.7	Updated tool support and core version for IP0K release.
6/27/08	8.5	Changed version number in Revision History table to match core release version.
9/18/08	8.6	Updated core to version 8.6; added support for Virtex-5 TXT FPGAs.
04/24/09	9.1	Update to core version 9.1, Xilinx tools 11.1. Added Virtex-6 device.
06/24/09	9.2	Update to core version 9.2, Xilinx tools 11.2.
09/16/09	9.3	Update to core version 9.3, Xilinx tools 11.3. Added Virtex-6 HXT and Virtex-6 -1L support, Virtex-6 CXT.
04/19/10	10.1	Update to core version 10.1, Xilinx tools 12.1. Added Spartan®-6 support.
03/01/11	10.2	Update to core version 10.2, Xilinx tools 13.1.

Table of Contents

Chapter 4: Customizing and Generating Core

Chapter 5: Designing with the Core

Chapter 6: Interfacing to the Core: Data Interfaces

Chapter 7: Interfacing to the Core: Control Interfaces, Statistics and MDIO

Chapter 8: Using Flow Control

Chapter 9: Constraining the Core

Chapter 10: Special Design Considerations

Chapter 11: Implementing Your Design

Chapter 12: Quick Start Example Design

Chapter 13: Detailed Example Design

Schedule of Tables

Schedule of Figures

Preface: About This Guide

Chapter 1: Introduction

Chapter 2: Licensing the Core

Chapter 3: Core Architecture

Chapter 4: Customizing and Generating Core

Chapter 5: Designing with the Core

Chapter 6: Interfacing to the Core: Data Interfaces

Chapter 7: Interfacing to the Core: Control Interfaces, Statistics and MDIO

Chapter 8: Using Flow Control

Chapter 9: Constraining the Core

Chapter 10: Special Design Considerations

Chapter 11: Implementing Your Design

Chapter 12: Quick Start Example Design

Chapter 13: Detailed Example Design

About This Guide

The *10-Gigabit Ethernet MAC User Guide* provides information about generating a LogiCORE™ IP 10-Gigabit Ethernet MAC core, customizing and simulating the core utilizing the provided example design, and running the design files through implementation using the Xilinx® tools.

Guide Contents

This guide contains the following chapters:

- Preface, About this Guide introduces you to the organization and purpose of the design guide and the conventions used in this document.
- Chapter 1, Introduction introduces the 10-Gigabit Ethernet MAC core. It describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, Licensing the Core provides instructions for licensing the core.
- Chapter 3, Core Architecture describes the overall architecture of the 10-Gigabit Ethernet MAC core and the major interfaces to the core.
- Chapter 4, Customizing and Generating Core provides instructions for generating the core using the Xilinx CORE Generator™ software.
- Chapter 5, Designing with the Core provides a general description of how to use the core in your own design.
- Chapter 6, Interfacing to the Core: Data Interfaces describes how to connect to the data interfaces of the core.
- Chapter 7, Interfacing to the Core: Control Interfaces, Statistics and MDIO describes the interfaces available for dynamically setting and querying the configuration and status of the core.
- Chapter 8, Using Flow Control describes the operation of the flow control logic of the core.
- Chapter 9, Constraining the Core describes how to constrain a design containing the core, as illustrated by the User Constraints File (UCF) delivered with the core when it is generated.
- Chapter 10, Special Design Considerations describes considerations that may apply in specific design cases.
- Chapter 11, Implementing Your Design describes how to simulate and implement your design.
- Chapter 12, Quick Start Example Design provides instructions to quickly generate the core and run the example design through implementation and simulation using the default settings.

- Chapter 13, Detailed Example Design describes the demonstration test bench in detail and provides directions for how to customize the demonstration test bench for use in an application.
- Appendix A, Verification and Interoperability identifies simulation and hardware testing verification for the 10-Gigabit Ethernet MAC core.
- Appendix B, Calculating the DCM Fixed Phase-Shift Value defines phase shifting requirements for the core.
- Appendix C, Core Latency provides the core latency data.

Additional Resources

To find additional documentation, see the Xilinx website at:

http://www.xilinx.com/support/documentation/index.htm.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

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

Conventions

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

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays. Signal names also.	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *User Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported

Convention	Meaning or Use	Example
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **bus[7:0]**, they are required.	**ngdbuild** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**lowpwr ={on|off}**
Vertical bar |	Separates items in a list of choices	**lowpwr ={on|off}**
Angle brackets < >	User-defined variable or in code samples	<directory name>
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis . . .	Repetitive material that has been omitted	**allow block** *block_name loc1 loc2 ... locn;*
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	*usr_teof_n* is active low.

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Guide Contents" for details. See "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Xilinx FPGA Designers Handbook.*
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

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.

About the Core

The 10-Gigabit Ethernet MAC core is a Xilinx CORE Generator™ software 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 following 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 support.xilinx.com/. Questions are routed to a team of engineers with expertise using the 10-Gigabit Ethernet MAC core.

Xilinx will provide technical support for use of this product as described in the *LogiCORE 10-Gigabit Ethernet MAC User Guide* and the *LogiCORE 10-Gigabit Ethernet MAC Getting Started Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the 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, please submit a WebCase from www.xilinx.com/support/clearexpress/websupport.htm. Be sure to include the following information:

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

Document

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

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

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™ Site License Agreement, which conforms to the terms of the SignOnce IP License standard defined by the Common License Consortium. 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® software 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 software.

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license, do the following:

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 software 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 will receive 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/ipcenter/ipaccess_fee.htm.

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE software 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 will be 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.

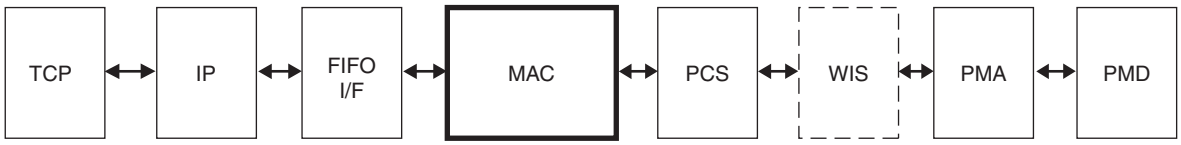


Figure 3-1: **Typical Ethernet System Architecture**

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.

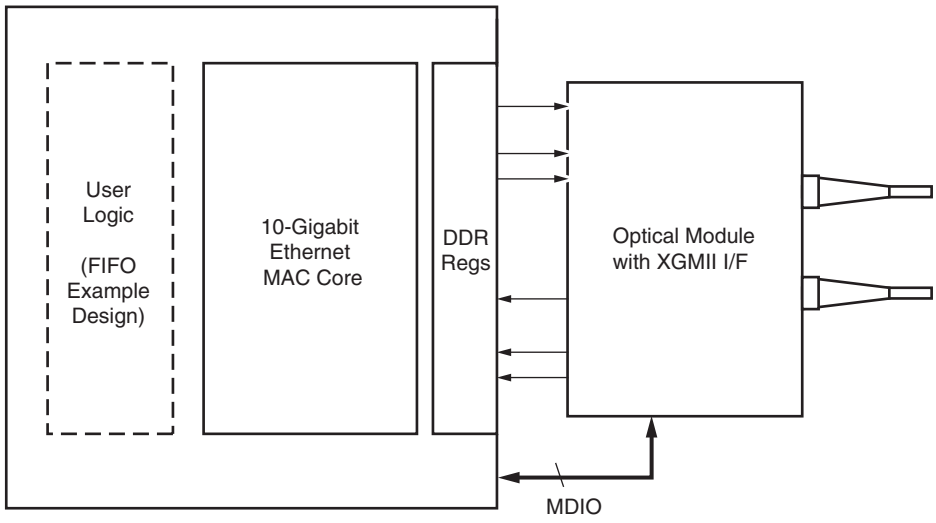


Figure 3-2: **10-Gigabit Ethernet MAC Core Connected to PHY with XGMII Interface**

The 10-Gigabit Ethernet MAC core is designed to be easily 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 fabric interface is used to interface to the XAUI core

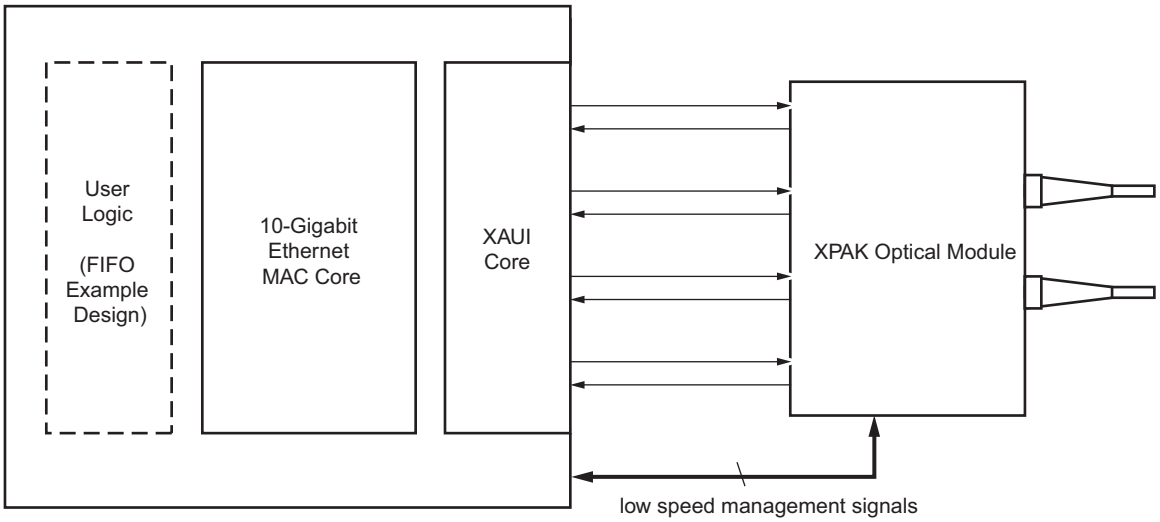


Figure 3-3: **10-Gigabit Ethernet MAC Core Used with Xilinx XAUI Core**

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

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 the following:

- Client-side 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
- Management interface and MDIO (optional)
- Statistics counters (optional)
- XGMII interface - connection to the physical layer device or logic.

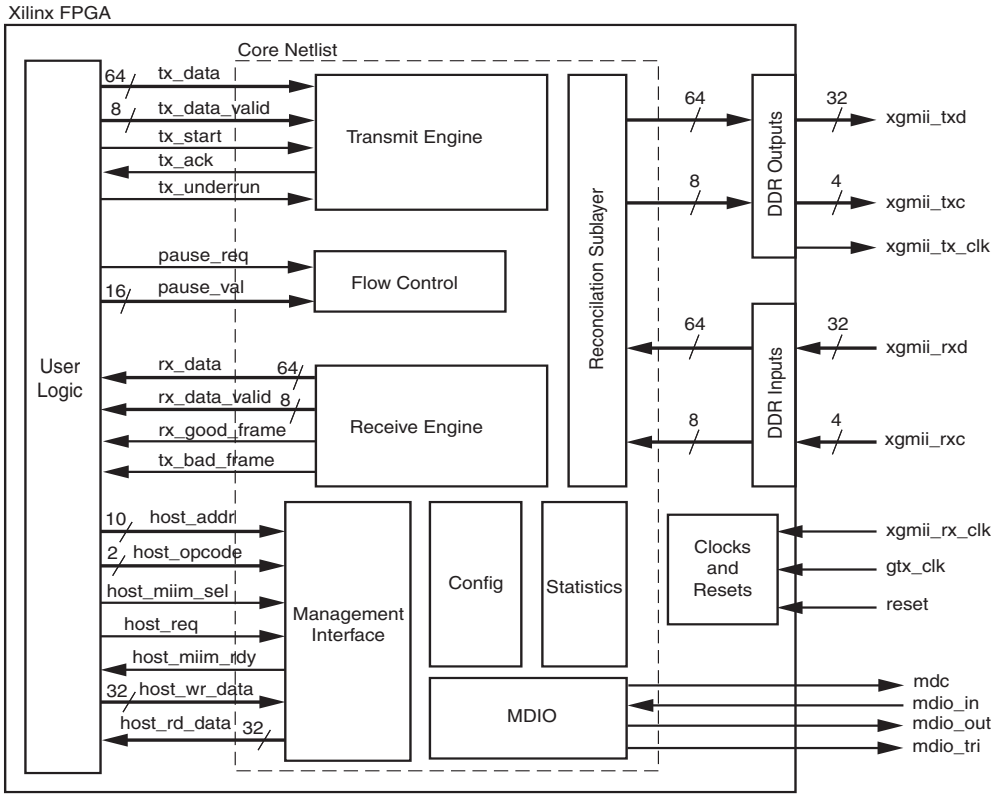


Figure 3-4: **Implementation of the 10-Gigabit Ethernet MAC Core**

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.

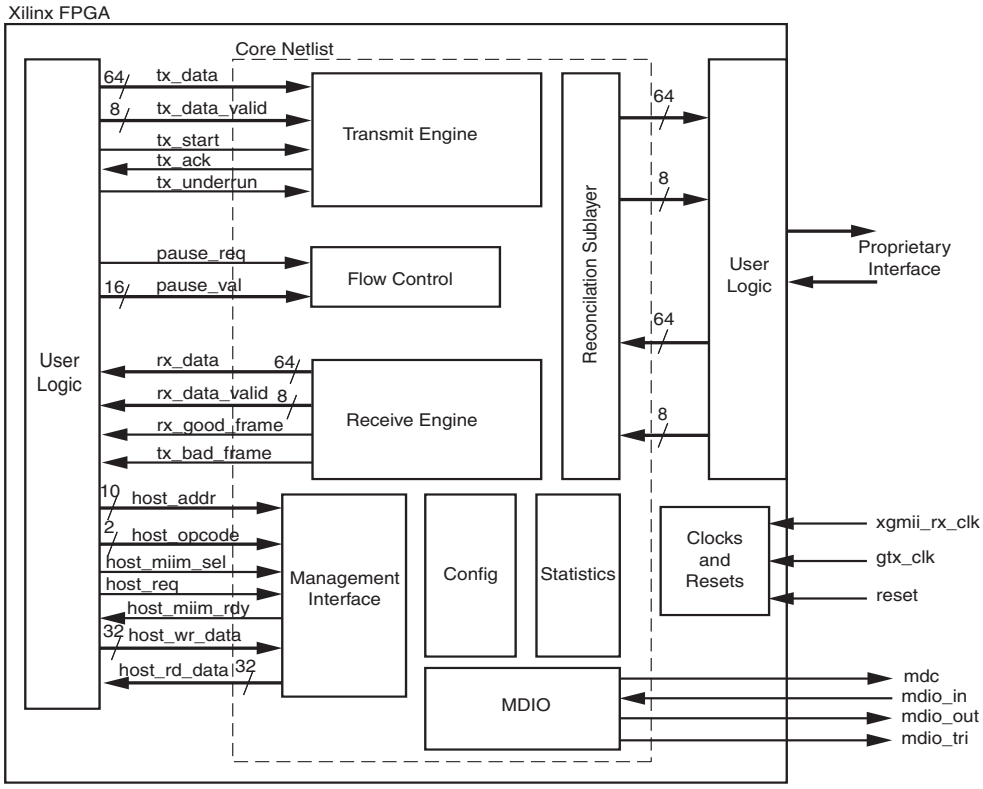


Figure 3-5: **Implementation of the Core with User Logic on PHY Interface**

Core Interfaces and Modules

Client-Side Interface - Transmit

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

Table 3-1: **Client-Side Interface Ports - Transmit**

Name	Direction	Description
tx_data[63:0]	Input	Transmit data, eight bytes wide.
tx_data_valid[7:0]	Input	Transmit control bits, one bit per transmit lane.
tx_start	Input	Transmit handshaking, signals client is ready to transmit data.
tx_ack	Output	Transmit handshaking, signals MAC is ready to accept data.
tx_underrun	Input	Transmit handshaking, signals client is unable to complete frame.
tx_ifg_delay[7:0]	Input	If IFG stretching is enabled, signals how long in XGMII columns the IFG should be.

Client-side Interface - Receive

The signals of the receive client-side interface are shown in Table 3-2. See Chapter 6, Interfacing to the Core: Data Interfaces for more information on connecting to the receive client-side interface.

Table 3-2: **Client-Side Interface Ports - Receive**

Name	Direction	Description
rx_data[63:0]	Output	Received data, eight bytes wide.
rx_data_valid[7:0]	Output	Received control bits, one bit per receive lane.
rx_good_frame	Output	Asserted at the end of frame to indicate the frame was successfully received and should be processed by the user logic.
rx_bad_frame	Output	Asserted at the end of frame to indicate the frame was not successfully received and should be discarded by the user logic.

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	Input	Request that a flow control frame is emitted from the MAC core.
pause_val[15:0]	Input	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 may 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]	Output	Transmit data to PHY
xgmii_txc[7:0]	Output	Transmit control to PHY
xgmii_rxd[63:0]	Input	Received data from PHY
xgmii_rxc[7:0]	Input	Received control from PHY

Management Interface

Configuration of the core, access to the statistics block, and access to the MDIO port is performed through the Management Interface, a 32-bit processor-neutral interface independent of the Ethernet data path. The ports associated with the Management Interface are shown in Table 3-5.

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

Name	Direction	Description
host_clk	Input	Clock for Management Interface. Range between 10 MHz and 133 MHz.
host_opcode[1:0]	Input	Defines operation to be performed over Management Interface.
host_addr[9:0]	Input	Address of register to be accessed.
host_wr_data[31:0]	Input	Data to write to register.
host_rd_data[31:0]	Output	Data read from register.
host_miim_sel	Input	When asserted, the MDIO interface is accessed.
host_req	Input	Used to request a transaction on the MDIO interface or read from the statistic registers.
host_miim_rdy	Output	When asserted, the MDIO interface has completed any pending transaction and is ready for a new transaction.

The Management Interface can be omitted at core customization stage; if omitted, `configuration_vector` 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: Control Interfaces, Statistics and MDIO for more information on this signal, including a breakdown of the configuration and status vector bits.

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

Name	Direction	Description
configuration_vector[68:0]	Input	Configuration signals for the core.
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: Control Interfaces, Statistics and MDIO for more information 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 tristate. '1' disconnects the output driver from the MDIO bus.

Statistic Vectors

In addition to the statistic counters described in Management Interface, page 29, 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™ software 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: Control Interfaces, Statistics and MDIO.

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

Name	Direction	Description
tx_statistics_vector[24:0]	Output	Aggregated statistics flags for transmitted frame.
tx_statistics_valid	Output	Valid strobe for tx_statistics_vector.
rx_statistics_vector[28:0]	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 may include Digital Clock Managers (DCMs) or Mixed-Mode Clock Managers (MMCMs), reset synchronizers, or other useful utility circuits that may 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 CORE Generator™ software. 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 software customization screen for the 10-Gigabit Ethernet MAC core.

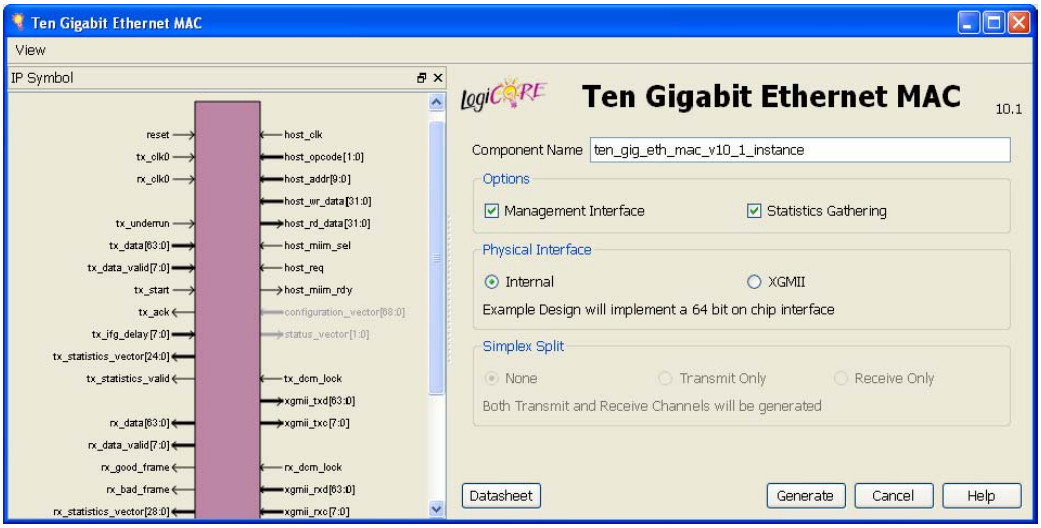


Figure 4-1: **10-Gigabit Ethernet MAC Customization Screen**

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

Component Name

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

Statistics Gathering

This checkbox selects whether the statistics counters will be 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.

Simplex Split

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

When Simplex Split is used, the Management Interface and Statistics Gathering are disabled. The MAC can only be configured using the Configuration Vector (See The Configuration and Status Vector, page 78). The Configuration Vector contains bit for both Transmit and Receive. When Receive Only is selected, the bits in the Configuration Vector which control the Transmitter are not used and need not be driven. The bit positions remain unchanged. The Receiver bits need not be driven if Transmit Only is selected. The Pause MAC Address is used by both the transmitter and receiver and should be driven in both cases.

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

Parameter Values in the XCO File

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

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

```
SELECT Ten_Gigabit_Ethernet_MAC Virtex5 Xilinx,_Inc. 10.2
CSET physical_interface = XGMII
CSET statistics_gathering = true
CSET component_name = the_core
CSET simplex_split = None
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
simplex_split	None, transmit_only, receive_only	None

Output Generation

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

- The netlist files for the core
- XCO files
- Release notes and documentation
- An HDL example design
- Scripts to synthesize, implement and simulate the example design

See the *LogiCORE IP 10-Gigabit Ethernet MAC Getting Started Guide* for a complete description of the CORE Generator software 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: Control Interfaces, Statistics and MDIO, 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 CORE Generator™ software 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 the *LogiCORE 10-Gigabit Ethernet MAC Getting Started Guide* 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 may not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx® tools to place and route the design.

Recognize Timing Critical Signals

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

Use Supported Design Flows

The core is synthesized in the CORE Generator software and is delivered to you as an NGC and EDIF netlist. The example implementation scripts provided currently use XST 12.1 as the synthesis tool for the HDL example design that is delivered with the core. Other synthesis tools may be used for the user application logic; the core will always be unknown to the synthesis tool and should appear as a black box.

Post-synthesis, only ISE® software v12.1 and higher tools are supported.

Make Only Allowed Modifications

The 10-Gigabit Ethernet MAC core is not user-modifiable. Do not make modifications as they may have adverse effects on system timing and protocol compliance. Supported user configurations of the 10-Gigabit Ethernet MAC core can only be made by the selecting the options from within the CORE Generator software 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 Client-Side Interface

The client-side interface on transmit has a 64-bit data path with 8 control bits to delineate bytes within the 64-bit port. Additionally, there are signals to handshake the transfer of data into the core. The signals are shown in Table 6-1.

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

Name	Direction	Description
tx_data[63:0]	Input	Frame data to be transmitted is supplied on this port.
tx_data_valid[7:0]	Input	Control signals for tx_data port. Each asserted signal on tx_data_valid signifies which bytes of tx_data are valid; that is, if tx_data_valid[0] is '1,' the signals tx_data[7:0] are valid.
tx_start	Input	Handshaking signal. Asserted by the client to make data available for transmission.
tx_ack	Output	Handshaking signal. Asserted when the first column of data on tx_data has been accepted.
tx_underrun	Input	Assert this pin to forcibly corrupt the current frame.
tx_ifg_delay[7:0]	Input	Control signal for configurable interframe gap adjustment.

For tx_data (Table 6-2), the port is logically divided into lane 0 to lane 7, with the corresponding bit of the `tx_data_valid` word signifying valid data on the `tx_data` port.

Table 6-2: **tx_data Lanes**

Lane	tx_data 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 outbound frame transfer is shown in Figure 6-1. When the client wants to transmit a frame, it asserts the `tx_start` signal, and on the next clock places the first column of data and control onto the `tx_data` and `tx_data_valid` ports. `tx_data_valid` must be set to all 0s during the clock cycle that `tx_start` is asserted.

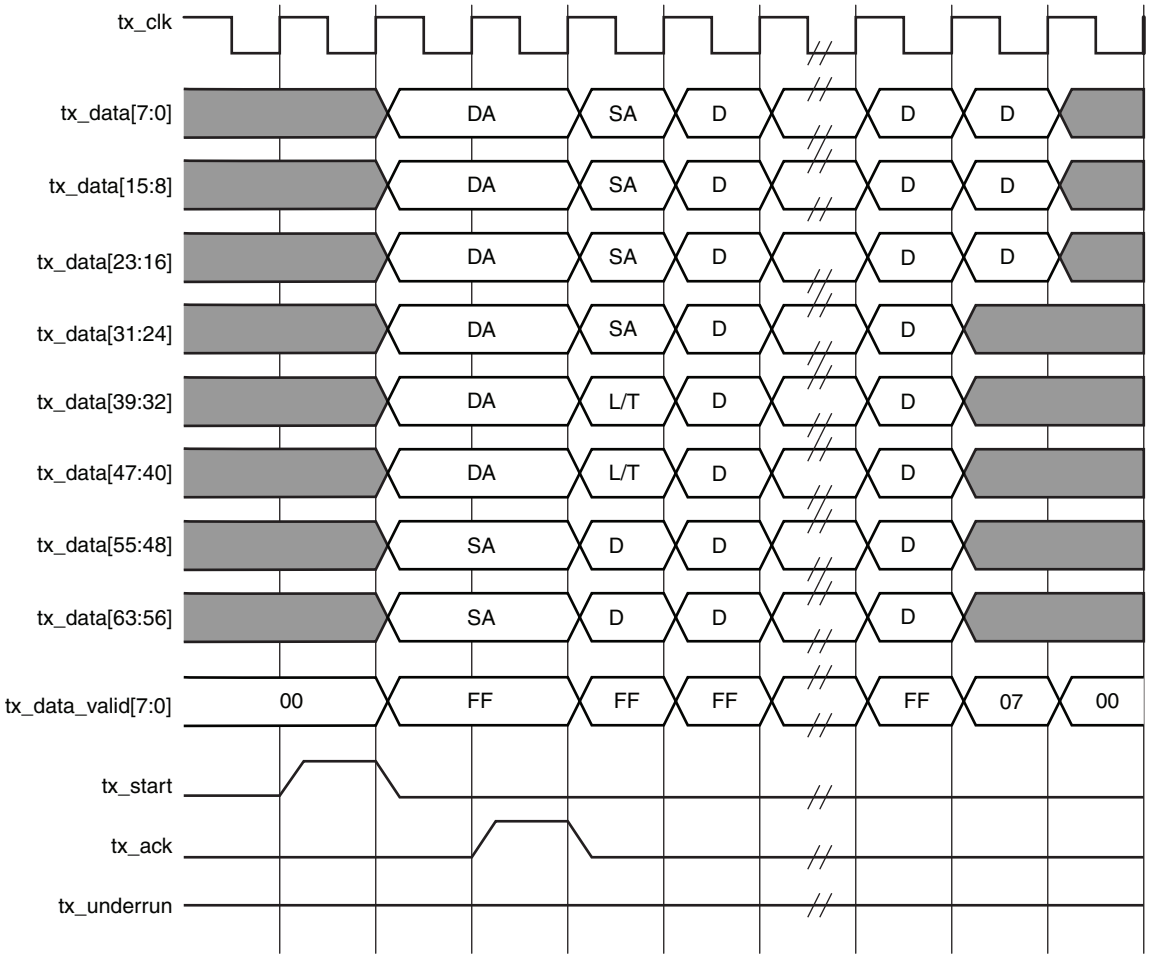


Figure 6-1: **Frame Transmission Across Client-side Interface**

When the MAC core has read this first column of data, it will assert the `tx_ack` signal; on the next and subsequent clock edges, the client must provide the remainder of the data for the frame.

The end of frame is signalled to the MAC core by having `tx_data_valid` not equal to hexadecimal "FF"; partially full columns of data are not permitted within a frame. As an example, in Figure 6-1 the value of `tx_data_valid` of hexadecimal "07" signals to the MAC core that only the lower three columns of data are valid on the transfer and additionally that the end of frame has been reached.

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 will be terminated correctly but not padded.

Transmission with In-Band FCS Passing

If the MAC core is configured to have the FCS field passed by the client (see Configuration Registers in Chapter 7), 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 will not perform any padding of the payload.

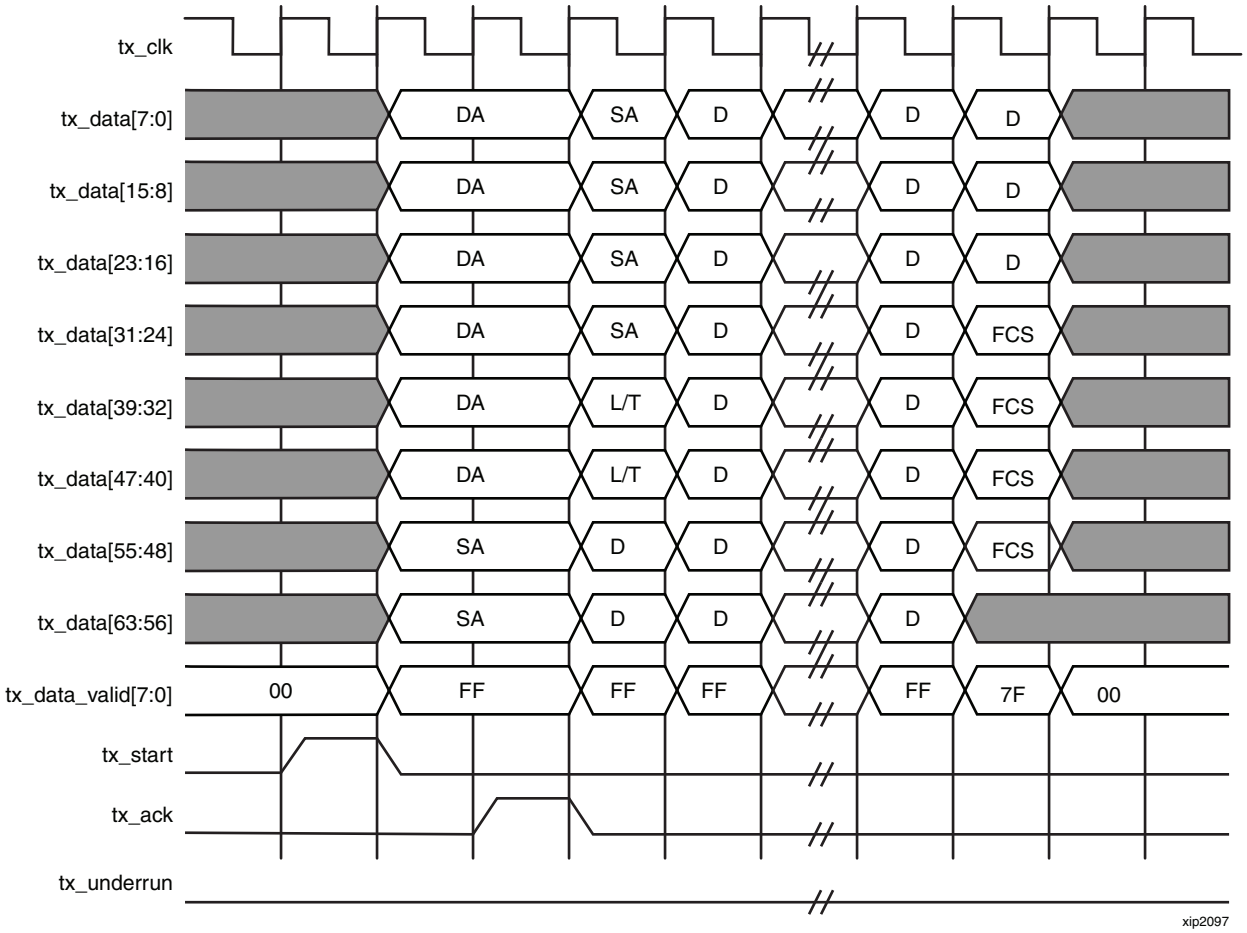


Figure 6-2: **Frame Transmission with Client-Supplied FCS**

Aborting a Transmission

The timing of an aborted transfer can be seen in Figure 6-3. This may happen, for instance, if a FIFO in the bus interface empties before a frame is completed. When the client asserts `tx_underrun` during a frame transmission, the MAC core will insert error codes into the XGMII data stream to corrupt the current frame; then the MAC core will fall back to idle transmission. It is the responsibility of the client to re-queue the aborted frame for transmission.

When an underrun occurs, `tx_start` may be asserted on the clock cycle after the `tx_underrun` assertion to start a new transmission.

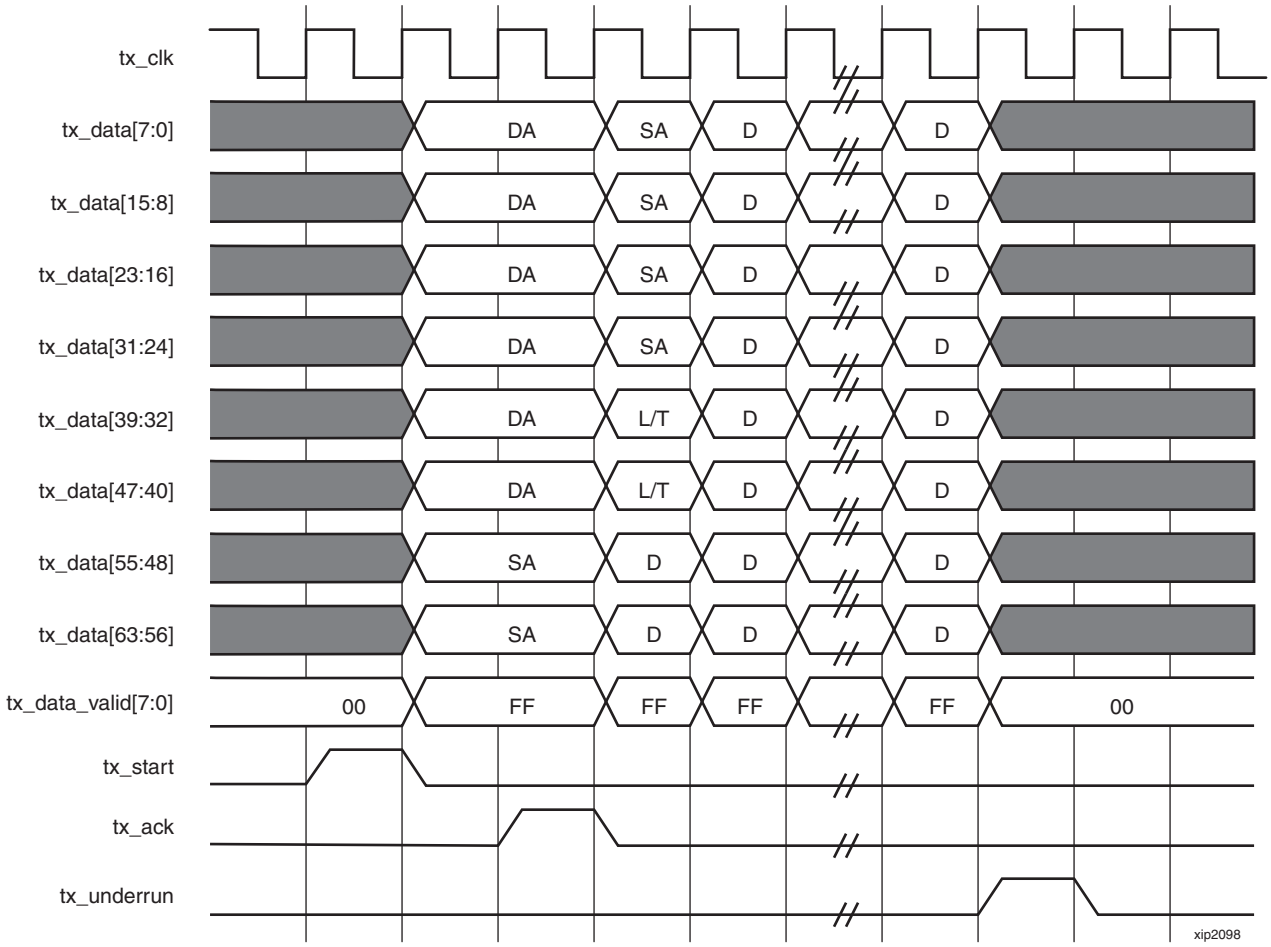


Figure 6-3: **Aborting a Frame Transmission**

Back-to-Back Transfers

Two situations can occur during back-to-back transfers; in one, the MAC core is ready to accept data, and in the other, the MAC must defer to comply with inter-packet gap requirements, a user request to extend the interframe gap or flow control requests.

Figure 6-4 shows the case where the MAC is immediately ready to accept the next frame of data. In the column after the last data is transferred for the first frame, the client asserts `tx_start` to signal that another frame is ready for transmission. The MAC core then asserts `tx_ack` to allow the client to begin the burst of frame parameters and data that make up the frame.

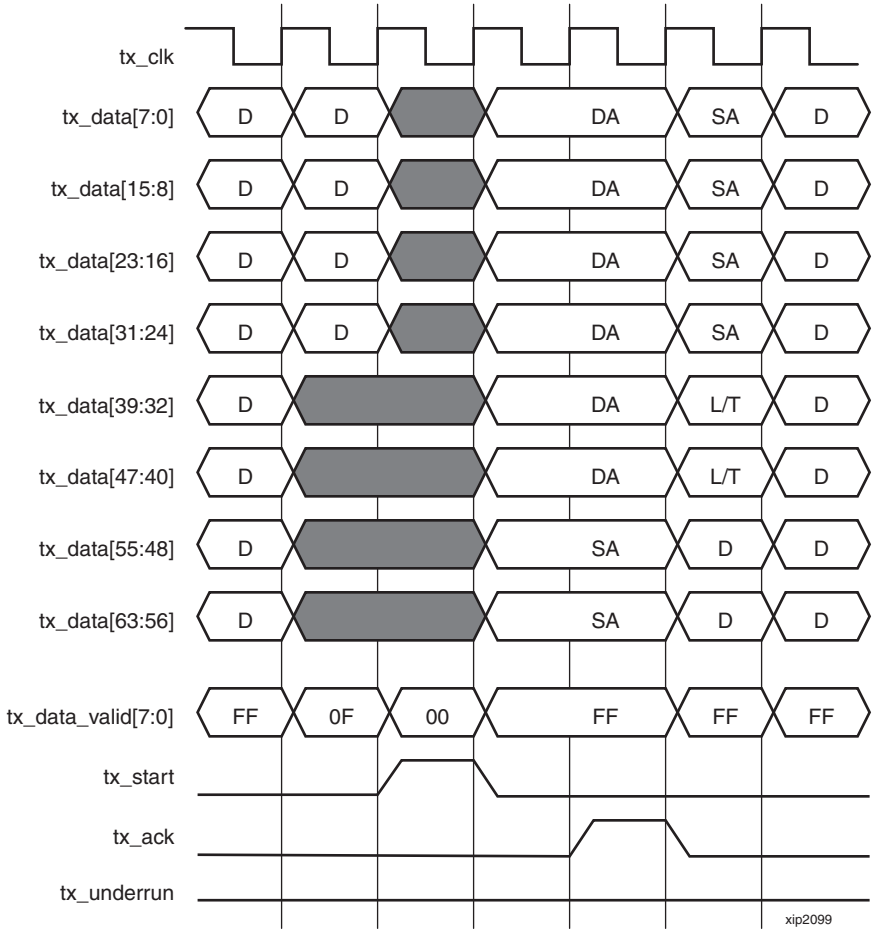


Figure 6-4: **Back-to-Back Frame Transmission, No Back Pressure**

Figure 6-5 shows a case where the MAC is exerting back pressure on the client to delay the start of transmission. In this case, the client has asserted `tx_start` to signal that another frame is ready for transmission, but the MAC core has delayed the assertion of `tx_ack` to allow the data burst to begin. After this burst starts, it continues in the same manner as in the cases described previously. In both cases, as in the case for normal frame transmission, the client must provide a whole frame of data in one burst; there is no mechanism to stop and start transfers within a single frame.

Circumstances under which this back pressure and associated delay will arise include if the interframe gap is under user control, or if the MAC is in WAN mode which increases the interframe gap to reduce the overall throughput of the core.

In both these cases, the `tx_data_valid` bus must be set to all 0s when the `tx_start` signal is asserted.

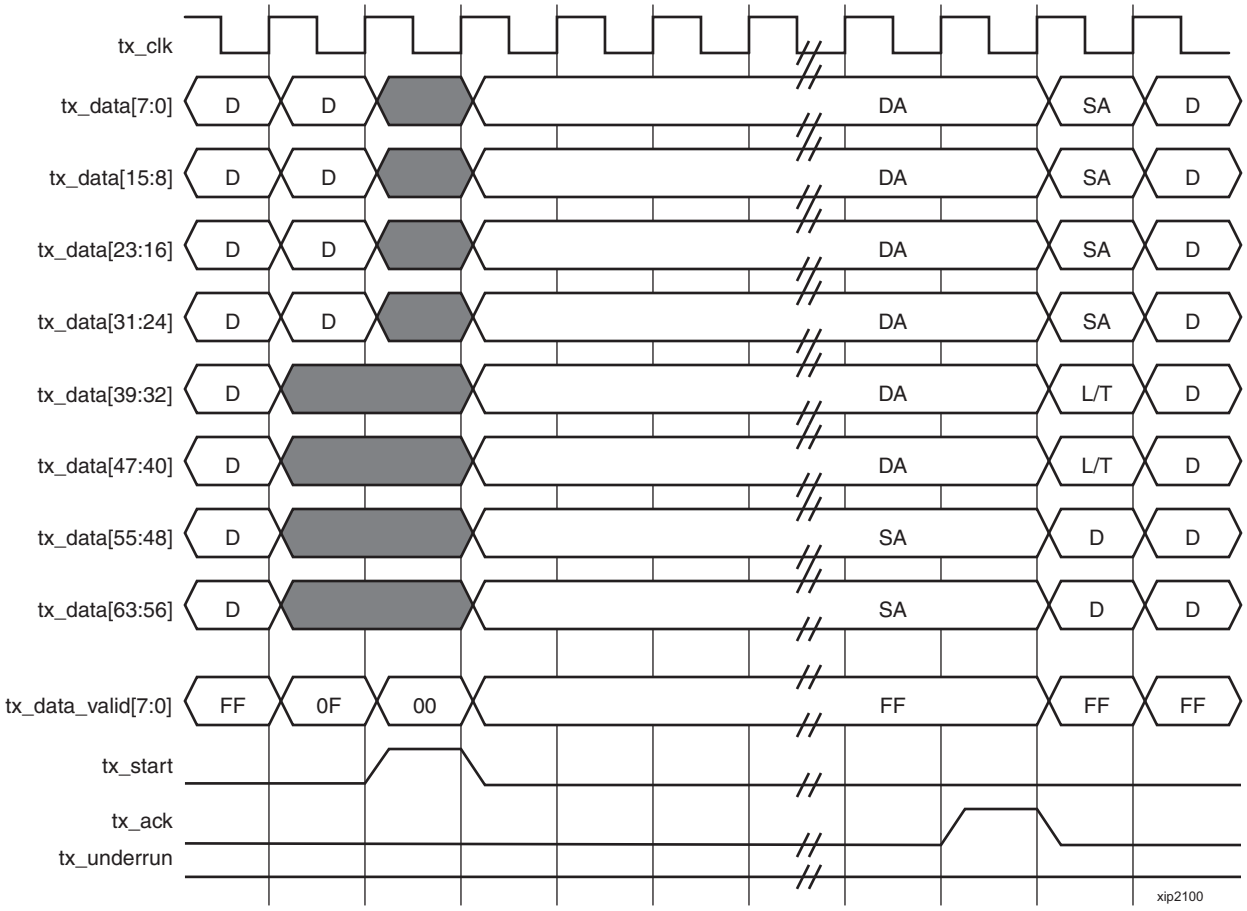


Figure 6-5: **Back-to-Back Frame Transmission with Back Pressure from MAC**

Transmission of Custom Preamble

You can elect to use a custom preamble field. If this function is selected (via 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_data[63:8]` when `tx_start` is asserted. Figure 6-6 shows a frame presented at the Transmit Client Interface with custom Preamble where P1 to P7 denote the custom data bytes.

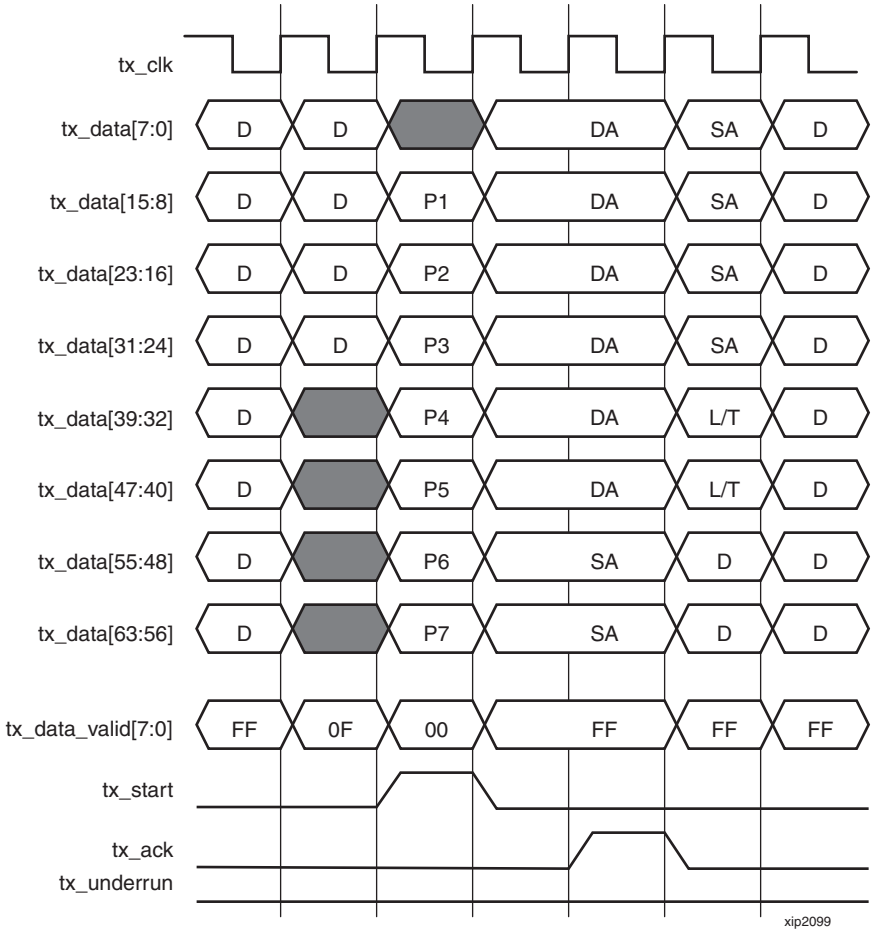


Figure 6-6: **Transmission of Frame with Custom Preamble**

The MAC core will substitute the IEEE standard preamble with that supplied by you. Figure 6-7 shows the transmission of a frame with custom preamble (P1 to P7) at the XGMII interface.

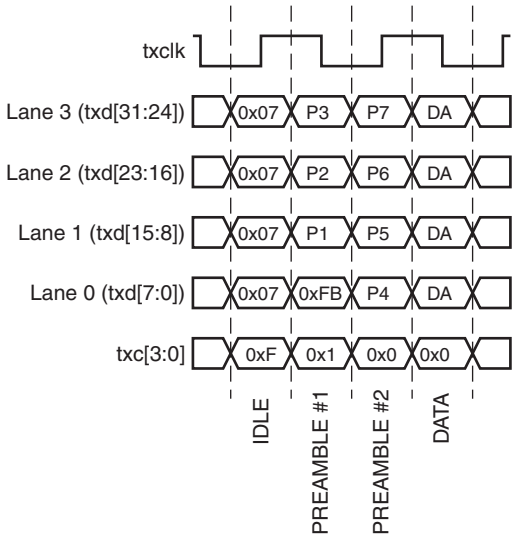


Figure 6-7: **XGMII Frame Transmission of Custom Preamble**

VLAN Tagged Frames

Transmission of a VLAN tagged frame (if enabled) is shown in Figure 6-8. 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 *IEEE 802.3-2008*.

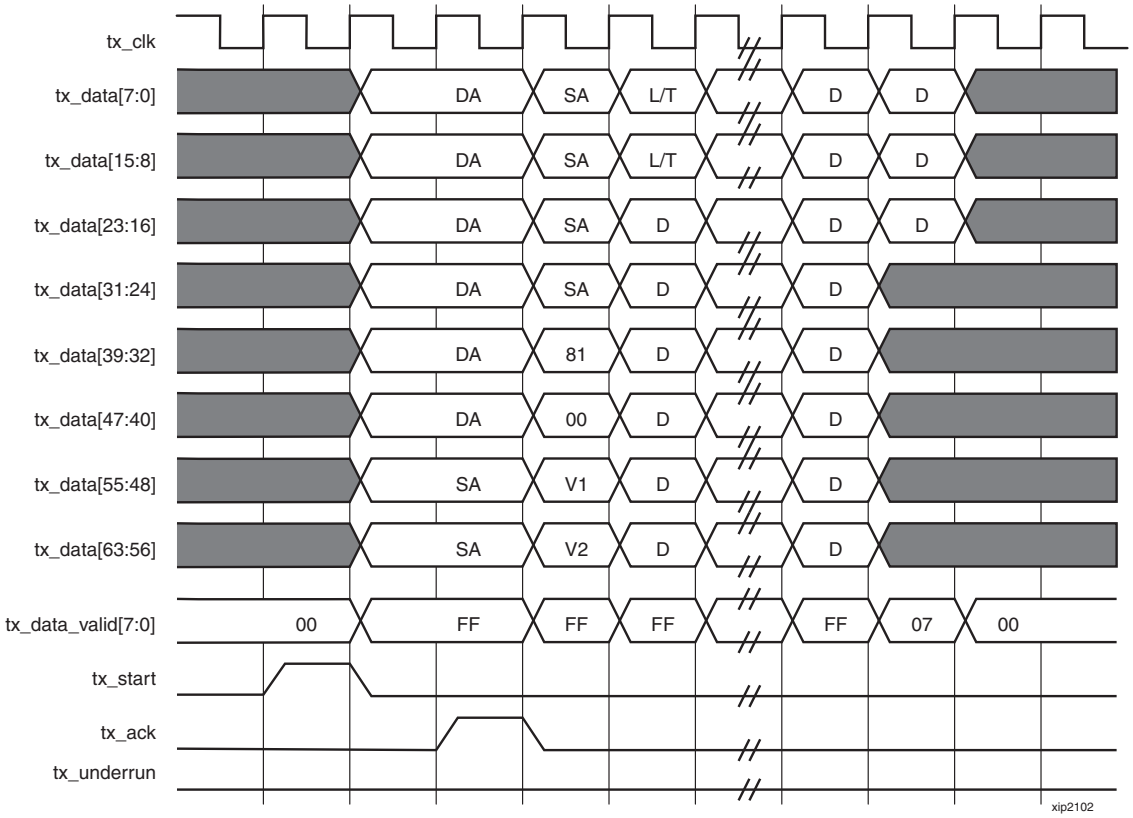


Figure 6-8: **Transmission of a VLAN Tagged Frame**

Maximum Permitted Frame Length

The maximum legal length of a frame specified in *IEEE 802.3-2008* is 1518 bytes for non-VLAN tagged frames. VLAN tagged frames may 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 will insert an error code to corrupt the current frame and the frame will be truncated to the maximum legal length. When jumbo frame handling is enabled, frames which are longer than the legal maximum are transmitted error free.

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

Interframe Gap Adjustment

You can elect to vary the length of the interframe gap. If this function is selected (via a configuration bit, see Configuration Registers in Chapter 7), the MAC will exert 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 is always maintained. Figure 6-9 shows the MAC operating in this mode.

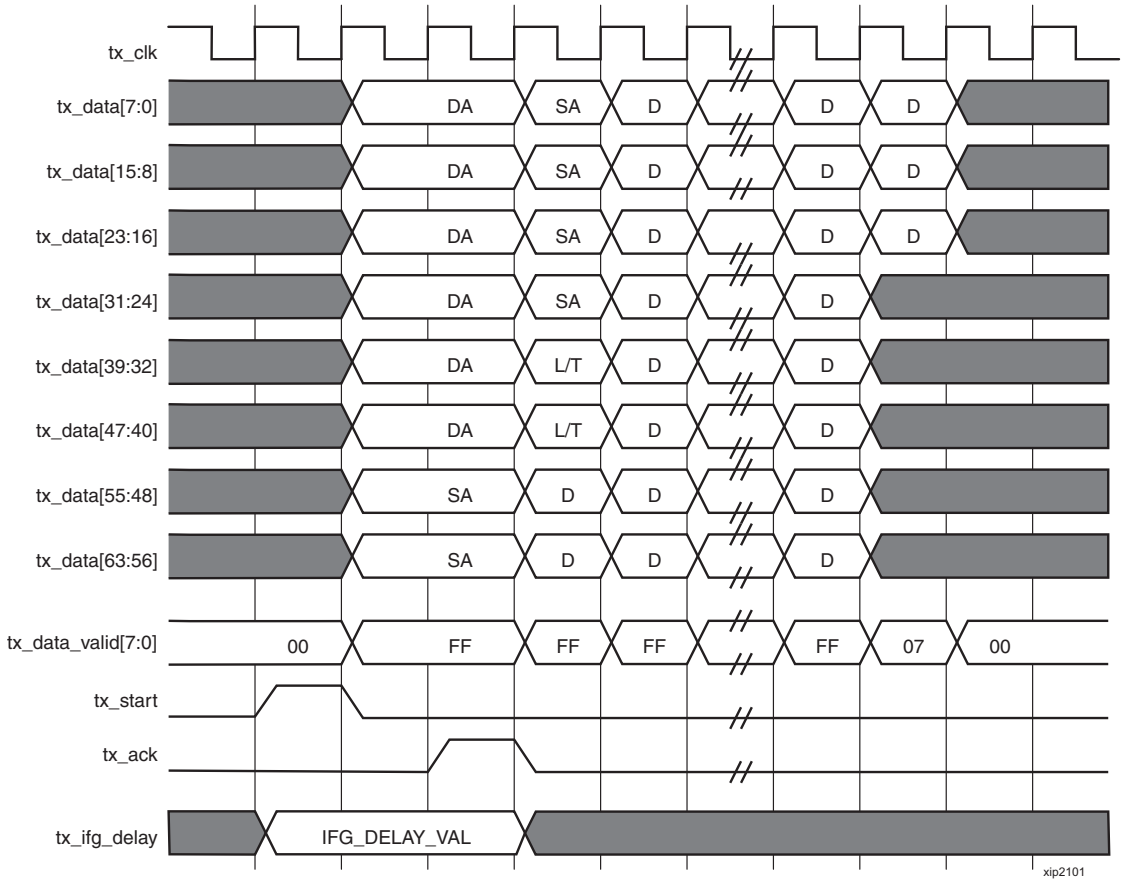


Figure 6-9: **Inter Frame Gap Adjustment**

Transmission of Frames During Local/Remote Fault Reception

When a local or remote fault has been received, the core may not transmit frames if Fault Inhibit has been disabled (via a configuration bit, see Configuration Registers in Chapter 7). When Fault Inhibit is disabled, the Reconciliation Sublayer transmits ordered sets as presented in *IEEE 802.3-2008*; 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 Client-Side Interface

The client-side interface on receive has a 64-bit data path with 8 control bits to delineate bytes within the 64-bit port. Additionally, there are signals to flag the validity of frames transferred out of the core. The signals are shown in Table 6-4.

Table 6-4: **Client-Side Interface Ports: Receive**

Name	Direction	Description
rx_data[63:0]	Output	Received data, eight bytes wide.
rx_data_valid[7:0]	Output	Received control bits, one bit per receive lane.
rx_good_frame	Output	Asserted at the end of frame to indicate the frame was successfully received and should be processed by the user logic.
rx_bad_frame	Output	Asserted at the end of frame to indicate the frame was not successfully received and should be discarded by the user logic.

For `rx_data` (Table 6-5), the port is logically divided into lane 0 to lane 7 with the corresponding bit of the `rx_data_valid` word signifying valid data on the `rx_data` port.

Table 6-5: **rx_data Lanes**

Lane	rx_data 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 Reception

The timing of a normal inbound frame transfer is represented in Figure 6-10. 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. Once frame reception begins, data is transferred on consecutive clock cycles to the receive client until the frame is complete. The MAC asserts the `rx_good_frame` signal to indicate that the frame was successfully received and that the frame should be analyzed by the client.

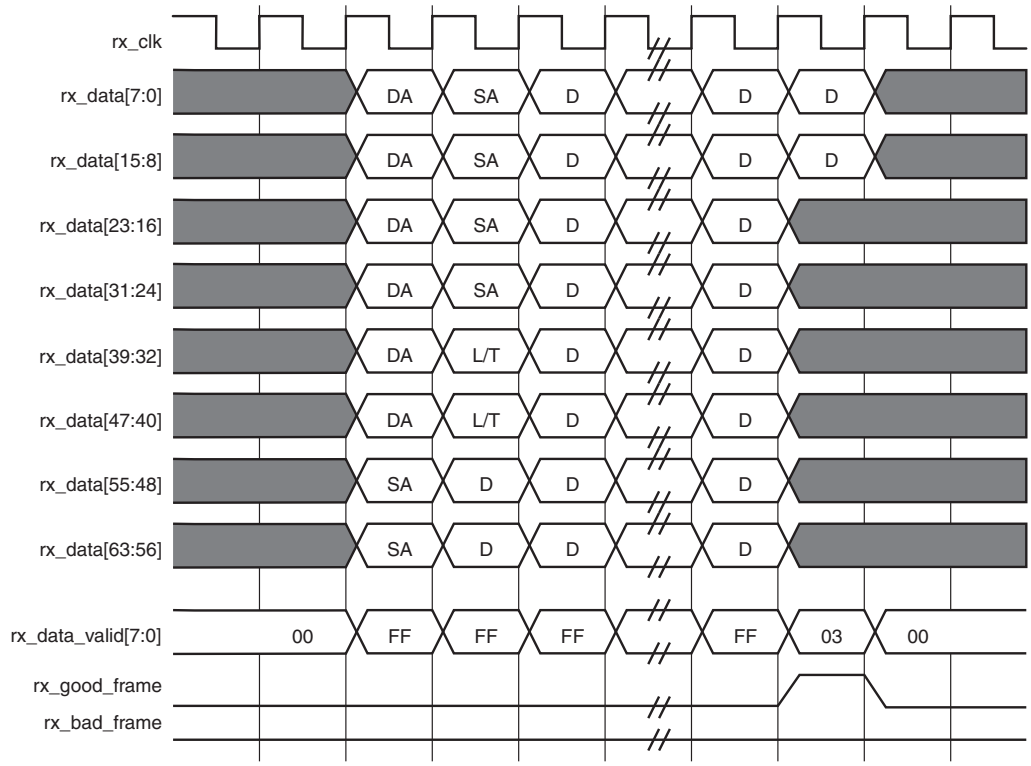


Figure 6-10: **Normal Frame Reception Across Client-Side Interface**

Frame parameters (destination address, source address, length/type and, optionally, FCS) are supplied on the data bus as shown on the timing diagram. The abbreviations are identical to those described in Table 6-3, page 42.

If the length/type field in the frame has the length interpretation, and this indicates that the inbound frame has been padded to meet the Ethernet minimum frame size specification, this pad will not be passed to the client in the data payload. The exception to this occurs when FCS passing is enabled. See Configuration Registers in Chapter 7.

There is always at least one clock cycle with `rx_data_valid` = 0x00 between frames; there is no valid data for this clock edge.

rx_good_frame, rx_bad_frame Timing

Although the timing diagram in Figure 6-10 shows the `rx_good_frame` signal asserted at the same time as the last valid data on `rx_data`, this is not always the case. The `rx_good_frame` and `rx_bad_frame` signals are only asserted 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 may result from padding at the end of the Ethernet frame. This is represented in Figure 6-11.

Neither `rx_good_frame` or `rx_bad_frame` will not be asserted until the frame checks are complete. If the Length/Type field in the frame is incorrect and the frame is longer than indicated, then `rx_bad_frame` may be asserted significantly more than seven clock cycles after the end of valid data.

Although `rx_good_frame` is illustrated, the same timing applies to `rx_bad_frame`. Either the `rx_good_frame` or `rx_bad_frame` signal will, however, always be asserted before the next frame data begins to appear on `rx_data`.

Note: `rx_statistics_valid` will appear exactly one clock cycle before `rx_good_frame` and `rx_bad_frame`.

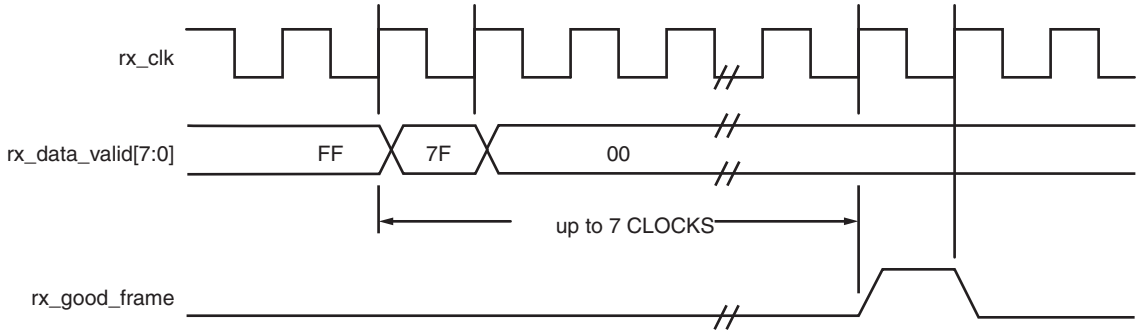


Figure 6-11: **Late Arrival of rx_good_frame**

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 6-12. In this case, the `rx_bad_frame` signal is asserted 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_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.
- 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.
- The XGMII data stream contains error codes.
- A valid pause frame, addressed to the MAC, is received when flow control is enabled.

See Overview of Flow Control, page 85 for more information.

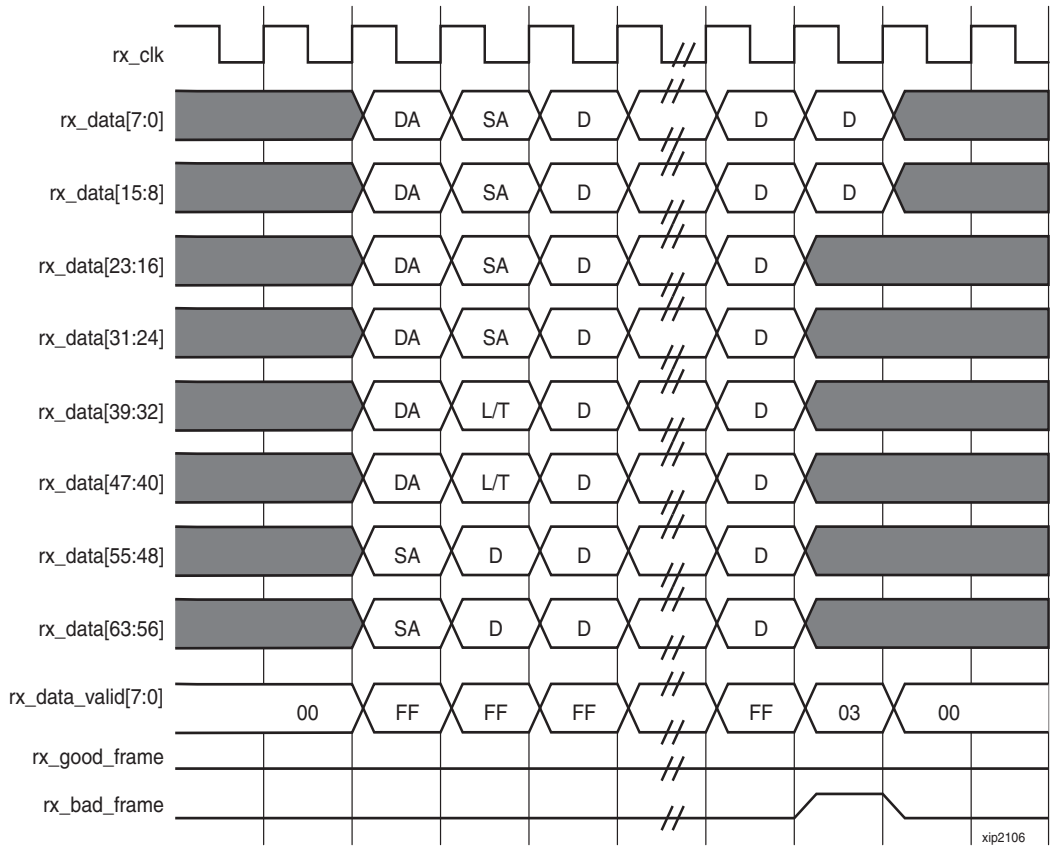


Figure 6-12: **Frame Reception with Error**

Reception with In-Band FCS Passing

Figure 6-13 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 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_bad_frame` is asserted if the FCS check fails.

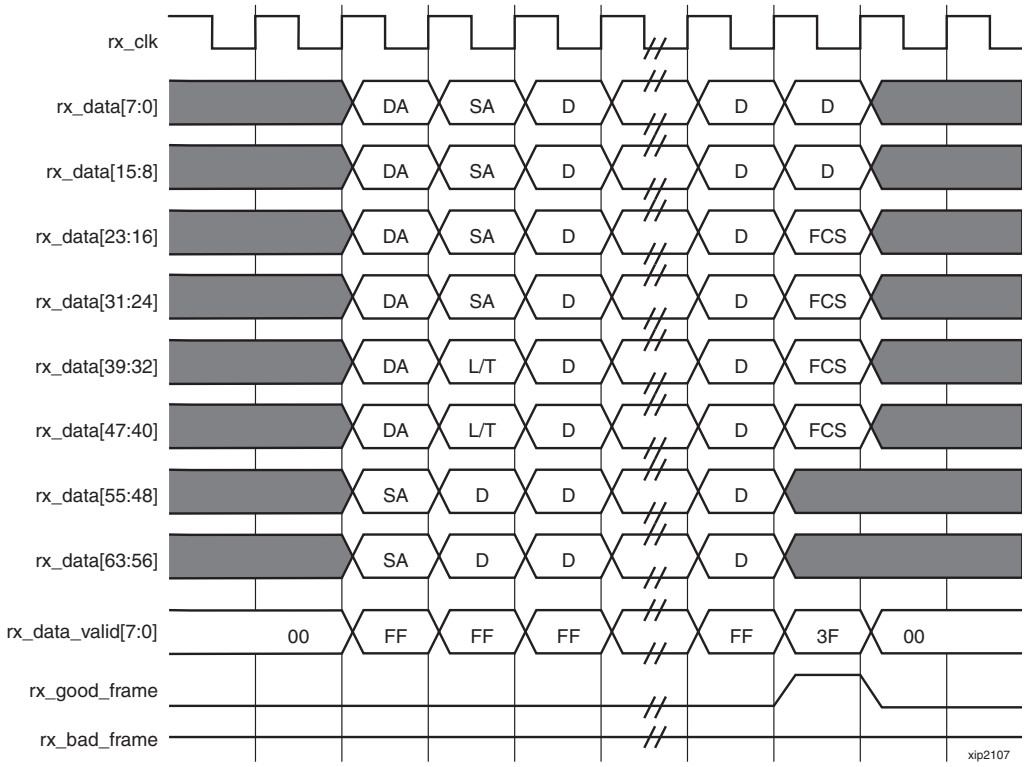


Figure 6-13: **Frame Reception with In-Band FCS Passing**

Reception of Custom Preamble

You can elect to use a custom preamble field. If this function is selected (via a configuration bit, see Configuration Registers in Chapter 7), the preamble field can be recovered from the received data and presented on the Client Interface. If this mode is enabled, the custom preamble data will presented on `rx_data[63:8]`. The `rx_data_valid` output will be asserted to frame the custom preamble. Figure 6-14 shows the reception of a frame with custom preamble.

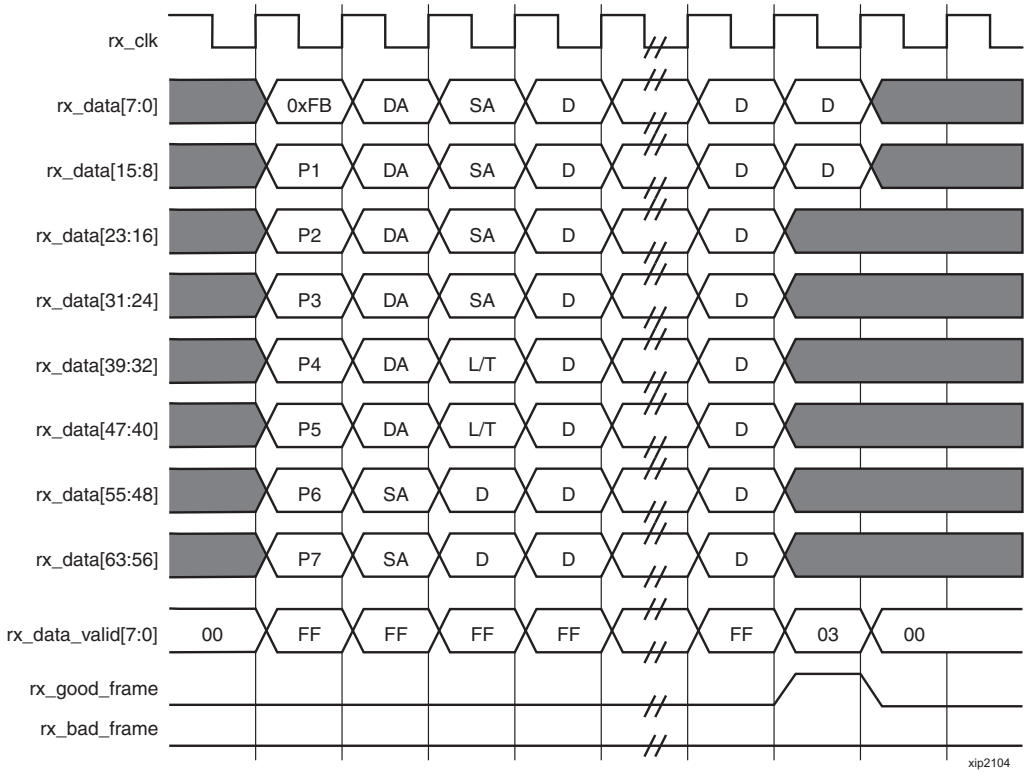


Figure 6-14: **Frame Reception with Custom Preamble**

In addition to the custom preamble, the end of frame condition may also be slightly altered depending on frame sizes and IFG. In normal operation the `rx_data_valid` output is guaranteed to go to 0x00 between frames. With the preamble preserve mode enabled, this is no longer guaranteed, and the end of frame condition is now `rx_data_valid` not equal to 0xFF. Figure 6-15 shows reception of 2 frames with `rx_data_valid` not equal to 0xFF for a single byte (`rx_data[63:56]`).

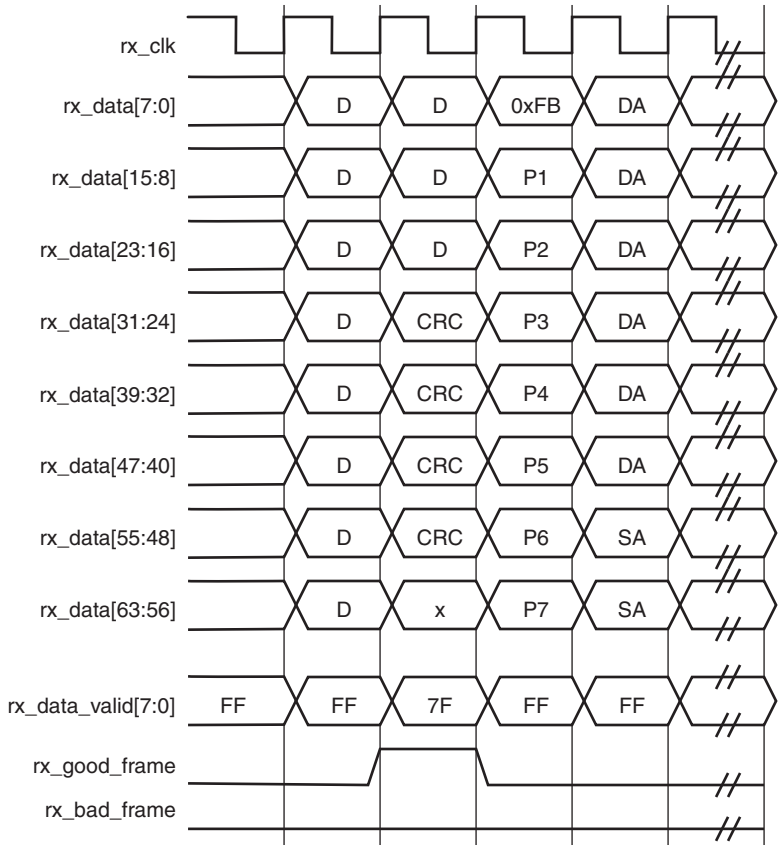


Figure 6-15: **Non-Standard End of Frame Due to Custom Preamble Mode**

VLAN Tagged Frames

The reception of a VLAN tagged frame (if enabled) is represented in Figure 6-16. 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 *IEEE 802.3-2008*.

All VLAN tagged frames are treated as Type frames, that is, any padding is treated as valid and passed to the client.

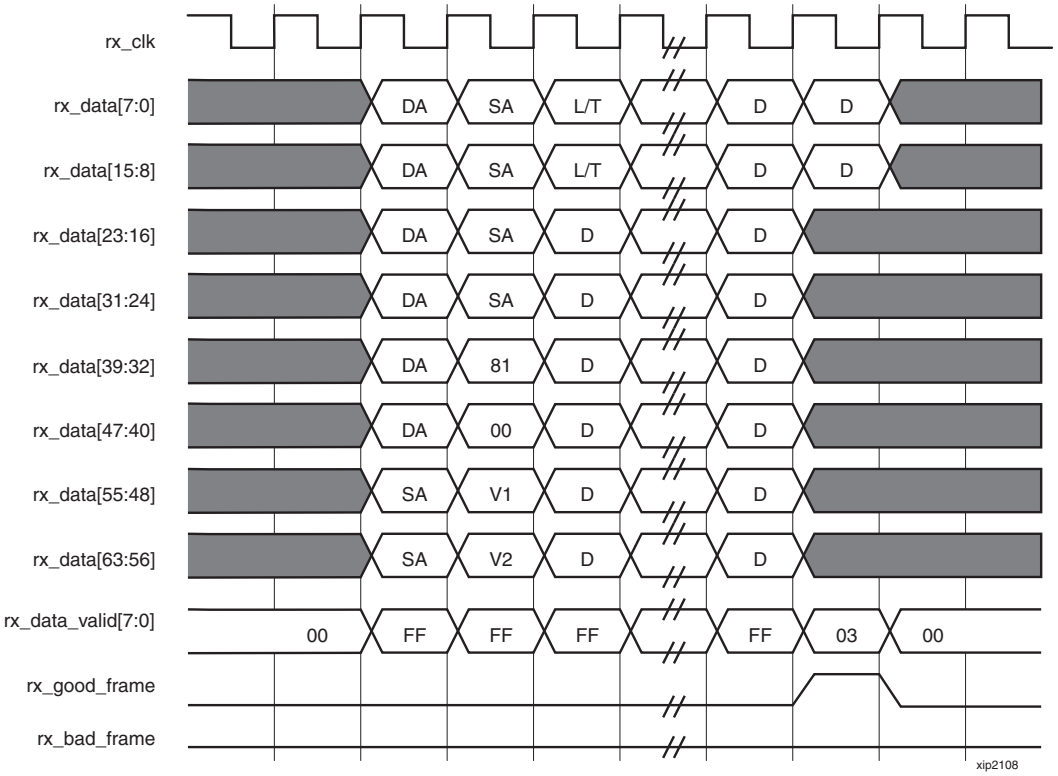


Figure 6-16: **Reception of a VLAN Tagged Frame**

Maximum Permitted Frame Length

The maximum legal length of a frame specified in *IEEE 802.3-2008* is 1518 bytes for non-VLAN tagged frames. VLAN tagged frames may be extended to 1522 bytes. When jumbo frame handling is disabled and the core receives a frame which exceeds the maximum legal length, `rx_bad_frame` will be asserted. When jumbo frame handling is enabled, frames which are longer than the legal maximum are received in the same way as shorter frames.

For more information on enabling and disabling jumbo 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 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 (a length interpretation), is checked against the actual data length received.
- A value in the length/type field that is less than decimal 46, (a type 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 will be used to deassert `rx_data_valid` after the indicated number of data bytes so that the padding bytes are removed from the frame. See Reception 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 will mark 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 will not be used to deassert `rx_data_valid`. Instead, `rx_data_valid` 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. See Overview of Flow Control, page 85 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 6-17.

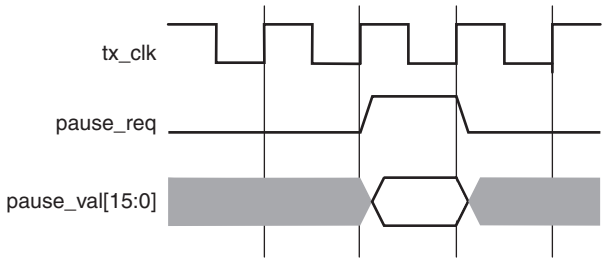


Figure 6-17: **Transmitting a Pause Frame**

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 will not disrupt any frame transmission in progress but will take priority over any pending frame transmission. This frame will be 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, the following 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 will be 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 will be ignored by the flow control logic and passed to the client with `rx_bad_frame` asserted.

The PHY-Side Interface

External XGMII vs. 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 note 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 may appear on the PHY side interface in lane 0 or lane 4. Likewise, the /S/ codeword may legally arrive at the RX PHY interface in either lane 0 or lane 4, but the MAC will always present it to the client logic with start of frame in lane 0.

XILINX®

Chapter 7

Interfacing to the Core: Control Interfaces, Statistics and MDIO

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 will be 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 a processor-independent interface with standard address, data, and control signals. It can be used as-is, or can simply be adapted to interface to common bus architectures.

This interface is used for:

- Configuring the MAC core
- 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
host_clk	Input	Clock for Management Interface. Range between 10 MHz and 133 MHz.
host_opcode[1:0]	Input	Defines operation to be performed over Management Interface.
host_addr[9:0]	Input	Address of register to be accessed.
host_wr_data[31:0]	Input	Data to write to register.
host_rd_data[31:0]	Output	Data read from register.
host_miim_sel	Input	When asserted, the MDIO interface is accessed.

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

Name	Direction	Description
host_req	Input	Used to request a transaction on the MDIO interface or read from the statistic registers.
host_miim_rdy	Output	When asserted, the MDIO interface has completed any pending transaction and is ready for a new transaction.

The Management Interface is accessed differently depending on the type of transaction; Table 7-2 is a truth table showing which access method is required for each transaction type. These access methods are described in the following sections.

Table 7-2: **Management Interface Transaction Types**

Transaction	HOST_MIIM_SEL	HOST_ADDR[9]
Configuration	0	1
Statistics	0	0
MDIO Access	1	X

Configuration Registers

Once 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 will 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
0x200	Receiver Configuration Word 0
0x240	Receiver Configuration Word 1
0x280	Transmitter Configuration
0x2C0	Flow Control Configuration
0x300	Reconciliation Sublayer Configuration
0x340	Management Configuration

The contents of each configuration register are shown in Tables 7-4 through 7-9.

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 data paths 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 will 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 will not perform the length/type field error checks as described in Length/Type Field Error Checks, page 60. When this bit is set to '0,' the length/type field checks will be performed; this is normal operation.
26	0	**Receiver Preserve Preamble Enable**. When this bit is set to '1,' the MAC receiver will preserve the preamble field of the received frame. When it is '0,' the preamble field is discarded as specified in *IEEE 802.3-2008*.
27	0	**VLAN Enable.** When this bit is set to '1,' VLAN tagged frames will be accepted by the receiver.
28	1	**Receiver Enable**. If set to '1,' the receiver block will be operational. If set to '0,' the block will ignore activity on the physical interface RX port.
29	0	**In-band FCS Enable**. When this bit is '1,' the MAC receiver will pass the FCS field up to the client as described in Reception with In-Band FCS Passing, page 56. When it is '0,' the client will not be passed the FCS. In both cases, the FCS will be verified on the frame.

Table 7-5: **Receiver Configuration Word 1 *(Cont'd)***

Bit	Default Value	Description
30	0	**Jumbo Frame Enable**. When this bit is set to '1,' the MAC receiver will accept frames that are greater than the maximum legal frame length specified in *IEEE 802.3-2008*. When this bit is '0,' the MAC will only accept frames up to the legal maximum.
31	0	**Receiver reset**. When this bit is set to '1,' the receiver will be reset. The bit will then automatically revert to '0.' This reset will also set 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 will preserve the custom preamble field presented on the Client Interface. When it is '0,' the standard preamble field specified in *IEEE 802.3-2008* will be transmitted.
24	0	**Deficit Idle Count Enable**. When this bit is set to '1,' the core will reduce 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 LAN Mode and In-band FCS are enabled or if Interframe Gap Adjust is enabled.
25	0	**Interframe Gap Adjust Enable**. When this bit is set to '1,' the core will read 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, page 50. When this bit is set to '0,' the transmitter will output 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 will automatically insert 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 will use normal Ethernet interframe gaps (LAN mode).
27	0	**VLAN Enable.** When this bit is set to '1,' the transmitter will allow 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.

Table 7-6: **Transmitter Configuration Word *(Cont'd)***

Bit	Default Value	Description
29	0	**In-band FCS Enable.** When this bit is '1,' the MAC transmitter will expect the FCS field to be passed in by the client as described in Transmission with In-Band FCS Passing, page 43. When this bit is '0,' the MAC transmitter will append padding as required, compute the value for the FCS field and append it to the frame.
30	0	**Jumbo Frame Enable**. When this bit is set to '1,' the MAC transmitter will send frames that are greater than the maximum legal frame length specified in *IEEE 802.3-2008*. When this bit is '0,' the MAC will only send frames up to the legal maximum.
31	0	**Transmitter Reset**. When this bit is set to '1,' the transmitter will be reset. The bit will then automatically revert to '0.' This reset will also set 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 will inhibit the transmitter operation as described in Receiving a Pause Frame, page 61. When this bit is '0,' received flow control frames will always be passed up to the client.
30	1	**Flow Control Enable (TX).** When this bit is '1,' asserting the PAUSE_REQ signal will send 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 will transmit ordered sets as laid out in *IEEE 802.3-2008*; that is, when the RS is receiving Local Fault ordered sets, it will transmit Remote Fault ordered sets. When it is receiving Remote Fault ordered sets, it will transmit idles code words. When this bit is set to '1,' the reconciliation sublayer will always transmit 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.

Table 7-8: **Reconciliation Sublayer Configuration Word *(Cont'd)***

Bit	Default Value	Description
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: **Management Configuration Word**

Bit	Default Value	Description
4:0	All 0s	**Clock Divide[4:0]**. Used as a divider value to generate MDC signal at 2.5 MHz. See MDIO Interface, page 74.
5	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 remain inactive.
31:6	N/A	**Reserved**.

Writing to the configuration registers through the Management Interface is depicted in Figure 7-1. When accessing the configuration registers (when `host_addr[9]` = 1 and `host_miim_sel` = 0), the upper bit of `host_opcode` functions as an active low write-enable signal. The lower `host_opcode` bit is a "don't care."

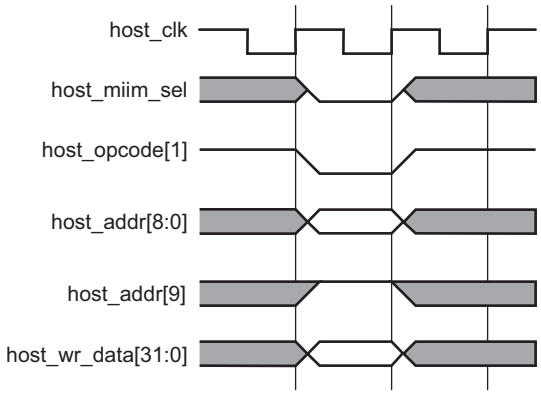


Figure 7-1: **Configuration Register Write Timing**

Reading from the configuration register words is similar, except that the upper host_opcode bit should be '1,' as shown in Figure 7-2. In this case, the contents of the register appear on `host_rd_data` and the `host_clk` edge after the register address is asserted onto `host_addr`.

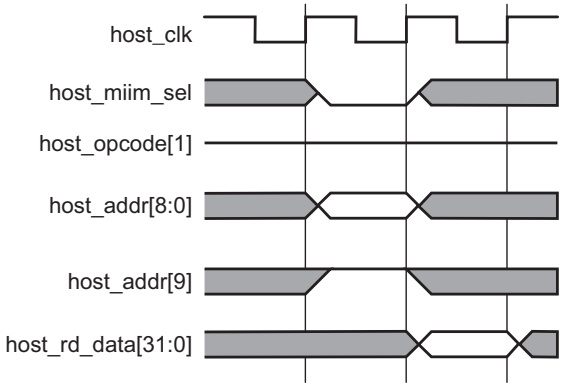


Figure 7-2: **Configuration Register Read Timing**

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 7-10.

As per the IEEE 802.3 specification 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 will only return to zero when they naturally wrap around or when the device is reconfigured.

Table 7-10: **Statistic Counters**

Address (hex)	Name	Description
0x000	Frames Received OK	A count of error free frames received.
0x001	Frame Check Sequence errors	A count of received frames that failed the CRC check and were at least 64 bytes in length.
0x002	Broadcast frames Received OK	A count of frames that were successfully received and were directed to the broadcast group address.
0x003	Multicast Frames Received OK	A count of frames that were successfully received and were directed to a non-broadcast group address.
0x004	64 byte Frames Received OK	A count of error-free frames received that were 64 bytes in length.
0x005	65-127 byte Frames Received OK	A count of error-free frames received that were between 65 and 127 bytes in length inclusive.
0x006	128-255 byte Frames Received OK	A count of error-free frames received that were between 128 and 255 bytes in length inclusive.
0x007	256-511 byte Frames Received OK	A count of error-free frames received that were between 256 and 511 bytes in length inclusive.
0x008	512-1023 byte Frames Received OK	A count of error-free frames received that were between 512 and 1023 bytes in length inclusive.
0x009	1024-MaxFrameSize byte Frames Received OK	A count of error-free frames received that were between 1024 bytes and the maximum legal frame size as specified in *IEEE 802.3-2008*.
0x00A	Control Frames Received OK	A count of error-free frames received that contained the MAC Control type identifier in the length/type field.

Table 7-10: **Statistic Counters *(Cont'd)***

Address (hex)	Name	Description
0x00B	Length/Type Out of Range	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).
0x00C	VLAN Tagged Frames Received OK	A count of error-free frames received with VLAN tags. This counter will only increment when the receiver has VLAN operation enabled.
0x00D	Pause Frames Received OK	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.
0x00E	Control Frames Received with Unsupported Opcode	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.
0x00F	Oversize Frames Received OK	A count of otherwise error-free frames received that exceeded the maximum legal frame length specified in *IEEE 802.3-2008*.
0x010	Undersized Frames Received	A count of the number of frames that were less than 64 bytes in length but were otherwise well formed.
0x011	Fragment Frames Received	A count of the number of packets received that were less than 64 bytes in length and had a bad frame check sequence field.
0x012	Number of Bytes Received	A count of bytes of frames that are received (destination address to frame check sequence inclusive).
0x013	Number of Bytes Transmitted	A count of bytes of frames that are transmitted (destination address to frame check sequence inclusive).

Table 7-10: **Statistic Counters *(Cont'd)***

Address (hex)	Name	Description
0x020	Frames Transmitted	A count of error-free frames transmitted.
0x021	Broadcast Frames Transmitted	A count of error-free frames transmitted to the broadcast address.
0x022	Multicast Frames Transmitted	A count of error-free frames transmitted to group addresses other than the broadcast address.
0x023	Underrun Errors.	A count of frames that would otherwise be transmitted by the core but could not be completed due to the assertion of tx_underrun during the frame transmission. This will not count frames which are less than 64 bytes in length.
0x024	Control Frames Transmitted OK	A count of error-free frames transmitted that contained the MAC Control Frame type identifier 88-08 in the length/type field.
0x025	64 byte Frames Transmitted OK	A count of error-free frames transmitted that were 64 bytes in length.
0x026	65-127 byte Frames Transmitted OK	A count of error-free frames transmitted that were between 65 and 127 bytes in length.
0x027	128-255 byte Frames Transmitted OK	A count of error-free frames transmitted that were between 128 and 255 bytes in length.
0x028	256-511 byte Frames Transmitted OK	A count of error-free frames transmitted that were between 256 and 511 bytes in length.
0x029	512-1023 byte Frames Transmitted OK	A count of error-free frames transmitted that were between 512 and 1023 bytes in length.
0x02A	1024-MaxFrameSize byte Frames Transmitted OK	A count of error-free frames transmitted that were between 1024 bytes and the maximum legal frame length specified in *IEEE 802.3-2008*.
0x02B	VLAN Tagged Frames Transmitted OK	A count of error-free frames transmitted that contained a VLAN tag. This counter will only increment when the transmitter has VLAN operation enabled.

Table 7-10: **Statistic Counters *(Cont'd)***

Address (hex)	Name	Description
0x02C	Pause Frames Transmitted OK	A count of error-free pause frames generated and transmitted by the core in response to an assertion of pause_req.
0x02D	Oversize Frames Transmitted OK	A count of otherwise error-free frames transmitted that exceeded the maximum legal frame length specified in *IEEE 802.3-2008*.

Figure 7-3 shows a statistics register access across the Management Interface. Each register is 64 bits wide and therefore must be read in a two-cycle transfer.

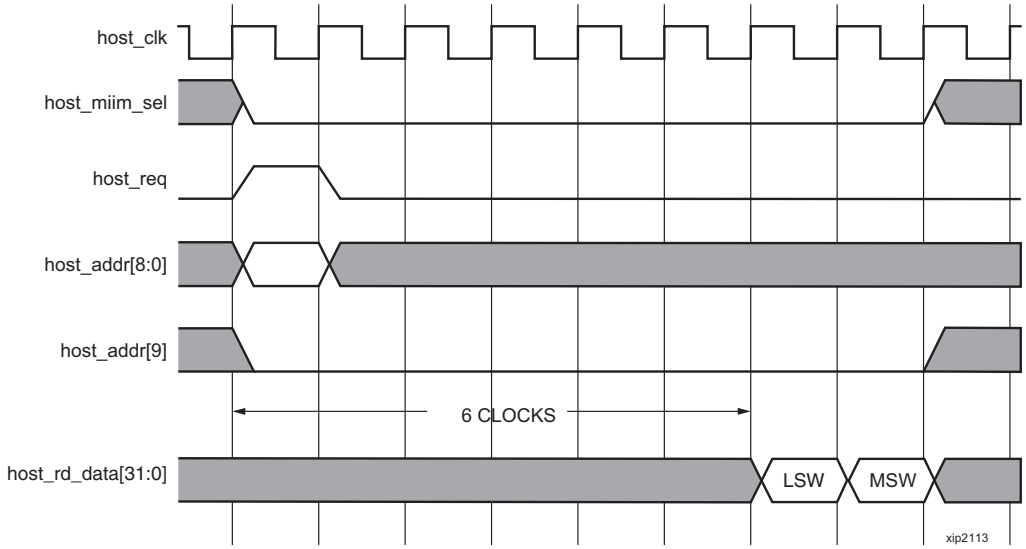


Figure 7-3: **Statistics Register Read Across Management Interface**

MDIO Interface

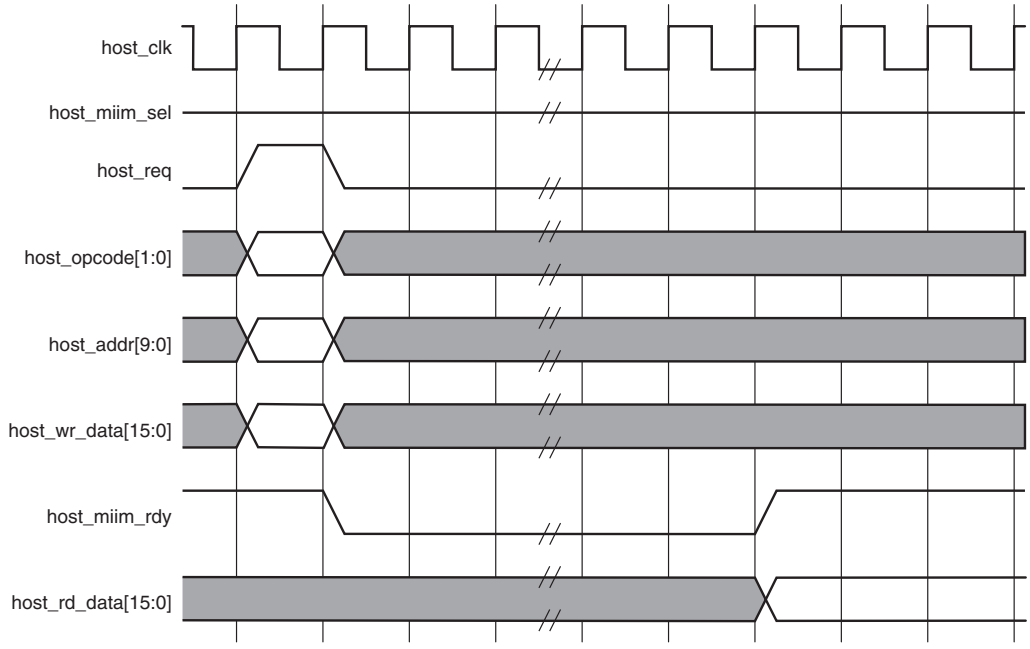
The Management Interface is also used to access the MDIO Interface of the MAC core; this interface is used to access the Managed Information Block (MIB) of the PHY components attached to the MAC core.

The MDIO Interface supplies a clock to the external devices, MDC. This clock is derived from the `host_clk` signal, using the value in the Clock Divide[4:0] configuration register. The frequency of MDC is given by the following equation:

$$f_{MDC} = \frac{f_{\text{HOST\_CLK}}}{(1 + \text{Clock Divide}[4:0]) \times 2}$$

The frequency of MDC given by this equation should not exceed 2.5 MHz to comply with the specification for this interface, *IEEE 802.3-2008*. To prevent MDC from being out of specification, the Clock Divide[4:0] value powers up at 00000, and while this value is in the register, it is impossible to enable the MDIO Interface.

Access to the MDIO Interface through the Management Interface is depicted in the timing diagram in Figure 7-4.



* If a read transaction is initiated, the host_rd_data bus is valid at the point indicated. If a write transaction is initiated, the host_wr_data bus must be valid at the indicated point. Simultaneous read and write is not permitted.

Figure 7-4: **MDIO Access Through the Management Interface**

For MDIO transactions, the following points apply:

- `host_miim_sel` is '1.'
- `host_opcode` maps to the OP (opcode) field of the MDIO frame.
- `host_addr` maps to the two address fields of the MDIO frame; PRTAD is `host_addr[9:5]`, and DEVAD is `host_addr[4:0]`.
- `host_wr_data[15:0]` maps into the address/data field of the MDIO frame when performing an address operation or a write operation.
- The address/data field of the MDIO frame maps into `host_rd_data[15:0]` when performing a read operation or a read/increment operation.

The MAC core signals to the host that it is ready for an MDIO transaction by asserting `host_miim_rdy`. A read or write transaction on the MDIO is initiated by a pulse on the host_req signal. This pulse is ignored if the MDIO interface already has a transaction in progress.

The MAC core then deasserts the `host_miim_rdy` signal while the transaction across the MDIO Interface is in progress. When the transaction across the MDIO Interface has been completed, the `host_miim_rdy` signal will be asserted by the MAC core; if the transaction is a read operation or a read/increment operation, the data will also be available on the `host_rd_data[15:0]` bus at this time.

The ports of the MDIO interface itself are shown in Table 7-11.

Table 7-11: **MDIO Interface Ports**

Name	Direction	Description
mdc	Output	MDIO Clock
mdio_in	Input	MDIO Input
mdio_out	Output	MDIO Output
mdio_tri	Output	MDIO Tristate. '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 7-5 illustrates the used of a SelectIO interface 3-state buffer as the bus interface.

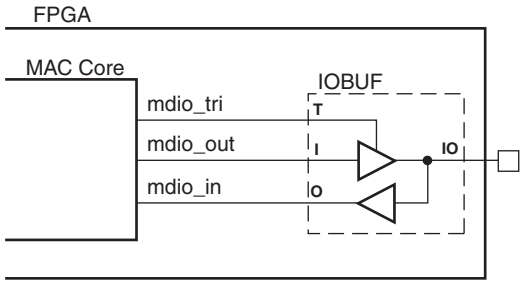


Figure 7-5: **Using a SelectIO Interface Tristate Buffer to Drive MDIO**

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

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

Set Address Transaction

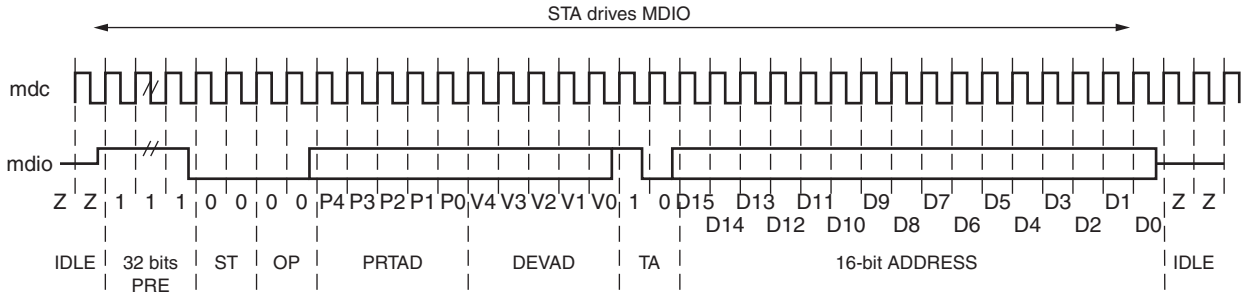


Figure 7-6: **MDIO Set Address Transaction**

Figure 7-6 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

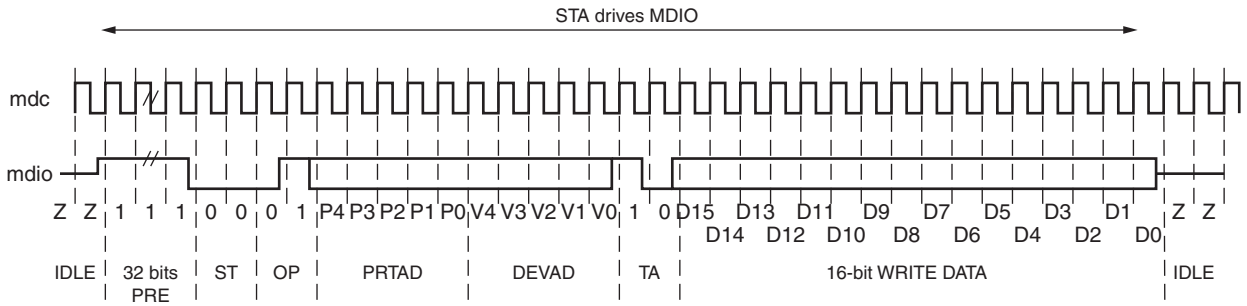


Figure 7-7: **MDIO Write Transaction**

Figure 7-7 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

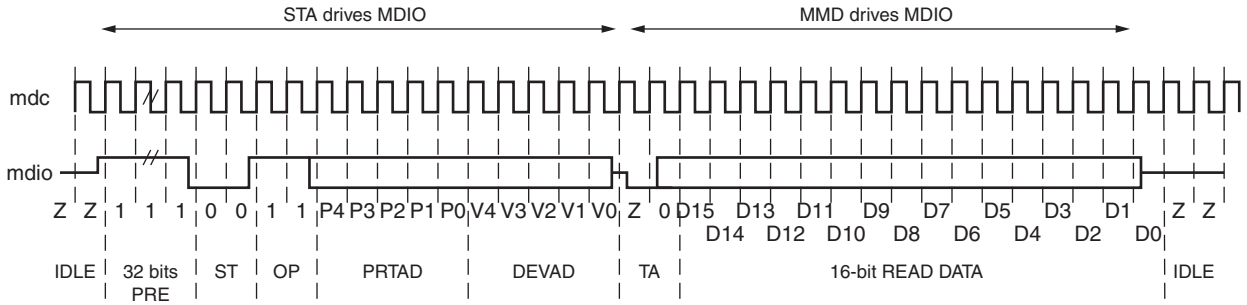


Figure 7-8: **MDIO Read Transaction**

Figure 7-8 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

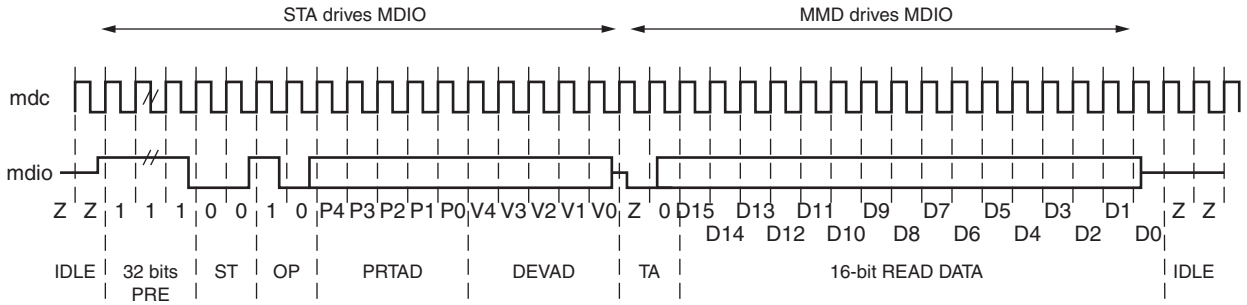


Figure 7-9: **MDIO Read-and-increment Transaction**

Figure 7-9 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 *IEEE 802.3-2008*.

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 7-12. 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 will 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 7-12.

Table 7-12: **configuration_vector Bit Definitions**

Bit(s)	Clock[1]	Description
47: 0	rx_clk0	**Pause frame MAC source address[47:0]**. This address shall be used by the MAC core to match against the Destination address of any incoming flow control frames, and as the source address for any outbound flow control frames. This address does not have any affect on frames passing through the main transmit and receive data paths 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 will be stored in byte [47:0] as 0xFFEEDDCCBBAA.
48	rx_clk0	**Receiver VLAN Enable**. When this bit is set to '1,' VLAN frames will be accepted by the receiver.
49	rx_clk0	**Receiver Enable**. When this bit is set to '1,' the receiver will be operational. When '0,' the receiver will ignore activity on the physical interface RX port.
50	rx_clk0	**Receiver In-band FCS Enable**. When this bit is '1,' the MAC receiver will pass the FCS field up to the client as described in Reception with In-Band FCS Passing, page 56. When it is '0,' the MAC receiver will not pass the FCS field. In both cases, the FCS field will be verified on the frame.
51	rx_clk0	**Receiver Jumbo Frame Enable**. When this bit is '0,' the receiver will not pass frames longer than the maximum legal frame size specified in *IEEE 802.3-2008*. When it is '1,' the receiver will not have an upper limit on frame size.
52	N/A	**Receiver Reset.** When this bit is '1,' the receiver is held in reset. This signal is an input to the reset circuit for the receiver block. See Reset Circuits, page 97 for more information.
53	tx_clk0	**Transmitter LAN/WAN Mode.** When this bit is '1,' the transmitter will automatically insert 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 will use standard Ethernet interframe gaps (LAN mode).

Table 7-12: **configuration_vector Bit Definitions *(Cont'd)***

Bit(s)	Clock[1]	Description
54	tx_clk0	**Transmitter Interframe Gap Adjust Enable**. When this bit is '1,' the transmitter will read the value of the tx_ifg_delay port and set the interframe gap accordingly. If it is set to '0,' the transmitter will insert a minimum interframe gap. This bit is ignored if bit 53 (Transmitter LAN/WAN Mode) is set to '1.'
55	tx_clk0	**Transmitter VLAN Enable**. When this bit is set to '1,' the transmitter will allow the transmission of VLAN tagged frames.
56	tx_clk0	**Transmitter Enable**. When this bit is set to '1,' the transmitter will be operational. When set to '0,' the transmitter will be disabled.
57	tx_clk0	**Transmitter In-Band FCS Enable**. When this bit is '1,' the MAC transmitter will expect the FCS field to be pass in by the client as described in Transmission with In-Band FCS Passing, page 43. When it is '0,' the MAC transmitter will append padding as required, compute the FCS and append it to the frame.
58	tx_clk0	**Transmitter Jumbo Frame Enable**. When this bit is '1,' the MAC transmitter will allow frames larger than the maximum legal frame length specified in *IEEE 802.3-2008* to be sent. When set to '0,' the MAC transmitter will only allow frames up to the legal maximum to be sent.
59	N/A	**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, page 97 special for more information.
60	rx_clk0	**Receive Flow Control Enable**. When this bit is '1,' received flow control frames will inhibit the transmitter operation as described in Receiving a Pause Frame, page 61. When it is '0,' received flow frames are passed up to the client.
61	tx_clk0	**Transmit Flow Control Enable**. When this bit is '1,' asserting the pause_req signal shall cause the MAC core to send a flow control frame out from the transmitter as described in Transmitting a Pause Frame, page 61. When this bit is '0,' asserting the pause_req signal will have no effect.
62	tx_clk0	**Deficit Idle Count Enable**. When this bit is set to '1,' the core will reduce the IFG as described in *IEEE 802.3-2008* 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 LAN Mode and In-band FCS are both enabled or if Interframe Gap Adjust is enabled.
63	-	**Reserved.** Tie to '0.'

Table 7-12: **configuration_vector Bit Definitions *(Cont'd)***

Bit(s)	Clock[1]	Description
64	tx_clk0	**Reconciliation Sublayer Fault Inhibit**. When this bit is '0,' the reconciliation sublayer will transmit ordered sets as laid out in *IEEE 802.3-2008*; that is, when the RS is receiving local fault ordered sets, it will transmit Remote Fault ordered sets. When it is receiving Remote Fault ordered sets, it will transmit idle code words. When this bit is '1,' the Reconciliation Sublayer will always transmit the data presented to it by the MAC, regardless of whether fault ordered sets are being received.
65	tx_clk0	**Transmitter Preserve Preamble Enable**. When this bit is set to '1,' the MAC transmitter will preserve the custom preamble field presented on the Client Interface. When it is '0,' the standard preamble field specified in *IEEE 802.3-2008* will be transmitted.
66	rx_clk0	**Receiver Preserve Preamble Enable**. When this bit is set to '1,' the MAC receiver will preserve the preamble field on the received frame. When it is '0,' the preamble field is discarded as specified in *IEEE 802.3-2008*.
67	rx_clk0	**Receiver Length/Type Error Disable**. When this bit is set to '1,' the core will not perform the length/type field error check as described in Length/Type Field Error Checks, page 60. When this bit is '0,' the length/type field checks will be performed; this is normal operation.
68	rx_clk0	**Control Frame Length Check Disable**. When this bit is set to '1,' the core will not mark control frames as 'bad' if they are greater than the minimum frame length.

1. The Clock heading denotes which clock domain the configuration signal is registered into before use by the core. It is recommended that the input signals be driven synchronously from this domain.

Table 7-13: **status_vector Bit Definitions**

Bit(s)	Clock	Description
0	rx_clk0	**Local Fault Received**. If this bit is '1,' the RS layer is receiving local fault sequence ordered sets. Read-only.
1	rx_clk0	**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 7-14.

All bit fields, with the exception of byte_valid, are valid only when the tx_statistics_valid is asserted. This is illustrated in Figure 7-10. byte_valid is significant on every tx_clk0 cycle.

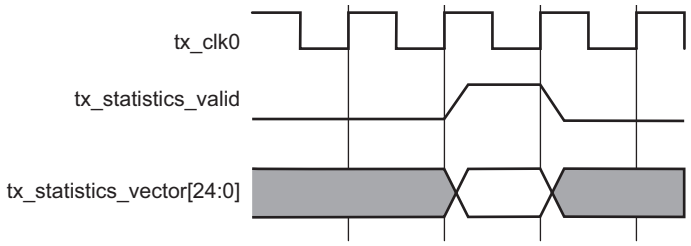


Figure 7-10: **Transmitter Statistics Output Timing**

Table 7-14: **Transmit Statistics Vector Bit Description**

Bit	Name	Description
24	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.
23 to 20	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.
19	vlan_frame	Asserted if the previous frame contained a VLAN identifier in the length/type field and transmitter VLAN operation is enabled.
18 to 5	frame_length_count	The length of the previously transmitted frame in bytes. The count will stick at 16383 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 7-15.

All bit fields, with the exception of `bytes_valid`, are valid only when `rx_statistics_valid` is asserted. This is illustrated in Figure 7-11. `bytes_valid` is significant on every `rx_clk0` cycle.

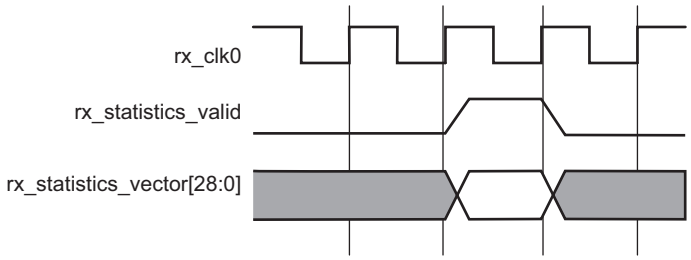


Figure 7-11: **Receiver Statistics Output Timing**

Table 7-15: **Receive Statistics Vector Description**

Bits	Name	Description
28	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).
27	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).
26	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.
25 to 22	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.
21	vlan_frame	Asserted if the previous frame contained a VLAN tag in the length/type field and VLAN operation was enabled in the receiver.

Table 7-15: **Receive Statistics Vector Description *(Cont'd)***

Bits	Name	Description
20	out_of_bounds	Asserted if the previous frame exceeded the maximum legal frame length specified in *IEEE 802.3-2008*. This is only asserted if jumbo frames are disabled.
19	control_frame	Asserted if the previous frame contained the MAC Control Frame identifier 88-08 in the length/type field.
18 to 5	frame_length_count	The length in bytes of the previous received frame. The count will stick at 16383 for any Jumbo frames larger than this value. If jumbo frames are disabled, the count will stick 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 may 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, page 64.

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 will result 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 problem.

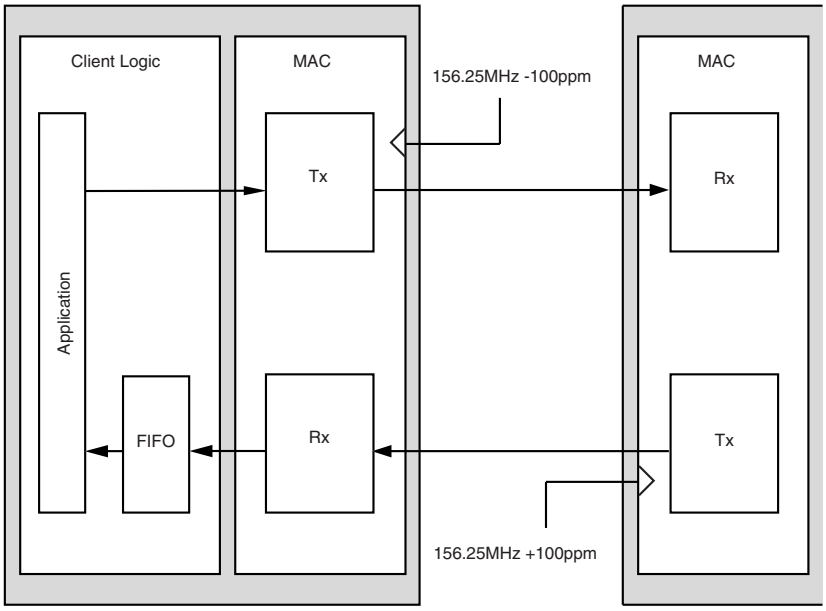


Figure 8-1: **The Requirement for Flow Control**

Flow Control Basics

A MAC may 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 may 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 may 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.

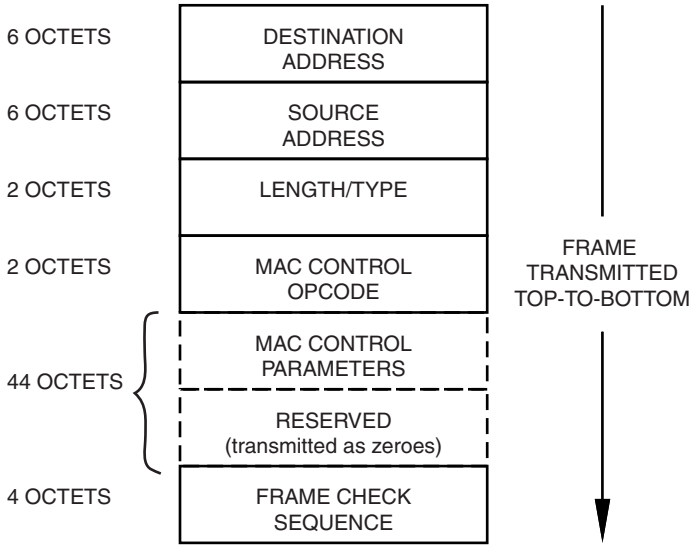


Figure 8-2: **MAC Control Frame Format**

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 may initiate a pause control frame by asserting the `pause_req` signal. Figure 8-3 displays this timing.

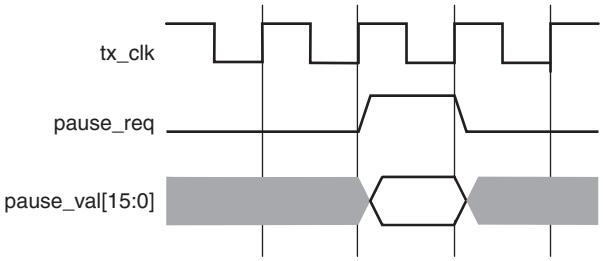


Figure 8-3: **Pause Request Timing**

This action causes the core to construct and transmit a pause control frame on the link with the following 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 will respond to).
- The source address used is the configurable Pause Frame MAC Address (see Configuration Registers, page 64).
- The value sampled from the `pause_val[15:0]` port at the time of the `pause_req` assertion will be 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 will then follow in preference to any pending client supplied frame.

A pause control frame initiated by this method will be 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 will be transmitted. The `pause_val[15:0]` value used will be the most recent value sampled.

Client-Initiated Pause Request

For maximum flexibility, flow control logic can be disabled in the core (see Configuration Registers, page 64) and alternatively implemented in the client logic connected to the core. Any type of control frame can be transmitted through the core via the client interface using the same transmission procedure as a standard Ethernet frame (see Normal Frame Transmission, page 41).

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, page 64), 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 Receiving a Pause Frame, page 61).

Pause Frame Reception Enabled

When pause control reception is enabled (see Configuration Registers, page 64) and an error-free frame is received by the MAC core, the following 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, page 64).
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, page 82).

Client-Initiated Response to a Pause Request

For maximum flexibility, flow control logic can be disabled in the core (see Configuration Registers, page 64) and alternatively implemented in the client logic connected to the core. Any type of error free control frame will then be 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 will fill and overflow. The aim is to implement a Flow Control method which will, over a long time period, reduce 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 will cause the right-hand MAC to cease transmission and the FIFO of the left-hand MAC will start 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 will immediately resume 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.

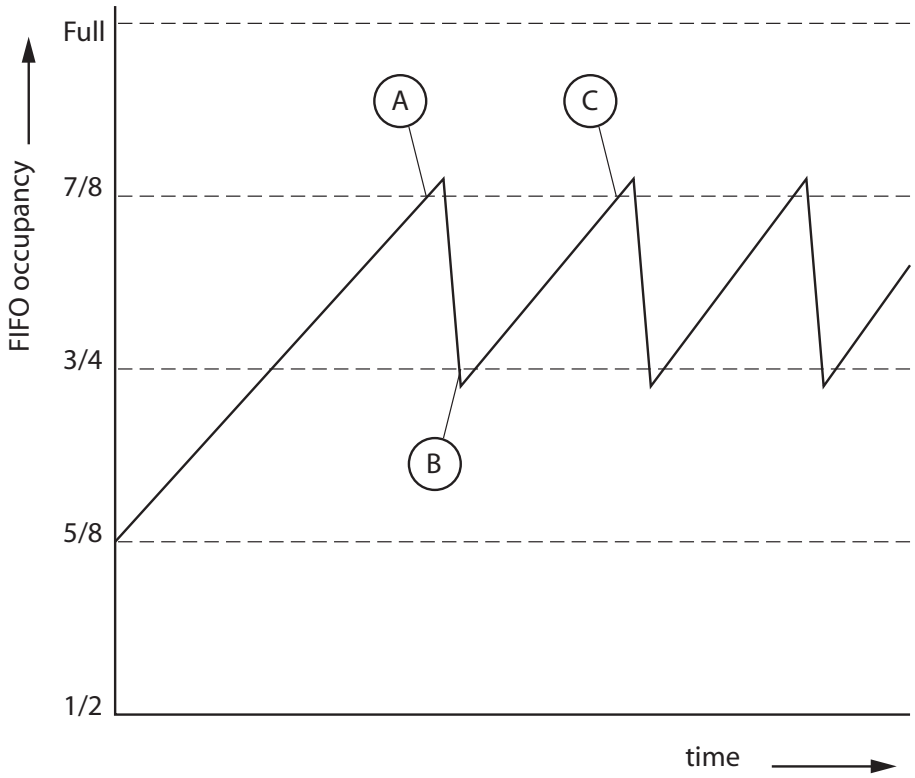


Figure 8-4: **Flow Control Implementation Triggered from FIFO Occupancy**

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 the *10-Gigabit Ethernet MAC Getting Started Guide* for a complete description of the CORE Generator™ software output files.

Not all constraints will be 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 Speedgrade Selection

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

```
# set the part and package
CONFIG PART = xc5vlx30tff665-1;
```

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

- Virtex®-4 family devices, speedgrade of -10 or faster
- Virtex-5 family devices, speed grade of -1 or faster
- 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 = "xgmactxgrp";
NET "tx_clk0" TNM_NET = "xgmactxgrp";
TIMESPEC "TS_xgmactx" = PERIOD "xgmactxgrp" 6400 ps HIGH 50 %;

# Set up DCM for transmit side
INST "tx_dcm" DLL_FREQUENCY_MODE    = HIGH;
INST "tx_dcm" DUTY_CYCLE_CORRECTION = TRUE;
INST "tx_dcm" CLK_FEEDBACK          = 1X;
```

This section sets the period of the transmit clock and sets the attributes of the DCM used in the example design.

```
# Main receive clock/period constraints
NET "xgmii_rx_clk" TNM_NET = "xgmacrxgrp";
NET "*rx_clk0*"      TNM_NET = "xgmacrxgrp";
TIMESPEC "TS_xgmacrx"    = PERIOD "xgmacrxgrp" 6400 ps HIGH 50 %;

# Set up DCM for receive side
INST "*rx_dcm" DLL_FREQUENCY_MODE    = HIGH;
INST "*rx_dcm" DUTY_CYCLE_CORRECTION = TRUE;
INST "*rx_dcm" CLK_FEEDBACK          = 1X;
INST "*rx_dcm" CLKOUT_PHASE_SHIFT    = FIXED


#  ******  Please CHECK this constraint. ******
#
#  Please check this constraint with reference to the
#  "LogiCORE Ten Gigabit Ethernet MAC User Guide".
INST "*rx_dcm" PHASE_SHIFT = 56;   # DCM Phase Shift offset for
                                   # xc2v1000fg456-5 Device.
```

This constraint sets a phase shift on the main output of the system DCM with respect to the input clock. This is used to obtain clock/data alignment on the inbound receive interface, whether XGMII or 64-bit SDR. See Chapter 10, Special Design Considerations for a description of the clock scheme, and Appendix B, Calculating the DCM Fixed Phase-Shift Value for instructions on how to set the phase shift value.

```
# The host clock can run at a maximum frequency of 133MHz.
NET "*host_clk_int" TNM_NET = "xgmachostgrp";
TIMESPEC "TS_host_clk" = PERIOD "xgmachostgrp" 7518 ps HIGH 50 %;
```

These two constraints set the period of the clock for the management section.

Flow-Control Constraints

```
############################################################
# Flow control clock crossing timing constraint

INST"*xgmac_core/BU2/U0/G_FLOWCONTROL.flwctrl/G_RX.pause/rx_pause_control_i/good_frame_to_
tx" TNM = "flow_grp";
INST"*xgmac_core/BU2/U0/G_FLOWCONTROL.flwctrl/G_RX.pause/rx_pause_control_i/pause_value_to
_tx_*" TNM = "flow_grp";
INST"*xgmac_core/BU2/U0/G_FLOWCONTROL.flwctrl/G_RX.pause/rx_pause_control_i/pause_req_to_t
x" 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.

Management Constraints

Configuration Registers

```
#########################################################
#  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_HOST.managen/conf/fc_out_*" TNM = "xgmac_config_regs";
INST "*xgmac_core/BU2/U0/G_HOST.managen/conf/rx0_out_*" TNM = "xgmac_config_regs";
INST "*xgmac_core/BU2/U0/G_HOST.managen/conf/rx1_out_*" TNM = "xgmac_config_regs";
INST "*xgmac_core/BU2/U0/G_HOST.managen/conf/tx_out_*" TNM = "xgmac_config_regs";
INST "*xgmac_core/BU2/U0/G_HOST.managen/conf/rs_out_*" TNM = "xgmac_config_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_rs_tig_grp";
INST "*xgmac_core/BU2/U0/rsgen/remote_fail_reg" TNM = "xgmac_rs_tig_grp";
TIMESPEC "TS_rs_tig" = FROM "xgmac_rs_tig_grp" TO "xgmac_config_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.

Statistic Counters

```
### Statistics ###
# Cover the clock domain crossing into and out of the host clock domain; needs
# to be limited to a single tx/rx clock period to guarantee the statistics
# data is stable at the host-side registers on a read.
INST "*xgmac_core/BU2/U0/G_HOST.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 "xgmachostgrp"
6400 ps;

INST "*xgmac_core/BU2/U0/G_HOST.managen/i1.stat/tx_data_reclock_*" TNM =
"xgmac_stats_tx_to_host_sources";
INST "*xgmac_core/BU2/U0/G_HOST.managen/i1.stat/stat_add/data_high_*" TNM =
"xgmac_stats_tx_to_host_sources";
INST "*xgmac_core/BU2/U0/G_HOST.managen/i1.stat/stat_add/data_low_*" TNM =
"xgmac_stats_tx_to_host_sources";
TIMESPEC "TS_stats_tx_to_host" = FROM "xgmac_stats_tx_to_host_sources" TO "xgmachostgrp"
6400 ps;

INST "*xgmac_core/BU2/U0/G_HOST.managen/address_reg_*" TNM = "xgmac_stats_host_sources";
INST "*xgmac_core/BU2/U0/G_HOST.managen/request_reg" TNM = "xgmac_stats_host_sources";
INST "*xgmac_core/BU2/U0/G_HOST.managen/i1.stat/address_rx_*" TNM =
"xgmac_stats_rx_sources";
INST "*xgmac_core/BU2/U0/G_HOST.managen/i1.stat/request_rx" TNM =
"xgmac_stats_rx_sources";
INST "*xgmac_core/BU2/U0/G_HOST.managen/i1.stat/address_tx_*" TNM =
"xgmac_stats_tx_sources";
INST "*xgmac_core/BU2/U0/G_HOST.managen/i1.stat/request_tx" TNM =
"xgmac_stats_tx_sources";
TIMESPEC "TS_stats_host_to_tx" = FROM "xgmac_stats_host_sources" TO
"xgmac_stats_tx_sources" 6400 ps DATAPATHONLY;
TIMESPEC "TS_stats_host_to_rx" = FROM "xgmac_stats_host_sources" TO
"xgmac_stats_rx_sources" 6400 ps DATAPATHONLY;

NET "*xgmac_core/BU2/U0/G_HOST.managen/i1.stat/rx_clk_quarter_small" TNM_NET =
"stats_rx_slowgrp1";
NET "*xgmac_core/BU2/U0/G_HOST.managen/i1.stat/rx_we_reg" TNM_NET = "stats_rx_slowgrp2";
NET "*xgmac_core/BU2/U0/G_HOSTmanagen_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_HOST.managen/i1.stat/tx_clk_quarter_small" TNM_NET =
"stats_tx_slowgrp1";
NET "*xgmac_core/BU2/U0/G_HOST.managen/i1.stat/tx_we_reg" TNM_NET = "stats_tx_slowgrp2";
NET "*xgmac_core/BU2/U0/G_HOST.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 multi-cycle 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.
NET "*xgmac_core/BU2/U0/G_HOST.managen/mdio_master_i/mdc_ce" TNM =
"mdc_grp";
TIMESPEC "TSmdc" = FROM "mdc_grp" TO "mdc_grp" 200 ns;
```

This section constrains the low-speed MDIO logic.

Reset Paths

```
############################################################
# Reset path constraints                                   #
#  These constraints add a measure of protection against   #
#  metastability and skew in the reset nets.              #
############################################################

NET "*xgmac_core/BU2/U0/G_RX_RESET.sync_rx_reset_i/reset_out*" MAXDELAY = 4500 ps;
NET "*xgmac_core/BU2/U0/G_RX_RESET.G_SYNC_RESET_FALLING.sync_rx_reset_1_i/reset_out*"
MAXDELAY = 4500 ps;
NET "*xgmac_core/BU2/U0/G_TX_RESET.sync_tx_reset_i/reset_out*" MAXDELAY = 4500 ps;
NET "*xgmac_core/BU2/U0/G_TX_RESET.sync_flow_ctrl_tx_reset_i/reset_out*" MAXDELAY = 4500
ps;
NET "*xgmac_core/BU2/U0/G_RX_RESET.sync_flow_ctrl_rx_reset_i/reset_out*" MAXDELAY = 4500
ps;
NET "*xgmac_core/BU2/U0/G_SYNC_MGMT_RESET.sync_mgmt_reset_i/reset_out*" MAXDELAY = 4500 ps;
```

This section constrains the synchronized reset paths in the core. See Reset Circuits on page 97 for a discussion of this 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/xgmii_rxd_core*"  IOB          = "TRUE";
INST "rxd_ddr*" IOB = "TRUE";
#  XGMII RXC Constraints.
NET "xgmii_rxc<*>" IOSTANDARD = "HSTL_I";
NET "xgmii_rxc<*>" IOBDELAY = "NONE";
```

```
INST "*xgmii_if/xgmii_rxc_core*"  IOB          = "TRUE";
INST "rxc_ddr*"  IOB = "TRUE";
```

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 will only be used in an XGMII implementation.

Transmitter Constraints

```
##################################################################
These constraints are to ignore false paths within the Transmitter block
##################################################################

# False paths on an internal counter load
INST "*xgmac_core/BU2/U0/G_FLOWCONTROL.flwctrl/tx/data_avail_in_reg_*"
TNM = "xgmac_ifg_false_paths_src_1";
INST "*xgmac_core/BU2/U0/G_FLOWCONTROL.flwctrl/tx/pause_amber_reg" TNM
= "xgmac_ifg_false_paths_src_1";
INST "*xgmac_core/BU2/U0/G_FLOWCONTROL.flwctrl/tx/mux_
control_state_*" TNM = "xgmac_ifg_false_paths_src_1";
INST "*xgmac_core/BU2/U0/G_TX.txgen/tx_controller_inst/
ifg_control_inst/state_*" 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_t
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;
```

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. This circuit provides controllable skews on the reset nets within the design.

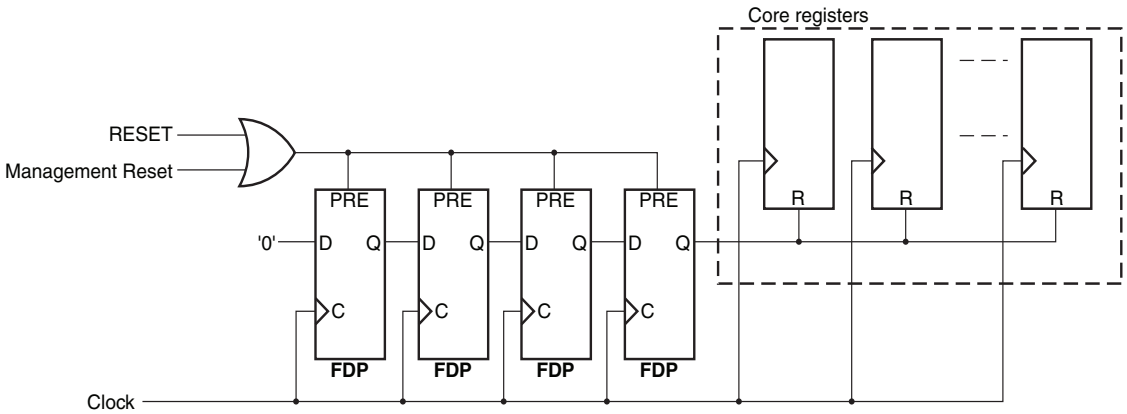


Figure 10-1: **Reset Circuit for a Single Clock/Reset Domain**

Multiple Core Instances

In a large design, it may 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` may 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 will derive 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.

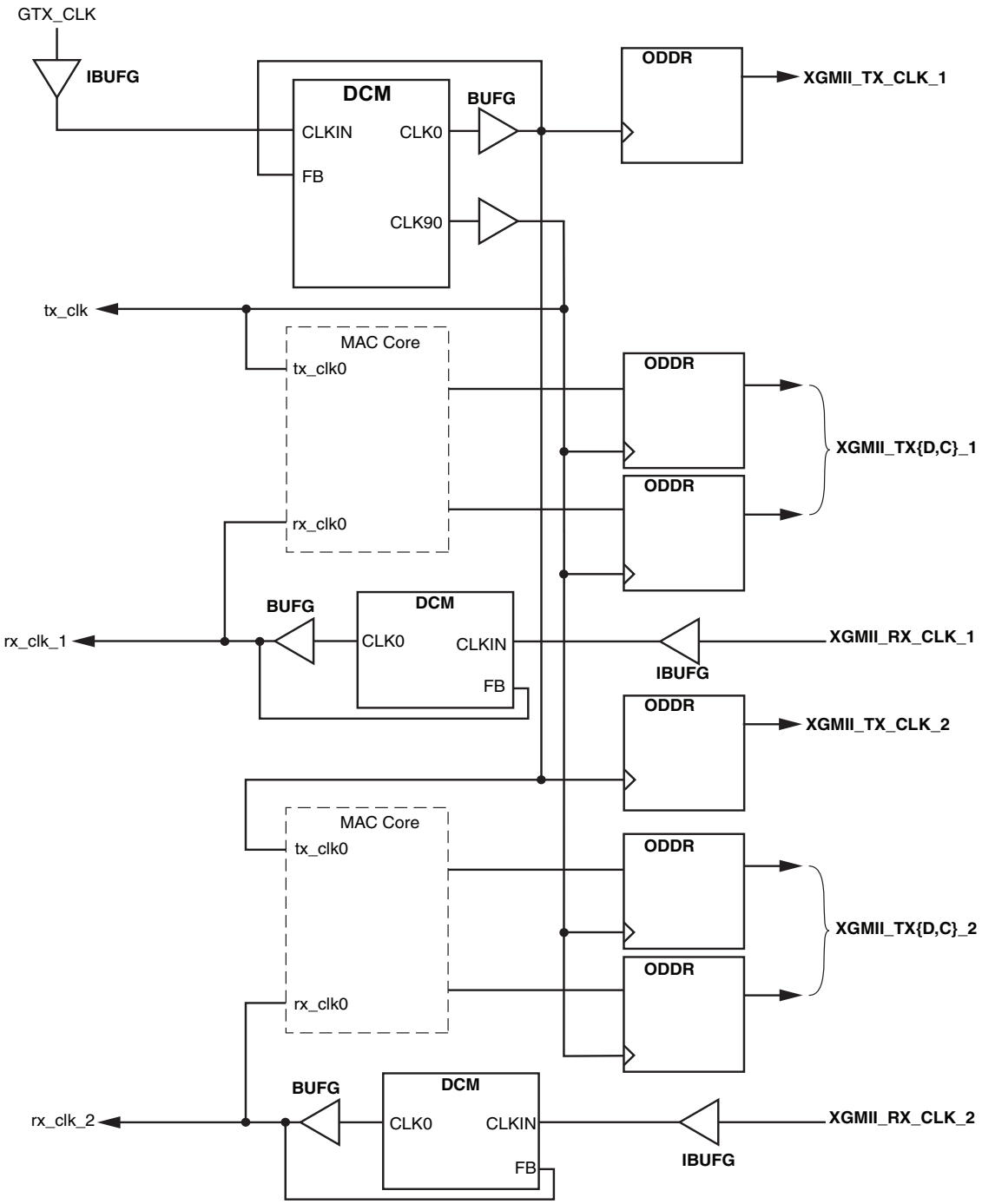


Figure 10-2: **Clock Management, Multiple Instances of the Core with XGMII**

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.

IOs 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/systemio/xaui/index.htm

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

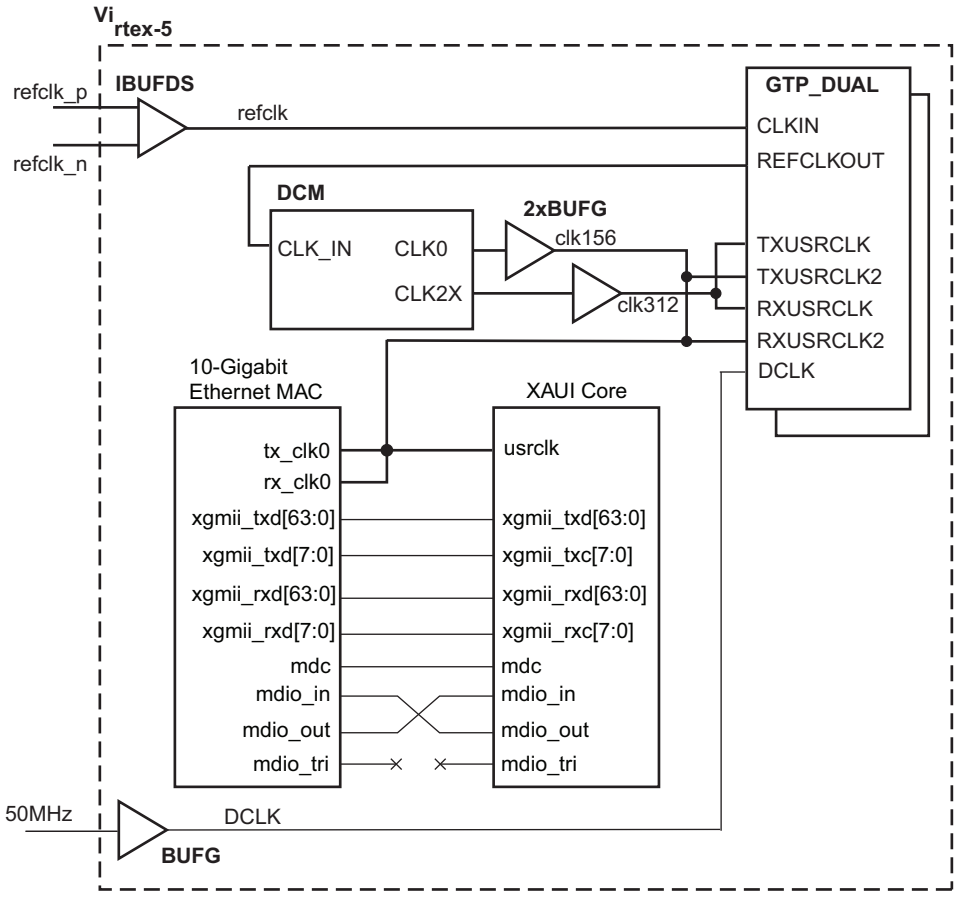


Figure 10-3: **10-Gigabit Ethernet MAC Core Integrated with XAUI Core**

Figure illustrates the connections and clock management logic required to interface the 10-Gigabit Ethernet MAC core to the XAUI core. 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 may be safely removed (although will cause no harm if left).

For more information on clocks and transceiver placement using the XAUI core, see the *LogiCORE XAUI User Guide*.

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_ACK in response to TX_START requests.
- The receiver failing to recognize frames in the inbound data stream.

After the timeout occurs, the core can be reactivated by reconfiguring the FPGA device.

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™ software 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.

Once your entire design is complete, create:

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

To synthesize the design, run:

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

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

XST: Verilog

In the CORE Generator software 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.

Once your entire design is complete, create:

- An XST project file *top_level_module_name*.prj listing all the user source code files. Make sure you include

  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 the *XST User Guide* for more information 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, please consult the Xilinx *Synthesis and Verification Design Guide*, included in your Xilinx software installation.

Other Implementation Information

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

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.

Introduction

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

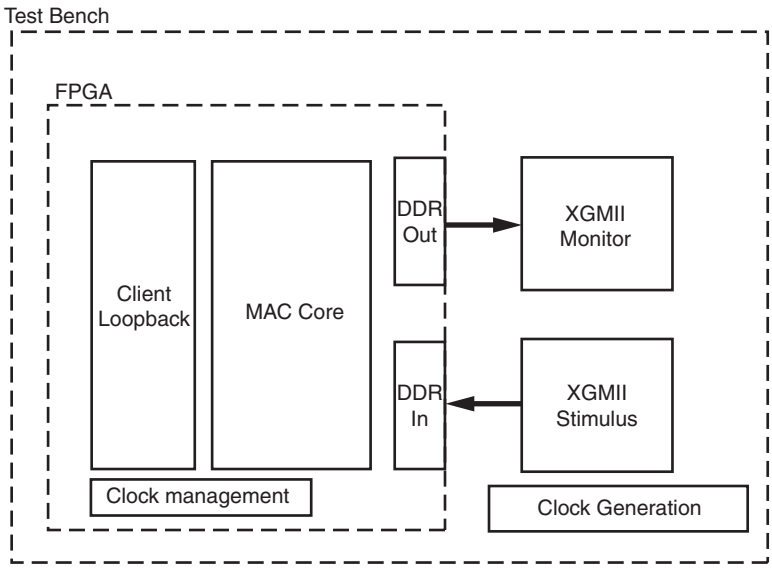


Figure 12-1: **Example Design and Test Bench**

- 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

The 10-Gigabit Ethernet MAC example design has been tested with Xilinx ISE® software v12.1, Mentor Graphics ModelSim v6.5c, Cadence Incisive Enterprise Simulator (IES) v9.2, and Synopsys VCS and VCS MX 2009.12.

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 CORE Generator™ software.

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

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®-5 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 may appear; click OK to continue. The 10-Gigabit Ethernet MAC customization screen appears.

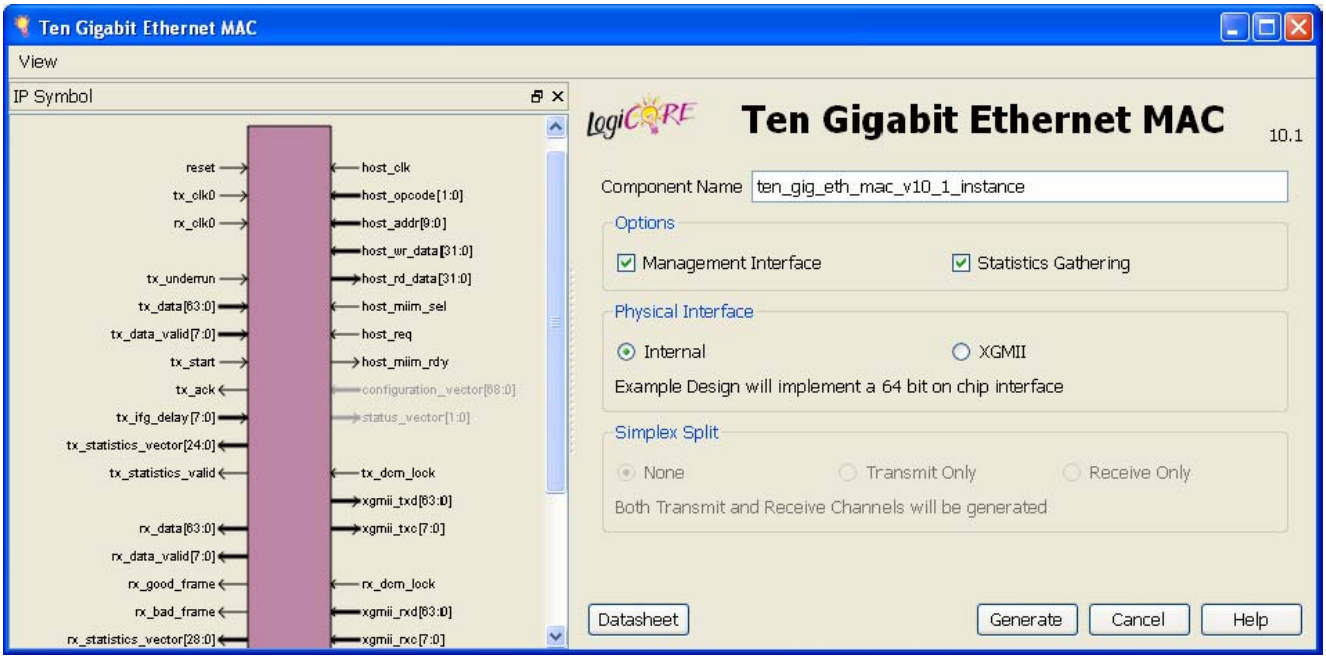


Figure 12-2: **10-Gigabit Ethernet MAC Customization Screen**

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 12, Quick Start 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 the following commands for either Linux or Windows platforms:

Linux

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

Windows

```
> cd <component_name>\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 will either be 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, go to Answer Record 15338 on www.xilinx.com/support for assistance 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.

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™ software, the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

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

<project directory>/<component name>
Core release notes file

<component name>/doc
Product documentation

<component name>/example design
Verilog and VHDL design files

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 CORE Generator software 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 software 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 software.
<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 software
<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 may 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	
ten_gig_eth_mac_ds201.pdf	10-Gigabit Ethernet MAC Data Sheet
ten_gig_eth_mac_gsg146.pdf	10-Gigabit Ethernet MAC Getting Started Guide
ten_gig_eth_mac_ug148.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 LocalLink block 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>*_local_link.vhd	LocalLink level VHDL file. For cores without a simplex split, this instances the LocalLink FIFO in addition to the block level.
*<component_name>*_block.vhd	Block level VHDL file. This instances the appropriate interface block and the MAC core.

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

Name	Description
address_swap.v[hd]	Address swap VHDL or Verilog file. This connects to the LocalLink interface and swaps the destination and source addresses of frame.
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 118.
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 120.
client_loopback.v[hd]	Client loopback design example instanced in the LocalLink level

Back to Top

example_design/fifo

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

Table 13-5: **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
rx_fifo.v[hd]	LocalLink receive FIFO wrapper. This implements the interface to the MAC receiver including frame dropping logic and has a LocalLink interface for the receive channel.
tx_fifo.v[hd]	LocalLink transmit FIFO. This implements the interface to the MAC transmitter including logic to initiate transmission when a full frame is stored in the FIFO and has LocalLink 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 13-6: **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. Once these are run, this directory contains the following files for timing simulation.

Table 13-7: **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 13-8: **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," page 118 or "10-Gigabit Ethernet MAC with 64-bit SDR Interface," page 120.

Back to Top

simulation/functional

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

Table 13-9: **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 13-10: **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 demo 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 the following 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 the following location:

Functional

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

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

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

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 the following 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 designs.

Example Design and HDL Wrapper

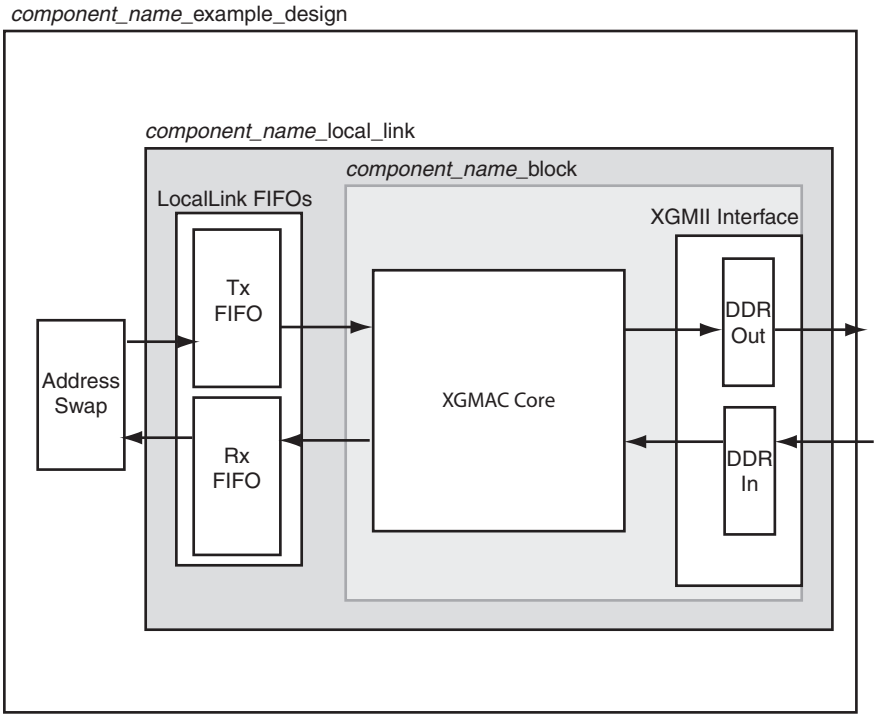


Figure 13-1: **Example Design and HDL Wrapper for 10-Gigabit Ethernet MAC with XGMII Interface**

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 the following 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 the following function:

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

The XGMII block performs the following functions:

- Receiver DCM/MMCM and clock buffer
- DDR logic for the XGMII Interface

Demonstration Test Bench

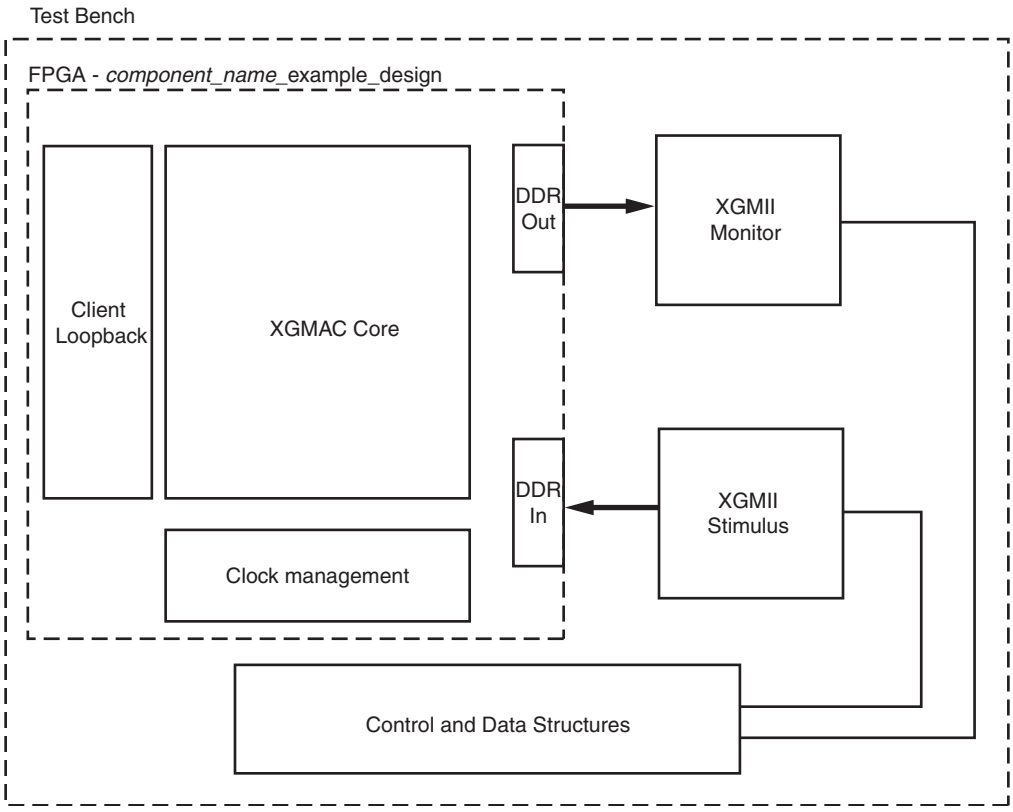


Figure 13-2: **Demonstration Test Bench for 10-Gigabit Ethernet MAC with XGMII Interface**

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

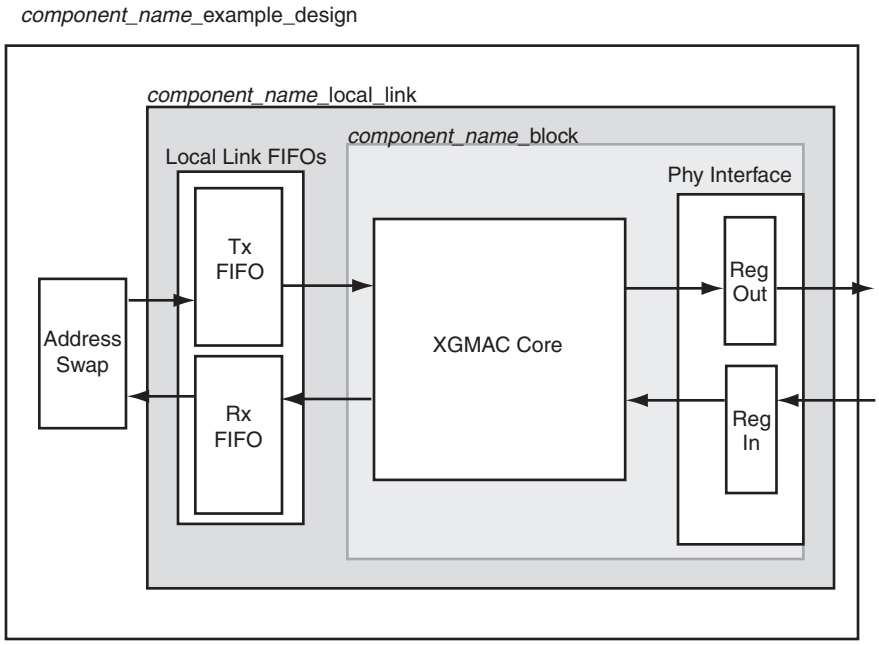


Figure 13-3: **Example Design and HDL Wrapper for 10-Gigabit Ethernet MAC with 64-bit Interface**

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 the following 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 the following function:

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

The physical interface block performs the following functions:

- Registers for the 64-bit SDR interface
- Receiver DCM/MMCM and clock buffer

Demonstration Test Bench

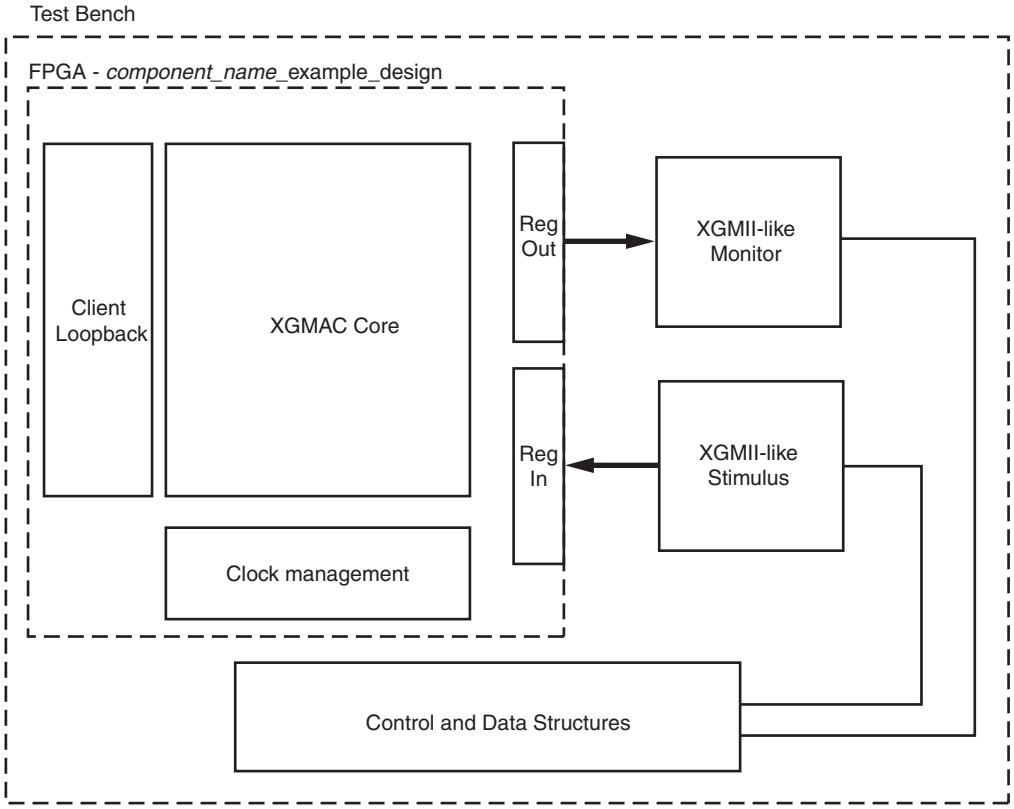


Figure 13-4: **Demonstration Test Bench for 10-Gigabit Ethernet MAC with 64-bit Interface**

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.

Transmit-Only Cores

Example Design and HDL Wrapper

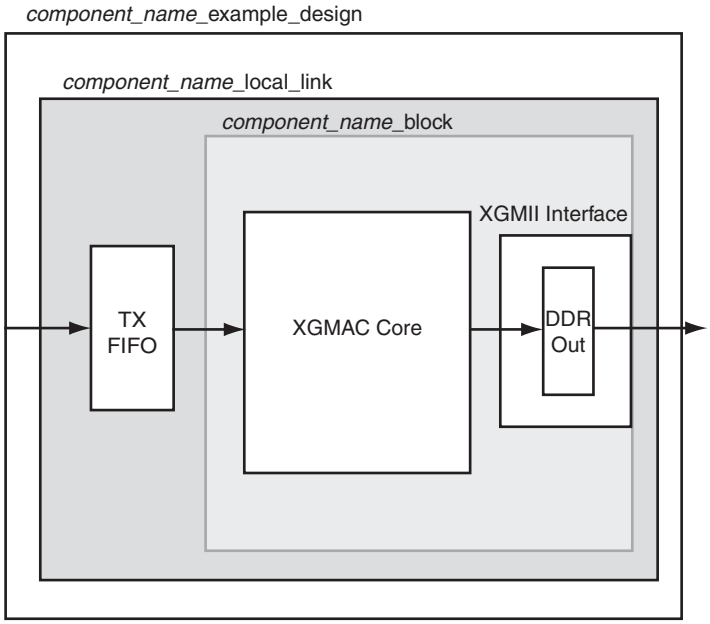


Figure 13-5: **HDL Wrapper for 10-Gigabit Ethernet MAC Transmit-Only Configurations**

The example design and HDL wrapper for transmit-only cores contain the following:

- Registers for either the 64-bit SDR interface or 32-bit DDR interface
- Global clock buffers and Digital Clock Managers (DCMs) or Multi-mode Clock Managers (MMCMs)

Because this is a unidirectional design, there is no client loopback logic in this example design.

Demonstration Test Bench

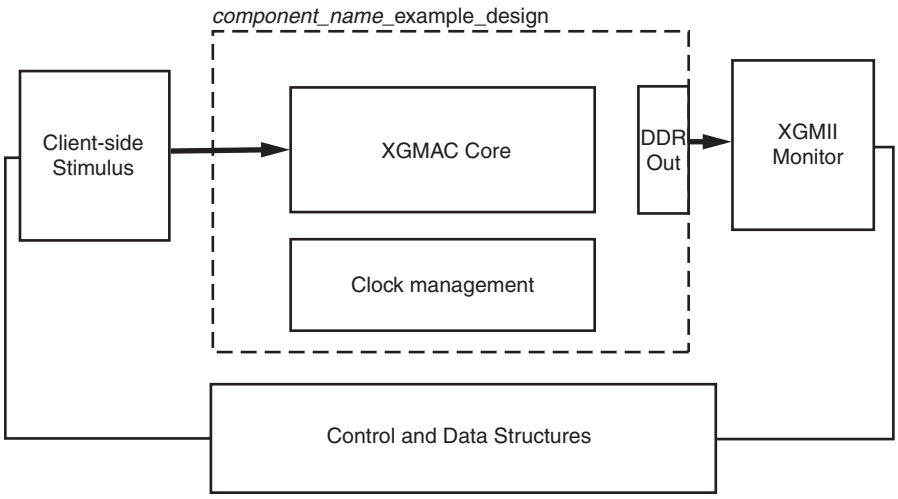


Figure 13-6: **Demonstration Test Bench for 10-Gigabit Ethernet MAC: Transmit Only**

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 major ports of the example design, and a control program that pushes frames of varying length and content through the design and checks the values as they exit the core.

Receive-Only Cores

Example Design and HDL Wrapper

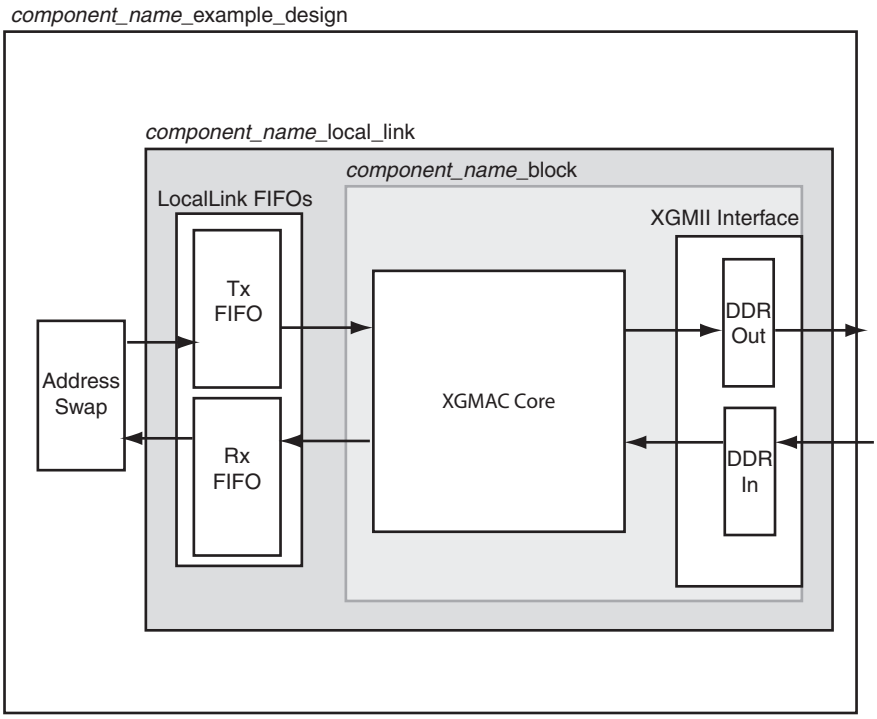


Figure 13-7: **HDL Wrapper for 10-Gigabit Ethernet MAC: Receive Only**

The example design and HDL wrapper for transmit-only cores contain the following:

- Registers for either the 64-bit SDR interface or 32-bit DDR interface
- Global clock buffers and Digital Clock Managers (DCMs) or Multi-Mode Clock Managers (MMCMs)

Because this is a unidirectional design, there is no client loopback logic in this example design.

Demonstration Test Bench

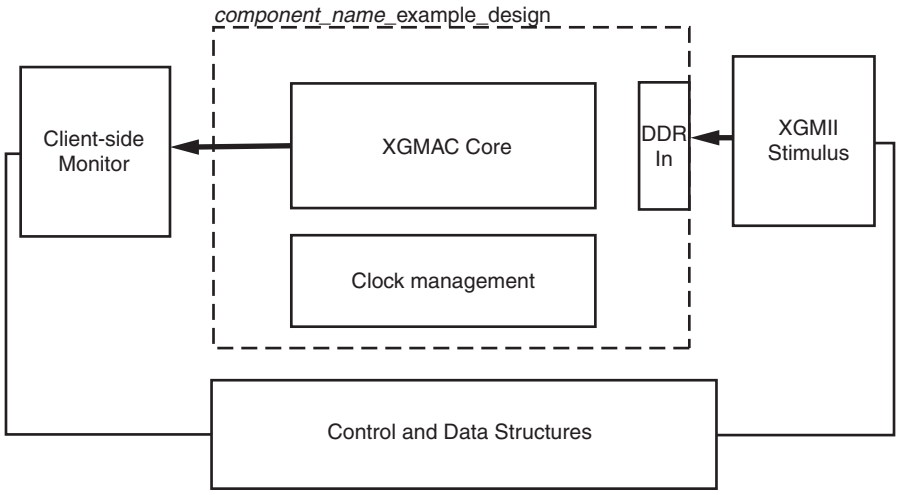


Figure 13-8: **Demonstration Test Bench for 10-Gigabit Ethernet MAC: Receive Only**

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 major ports of the example design, and a control program that pushes frames of varying length and content through the design and checks the values as they exit the core.

Overview of LocalLink Interface

Data is transferred on the LocalLink interface from source to destination, with the flow governed by the four active low control signals `sof_n`, `eof_n`, `src_rdy_n` and `dst_rdy_n`. The flow of data is controlled by the `src_rdy_n` and `dst_rdy_n` signals. Only when these signals are asserted simultaneously is data transferred from source to destination. The individual packet boundaries are marked by the `sof_n` and `eof_n` signals. The `rem[2:0]` signal is used to indicate the position of the sof, always denoted by '001' coincident with `sof_n` being asserted. When `eof_n` is asserted, `rem[2:0]` is encoded to show which of the 8 bytes contain valid frame data (see Table 13-11). For more information on the LocalLink interface see:

www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?iLanguageID=1&key=LocalLink_UserInterface

Figure 13-9 shows the transfer of a frame without flow control.

Table 13-11: **Remainder Binary Encoding**

rem[2:0]	Bytes Valid[7:0]	Data Valid
000	00000001	[7:0]
001	00000011	[15:0]
010	00000111	[23:0]
011	00001111	[31:0]
100	00011111	[39:0]
101	00111111	[47:0]
110	01111111	[55:0]
111	11111111	[63:0]

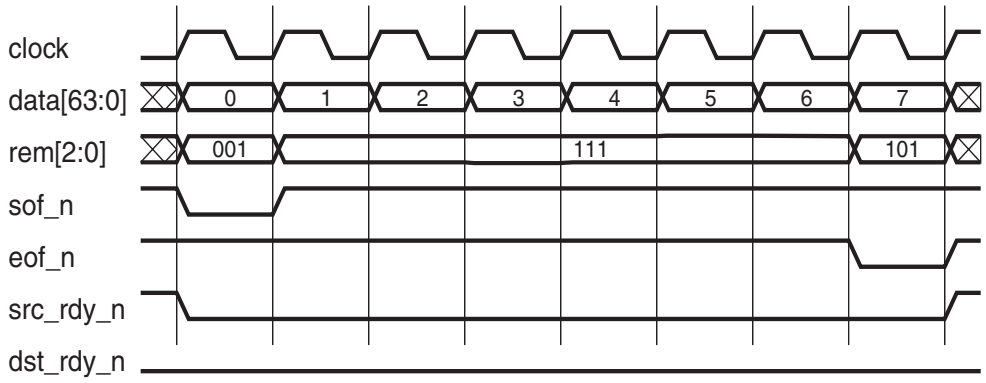


Figure 13-9: **Frame Transfer Across LocalLink Interface**

Figure 13-10 illustrates frame transfer of a frame, where both the `src_rdy_n` and `dst_rdy_n` signals are used to control the flow of data across the interface.

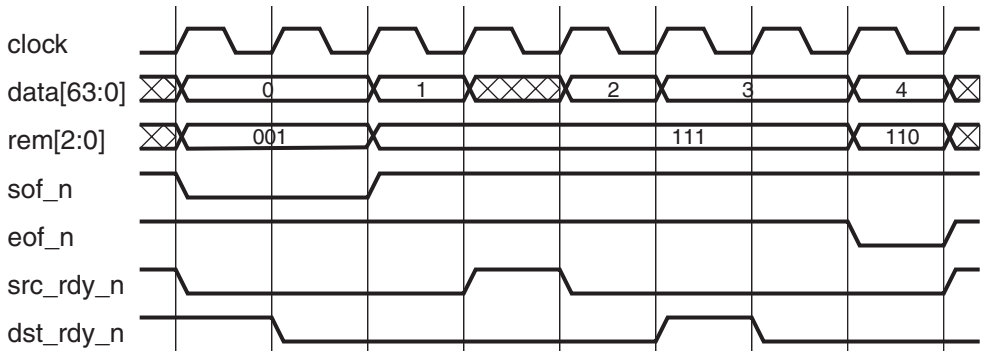


Figure 13-10: **Frame Transfer With Flow Control**

Appendix A

Verification and Interoperability

The 10-Gigabit Ethernet MAC core has been verified using both simulation and hardware testing.

Simulation

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

- 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

Hardware Testing

The core has been used in a number of hardware test platforms within Xilinx. In particular, the core has been used in a test platform design with the Xilinx XAUI core. This design comprises the MAC, XAUI, a *ping* loopback FIFO, and a test pattern generator all under embedded PowerPC® processor control. This design has been used for conformance and interoperability testing at the University of New Hampshire Interoperability Lab.

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. Once 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_data of the Transmit Client interface (until that data appears on xgmii_txd on the XGMII interface), the latency through the core in the transmit direction is 16 clk periods of the tx_clk. This can increase to 16.5 clk periods when the core changes the alignment of data at the XGMII interface.

Receive Path Latency

Measured from the xgmii_rxd port on the XGMII Receive interface (until the data appears on rx_data port of the receiver side Client interface), the latency through the core in the receive direction is 13.5 clock cycles of rx_clk. This can increase to 14 clk periods if the core needs to modify the alignment of data at the Receive Client interface.