

LogiCORE™ Tri-Mode Ethernet MAC v3.4

User Guide

UG138 August 8, 2007





Xilinx is disclosing this Specification to you solely for use in the development of designs to operate on Xilinx FPGAs. Except as stated herein, none of the Specification may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of this Specification may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Specification; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Specification. Xilinx reserves the right to make changes, at any time, to the Specification as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Specification.

THE SPECIFICATION IS PROVIDED "AS IS" WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE SPECIFICATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE SPECIFICATION, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE SPECIFICATION, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREUNDER FOR USE OF THE SPECIFICATION. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE SPECIFICATION TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Specification is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems ("High-Risk Applications"). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Specification in such High-Risk Applications is fully at your risk.

© 2004-2007 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

Tri-Mode Ethernet MAC v3.4 Revision History

The following table shows the revision history for this document.

Date	Version	Revision
9/30/04	1.1	Initial Xilinx release.
4/28/05	2.0	Updated to version 2.1 of the core, Xilinx tools 7.1i, support for Spartan-3E.
1/18/06	2.1	Updated to version 2.2 of the core, release date, and Xilinx tools 8.1i.
7/13/06	3.1	Updated to version 3.1 of the core; Xilinx tools 8.2i.
9/21/06	3.2	Updated to version 3.2 of the core, added support for Spartan-3A.
2/15/07	3.3	Updated to version 3.3 of the core; Xilinx tools 9.1i.
8/8/07	3/4	Updated to version 3.4 of the core; Xilinx tools 9.2i.

Table of Contents

Schedule of Tables	9
Schedule of Figures	11
Preface: About This Guide	
Guide Contents	15
Additional Resources	16
Conventions	17
Typographical	17
Online Document	18
Chapter 1: Introduction	
About the Core	19
Recommended Design Experience	19
Additional Core Resources	19
Related Xilinx Ethernet Products and Services	20
Specifications	20
Technical Support	20
Feedback	20
Tri-Mode Ethernet MAC Core	20
Document	20
Chapter 2: Core Architecture	
System Overview	21
Core Components	21
Core Interfaces	23
Optional Interfaces	23
Client Side Interface Signals	27
Management Interface Signals	29
Configuration Vector Signals	29
Address Filter Signals	30
Clock, Speed Indication, and Reset Signals	30
Physical Interface Signals	31
Optional MDIO Signals	32
Chapter 3: Generating the Core	
GUI Interface	33
Component Name	33
Management Interface	33
Clock Enables	34
Address Filter	34
Number of Address Table Entries	34

Physical Interface	34
Parameter Values in the XCO File	34
Output Generation	35

Chapter 4: Designing with the Core

General Design Guidelines	37
Design Steps	37
Using the HDL Example Design as User Top-level	37
Using the HDL Example Design in a User Design	37
Understand Signal Pipelining	38
Register All I/Os	38
Recognize Timing Critical Signals	38
Use Supported Design Flows	38
Make Only Allowed Modifications	38

Chapter 5: Using the Client Side Data Path

Receiving Inbound Frames	39
Normal Frame Reception	39
Without Clock Enables	39
Using Clock Enables	41
emacclientrxgoodframe and emacclientrxbadframe Timing	42
Frame Reception with Errors	42
Client-Supplied FCS Passing	42
VLAN Tagged Frames	43
Maximum Permitted Frame Length	43
Length/Type Field Error Checks	44
Enabled	44
Disabled	44
Address Filter	44
Receiver Statistics Vector	45
Transmitting Outbound Frames	47
Normal Frame Transmission	47
Without Clock Enables	47
Using Clock Enables	48
Padding	49
Client-Supplied FCS Passing	50
Client Underrun	50
Back-to-Back Transfers	51
VLAN Tagged Frames	52
Maximum Permitted Frame Length	53
Frame Collisions: Half-Duplex Operation Only	53
Interframe Gap Adjustment: Full-Duplex Mode Only	54
Transmitter Statistics Vector	55

Chapter 6: Using Flow Control

Overview of Flow Control	57
Flow Control Requirement	57
Flow Control Basics	58
Pause Control Frames	58
Flow Control Operation of the TEMAC	59
Transmitting a Pause Control Frame	59

Core-initiated Pause Request	59
Client-initiated Pause Request	60
Receiving a Pause Control Frame	60
Core-initiated Response to a Pause Request	60
Client-initiated Response to a Pause Request	61
Flow Control Implementation Example	61

Chapter 7: Using the Physical Side Interface

Implementing External GMII	63
GMII/MII Transmit Interface	63
Virtex-II Pro, Virtex-II, Spartan-3, Spartan-3E, and Spartan-3A Devices	63
Virtex-4 and Virtex-5 Devices	65
GMII/MII Receive Interface	66
Virtex-II and Virtex-II Pro Devices	66
Spartan-3, Spartan-3E, and Spartan-3A Devices	66
Virtex-4 Devices	68
Virtex-5 Devices	69
Implementing External RGMII	70
RGMII Transmit Interface	70
Virtex-II Pro, Virtex-II, Spartan-3, and Spartan-3A Devices	70
Virtex-4 Devices	71
Virtex-5 Devices	73
RGMII Receiver Interface	74
Virtex-II Pro, Virtex-II, Spartan-3, and Spartan-3A Devices	74
Virtex-4 Devices	74
Virtex-5 Devices	75
RGMII Inband Status Decoding Logic	77
Using the MDIO Interface	77
Connecting the MDIO to an Internally Integrated PHY	77
Connecting the MDIO to an External PHY	78
Connecting the MDIO to an External and Internal PHY	78

Chapter 8: Configuration and Status

Using the Optional Management Interface	81
hostclk	81
Configuration Registers	81
Register Maps	82
Using the Management Interface	88
Accessing Configuration	88
MDIO Interface	91
Introduction to MDIO	91
Write Transaction	92
Read Transaction	92
Accessing MDIO via the TEMAC	92
Accessing Configuration without the Management Interface	94
Configuration Vector Description	95

Chapter 9: Constraining the Core

Required Constraints	101
Device, Package, and Speedgrade Selection	101

I/O Location Constraints	101
Placement Constraints	101
Timing Constraints	101
PERIOD(s) for Clock nets	101
Timespecs for Reset Logic within the Core	104
Constraints when Implementing an External GMII	104
Understanding Timing Reports for GMII Setup/Hold timing	107
Spartan-3 Devices	107
Virtex-II or Virtex-II Pro Devices	108
Virtex-4 or Virtex-5 Devices with Delayed Data/Control	108
Virtex-4 or Virtex-5 Devices with Delayed Clock	109
Constraints when Implementing an External RGMII	110
RGMII DDR Constraints	113
Understanding Timing Reports for RGMII Setup/Hold timing	113
None Virtex-4 or Virtex-5 Devices	113
Virtex-4 or Virtex-5 Devices with Delayed Data/Control	114
Virtex-4 or Virtex-5 Devices with Delayed Clock	115

Chapter 10: Clocking and Reset

Clocking	117
GMII/MII Transmit Clock Generation	117
GMII/MII Receive Clock Generation	119
RGMII Transmit Clock Generation	120
RGMII Receive Clock Generation	122
Multiple Cores	123
Clock Sharing	123
BUFGMUX Usage	124
Reset Conditions	126

Chapter 11: Interfacing to Other Cores

Integrating with the Ethernet 1000BASE-X PCS/PMA or SGMII Core	127
Integration to Provide SGMII	127
Virtex-II Pro Devices	127
Virtex-4 Devices	129
Virtex-5 Devices	130
Integrating with the Ethernet Statistics Core	130

Chapter 12: Implementing Your Design

Pre-implementation Simulation	133
Using the Simulation Model	133
Synthesis	133
XST - VHDL	133
XST - Verilog	134
Implementation	134
Generating the Xilinx Netlist	134
Mapping the Design	134
Placing and Routing the Design	134
Static Timing Analysis	135
Generating a Bitstream	135

Post-Implementation Simulation	135
Generating a Simulation Model	135
Using the Model	135
Other Implementation Information	136

Appendix A: Using the Client Side FIFO

Overview of LocalLink Interface	137
Receive FIFO Operation	138
LocalLink Interface	138
Transmit FIFO Operation	139
LocalLink Interface	139
Clock Requirements	140
User Interface Data Width Conversion	140

Appendix B: Core Verification, Compliance, and Interoperability

Verification by Simulation	141
Hardware Verification	141

Appendix C: Core Latency

General	143
Transmit Path Latency	143
Receive Path Latency	143

Appendix D: Calculating the DCM Phase Shift

DCM Phase Shifting Requirements	145
Finding the Ideal Phase Shift Value	145

Schedule of Tables

Chapter 2: Core Architecture

Table 2-1: Client Interface Signal Pins	27
Table 2-2: Optional Management Interface Signal Pinout	29
Table 2-3: Alternative to the Optional Management Interface: Configuration Vector Signal Pinout	29
Table 2-4: Address Filter Unicast Address	30
Table 2-5: Clock, Speed Indication and Reset Signals	30
Table 2-6: GMII/MII Interface Signal Pinout	31
Table 2-7: MDIO Interface Signal Pinout	32

Chapter 3: Generating the Core

Table 3-1: XCO File Values and Default Values	34
---	----

Chapter 5: Using the Client Side Data Path

Table 5-1: Abbreviations Used in Timing Diagrams	39
Table 5-2: Bit Definition for the Receiver Statistics Vector	45
Table 5-3: Bit Definition for the Transmitter Statistics Vector	55

Chapter 8: Configuration and Status

Table 8-1: Management Interface Transaction Types	81
Table 8-2: Configuration Registers	82
Table 8-3: Receiver Configuration Word 0	82
Table 8-4: Receiver Configuration Word 1	83
Table 8-5: Transmitter Configuration Word	84
Table 8-6: Flow Control Configuration Word	84
Table 8-7: Management Configuration Word	85
Table 8-8: MAC Speed Configuration Word	85
Table 8-9: Unicast Address (Word 0)	86
Table 8-10: Unicast Address (Word 1)	86
Table 8-11: Address Table Configuration (Word 0)	86
Table 8-12: Address Table Configuration (Word 1)	87
Table 8-13: Address Filter Mode	87
Table 8-14: Configuration Vector Bit Definition	95

Chapter 9: Constraining the Core

<i>Table 9-1: Input GMII Timing</i>	105
<i>Table 9-2: Input RGMII Timing</i>	111

Appendix A: Using the Client Side FIFO

<i>Table A-1: Receive FIFO LocalLink Interface</i>	138
<i>Table A-2: Transmit FIFO LocalLink Interface</i>	139

Schedule of Figures

Chapter 2: Core Architecture

Figure 2-1: Tri-Mode Ethernet MAC Block Diagram.....	21
Figure 2-2: Component Pinout for MAC with Optional Management Interface (clock_enables = false).....	23
Figure 2-3: Component Pinout for MAC without Optional Management Interface (clock_enables = false).....	24
Figure 2-4: Component Pinout for MAC with Optional Management Interface (clock_enables = true)	25
Figure 2-5: Component Pinout for MAC without Optional Management Interface (clock_enables = true)	26

Chapter 3: Generating the Core

Figure 3-1: Core Customization Screen	33
---	----

Chapter 5: Using the Client Side Data Path

Figure 5-1: Normal Frame Reception	40
Figure 5-2: Normal Frame Reception at 1 Gbps with Optional Clock Enables	41
Figure 5-3: Normal Frame Reception at 10/100 Mbps with Optional Clock Enables ...	41
Figure 5-4: Frame Reception with Error.....	42
Figure 5-5: Frame Reception with In-Band FCS Field.....	43
Figure 5-6: Reception of a VLAN Tagged Frame	43
Figure 5-7: Receiver Statistics Vector Timing.....	45
Figure 5-8: Normal Frame Transmission Across Client Interface.....	48
Figure 5-9: Normal Frame Transmission at 1000 Mbps with Optional Clock Enables .	48
Figure 5-10: Normal Frame Transmission at 10/100 Mbps with Optional Clock Enables	49
Figure 5-11: Frame Transmission with Client-supplied FCS.....	50
Figure 5-12: Frame Transmission with Underrun	51
Figure 5-13: Back-to-Back Frame Transmission	52
Figure 5-14: Transmission of a VLAN Tagged Frame.....	52
Figure 5-15: Collision Handling: Frame Retransmission Required	53
Figure 5-16: Collision Handling: No Frame Retransmission Required.....	54
Figure 5-17: Interframe Gap Adjustment.....	54
Figure 5-18: Transmitter Statistics Vector Timing.....	55

Chapter 6: Using Flow Control

Figure 6-1: The Requirement for Flow Control	57
Figure 6-2: MAC Control Frame Format	58
Figure 6-3: Pause Request Timing.....	59

<i>Figure 6-4: Pause Request Timing with Clock Enables</i>	59
<i>Figure 6-5: Flow Control Implementation Triggered from FIFO Occupancy</i>	62

Chapter 7: Using the Physical Side Interface

<i>Figure 7-1: External GMII/MII Transmit Interface</i>	64
<i>Figure 7-2: External GMII/MII Transmit Interface in a Virtex-4/Virtex-5 Device</i>	65
<i>Figure 7-3: External GMII/MII Receive Interface</i>	66
<i>Figure 7-4: GMII/MII Receive Logic for Spartan-3, Spartan-3E, and Spartan-3A Devices</i>	67
<i>Figure 7-5: GMII/MII Receive Logic for Virtex-4 Devices</i>	68
<i>Figure 7-6: GMII/MII Receive Logic for Virtex-5 Devices</i>	69
<i>Figure 7-7: External RGMII Transmit Interface</i>	70
<i>Figure 7-8: External RGMII Transmit Interface in a Virtex-4 Device</i>	71
<i>Figure 7-9: External RGMII Transmit Interface in a Virtex-5 Device</i>	73
<i>Figure 7-10: External RGMII Receive Interface</i>	74
<i>Figure 7-11: External RGMII Receive Interface in Virtex-4 Devices</i>	75
<i>Figure 7-12: External RGMII Receive Interface in Virtex-5 Devices</i>	76
<i>Figure 7-13: RGMII Inband Status Logic</i>	77
<i>Figure 7-14: External MDIO Interface</i>	78
<i>Figure 7-15: Internal and External MDIO Interfaces</i>	79

Chapter 8: Configuration and Status

<i>Figure 8-1: Configuration Register Write Timing</i>	88
<i>Figure 8-2: Configuration Register Read Timing</i>	89
<i>Figure 8-3: Address Table Write Timing</i>	90
<i>Figure 8-4: Address Table Read Timing</i>	91
<i>Figure 8-5: MDIO Write Transaction</i>	92
<i>Figure 8-6: MDIO Read Transaction</i>	92
<i>Figure 8-7: MDIO Access Through Management Interface</i>	93

Chapter 9: Constraining the Core

<i>Figure 9-1: Input GMII Timing</i>	105
<i>Figure 9-2: Timing Report Setup/Hold</i>	110
<i>Figure 9-3: Input RGMII Timing</i>	111
<i>Figure 9-4: Timing Report Setup/Hold</i>	116

Chapter 10: Clocking and Reset

<i>Figure 10-1: GMII/MII Transmit Clock Generator</i>	117
<i>Figure 10-2: 10/100 Mbps MII Transmit Clock Generator</i>	118
<i>Figure 10-3: GMII/MII Transmit Clock Generator (clock_enables = true)</i>	118
<i>Figure 10-4: GMII/MII Receive Clock Generator</i>	119
<i>Figure 10-5: 10/100 Mbps MII Receive Clock Generator</i>	119
<i>Figure 10-6: GMII/MII Receive Clock Generator (clock_enables = true)</i>	120

<i>Figure 10-7: RGMII Transmit Clock Generator (clock_enables = false)</i>	120
<i>Figure 10-8: RGMII Transmit Clock Generator (clock_enables = true)</i>	121
<i>Figure 10-9: RGMII Transmit Clock Generator (clock_enables = false) for Virtex-5</i> . .	121
<i>Figure 10-10: RGMII Receive Clock Generator (clock_enables = false)</i>	122
<i>Figure 10-11: RGMII Receive Clock Generator (clock_enables = true)</i>	122
<i>Figure 10-12: RGMII Receive Clock Generator (clock_enables = false) for Virtex-4 and Virtex-5 Devices</i>	123
<i>Figure 10-13: Clock Sharing across Two MAC Cores</i>	124
<i>Figure 10-14: Suggested BUFGMUX Scheme</i>	125
<i>Figure 10-15: Alternative BUFGMUX Scheme</i>	125
<i>Figure 10-16: Reset Circuit for One Clock/reset Domain</i>	126

Chapter 11: Interfacing to Other Cores

<i>Figure 11-1: Tri-Mode Ethernet MAC Extended to Implement SGMII (Virtex-II Pro)</i>	128
<i>Figure 11-2: Tri-Mode Ethernet MAC Extended to Implement SGMII (Virtex-4)</i>	129
<i>Figure 11-3: Tri-Mode Ethernet MAC Extended to Implement SGMII (Virtex-5)</i>	130
<i>Figure 11-4: Tri-Mode Ethernet MAC with Statistics</i>	131

Appendix A: Using the Client Side FIFO

<i>Figure A-1: Typical 10M/100M/1G Ethernet FIFO Implementation</i>	137
<i>Figure A-2: Frame Transfer across LocalLink Interface</i>	138
<i>Figure A-3: Frame Transfer with Flow Control</i>	138

About This Guide

The *Tri-Mode Ethernet MAC v3.4 User Guide* describes the function and operation of the LogiCORE™ Tri-Mode Ethernet MAC (TEMAC) core, as well as information about designing, customizing, and implementing the core.

Guide Contents

This guide contains the following chapters:

- [“Preface, About this Guide”](#) introduces the organization and purpose of the user guide and provides a list of additional resources and conventions used in this document.
- [Chapter 1, “Introduction”](#) describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- [Chapter 2, “Core Architecture”](#) provides an overview of the core and discusses the signal interface.
- [Chapter 3, “Generating the Core”](#) describes how to generate the core and defines customization options.
- [Chapter 4, “Designing with the Core”](#) provides general guidelines for creating designs using the core.
- [Chapter 5, “Using the Client Side Data Path”](#) provides information about using the client-side interface of the core.
- [Chapter 6, “Using Flow Control”](#) details the flow control capabilities of the core.
- [Chapter 7, “Using the Physical Side Interface”](#) describes how to use the core to provide GMII/MII, RGMII and MDIO functionality.
- [Chapter 8, “Configuration and Status”](#) describes how to operate the Management Interface.
- [Chapter 9, “Constraining the Core”](#) describes constraints in the design.
- [Chapter 10, “Clocking and Reset”](#) discusses suggested clocking schemes and reset circuitry.
- [Chapter 11, “Interfacing to Other Cores,”](#) describes how to interface the core to the Ethernet 1000BASE-X PCS/PMA or SGMII core in order to provide SGMII functionality. In addition, the integration of the core with the Ethernet Statistics Core is discussed.
- [Chapter 12, “Implementing Your Design”](#) provides instructions for setting up the synthesis, simulation, and implementation environment, and how to generate a bitstream.

- [Appendix A, “Using the Client Side FIFO”](#) describes the operation of the FIFO included in the core example design.
- [Appendix B, “Core Verification, Compliance, and Interoperability”](#) describes how the core was verified and certified for compliance, as well as its interoperability with other devices.
- [Appendix C, “Core Latency”](#) describes the core latency.
- [Appendix D, “Calculating the DCM Phase Shift”](#) provides instructions for calculating a DCM phase shift value to meet input setup and hold timing.

Additional Resources

For additional information, go to www.xilinx.com/support. The following table lists some of the resources you can access from this website or by using the provided URLs.

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

Conventions

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

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
Courier font	Messages, prompts, and program files that the system displays	speed grade: - 100
Courier bold	Literal commands you enter in a syntactical statement	ngdbuild design_name
<i>Italic font</i>	References to other manuals	See the <i>User Guide</i> for details.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are <i>not</i> connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets []	An optional entry or parameter. However, in bus specifications, such as bus[7 : 0] , they are required.	ngdbuild [option_name] design_name
Brackets <>	User-defined variable, for example, a directory or project name.	<project directory>
Braces { }	A list of items from which you must choose one or more	lowpwr = {on off}
Vertical bar	Separates items in a list of choices	lowpwr = {on off}
Vertical ellipsis .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...		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 linking conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section “ Additional Resources ” for details. See “ Title Formats ” in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the <i>Virtex-II Handbook</i> .
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

Introduction

The Tri-Mode Ethernet MAC (TEMAC) core is a fully verified solution design that supports Verilog-HDL and VHDL. In addition, the example design provided with the core is in both Verilog and VHDL.

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

About the Core

The TEMAC core is a Xilinx CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see www.xilinx.com/systemio/temac/index.htm. For information about system requirements and licensing the core, see Chapter 2, “Licensing the Core,” in the *Getting Started Guide*.

Recommended Design Experience

Although the TEMAC core is a fully verified solution, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application. For best results, previous experience building high performance, pipelined FPGA designs using Xilinx implementation software and User Constraint Files (UCF) is recommended.

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

Additional Core Resources

For more details and updates on the TEMAC core, see the following documents, located on the Xilinx Tri-Mode Ethernet MAC product page, accessible from the www.xilinx.com/systemio/temac/index.htm

- *Tri-Mode Ethernet MAC Data Sheet*
- *Tri-Mode Ethernet MAC Release Notes*
- *Tri-Mode Ethernet MAC Getting Started Guide*

For updates to this document, see the *Tri-Mode Ethernet MAC User Guide* located on the Tri-Mode Ethernet MAC product page.

Related Xilinx Ethernet Products and Services

See the Ethernet Products and Services page at:

www.xilinx.com/products/design_resources/conn_central/grouping/ethernet.htm

Specifications

- IEEE 802.3-2002
- Reduced Gigabit Media Independent Interface (RGMI) version 2.0

Technical Support

For technical support, see support.xilinx.com/. Questions are routed to a team of engineers with expertise using the TEMAC core.

Xilinx will provide technical support for use of this product as described in the *Tri-Mode Ethernet MAC User Guide* and the *Tri-Mode 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 TEMAC core and the documentation supplied with the core.

Tri-Mode Ethernet MAC Core

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

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

Document

For comments or suggestions about the core, please submit a WebCase from www.xilinx.com/support/clearxpress/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

Core Architecture

This chapter describes the TEMAC core architecture including all interfaces and the major functional blocks.

System Overview

Figure 2-1 illustrates a block diagram of the TEMAC core.

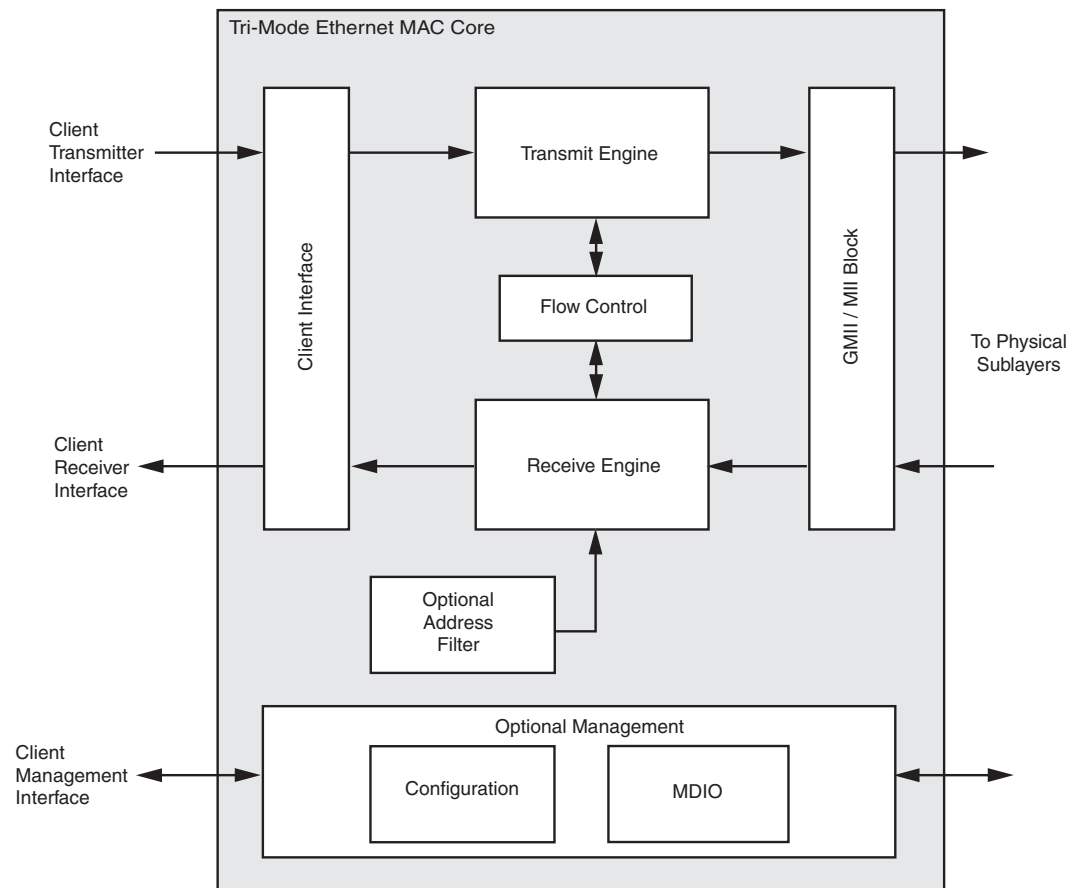


Figure 2-1: Tri-Mode Ethernet MAC Block Diagram.

Core Components

The major functional blocks of the MAC are:

- Client interface

- Transmit engine
- Flow control block
- Receive engine
- Optional Management Interface and MDIO
- GMII/MII interface
- Optional Address Filter

The client interface has fully independent 8-bit interfaces for both transmit and receive to support full-duplex operation.

Configuration of the core and access to the MDIO port are accessed through the optional Management Interface, a 32 bit processor-neutral data pathway that is independent of the Ethernet data pathway. When the Management Interface is omitted, configuration of the core can still be made via an alternative configuration vector.

Core Interfaces

Optional Interfaces

Figure 2-2 shows the pinouts with the optional Management Interface when the core is built with the `clock_enables` option set to false

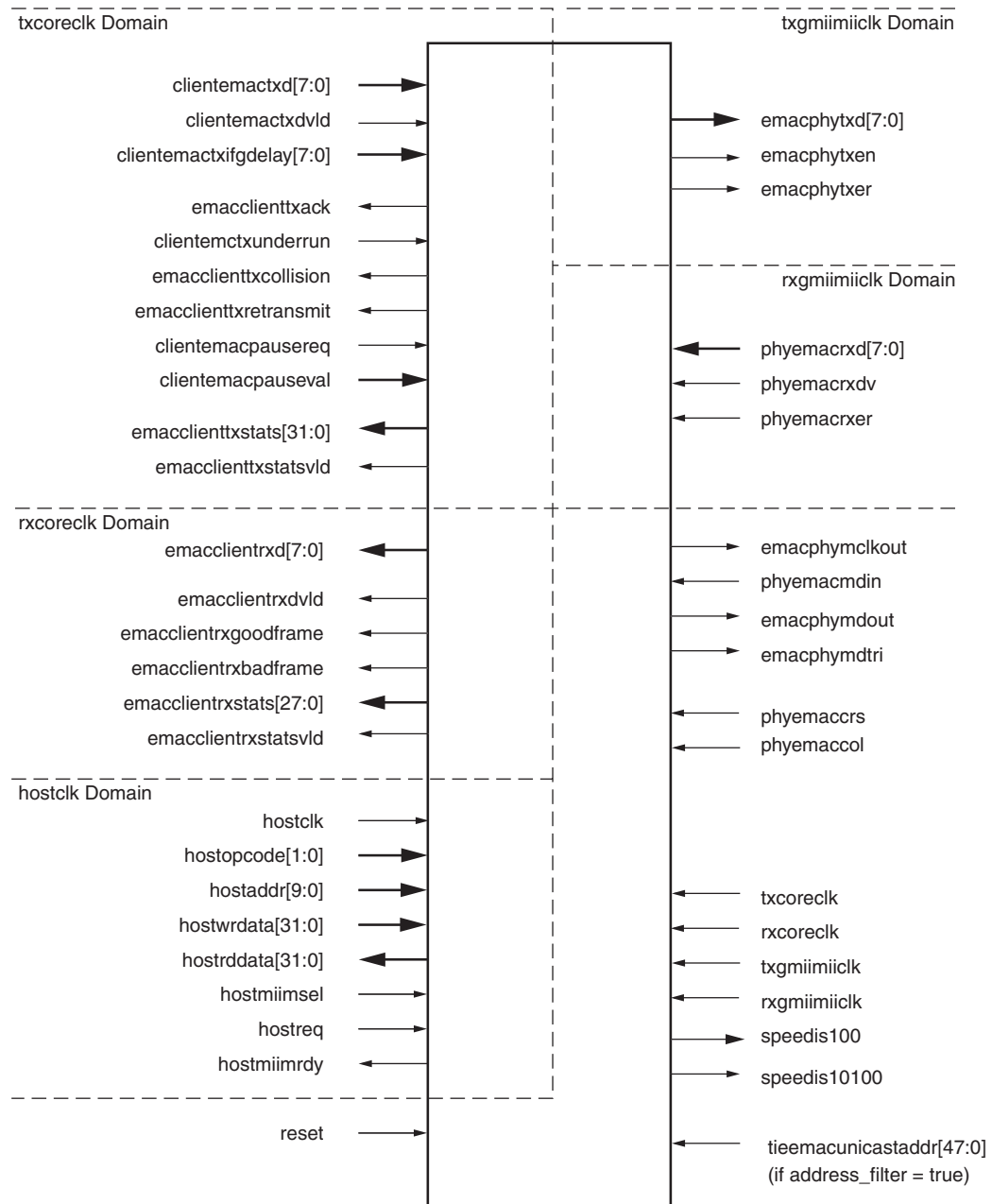


Figure 2-2: Component Pinout for MAC with Optional Management Interface (`clock_enables` = false)

Figure 2-3 shows the pinouts without the optional Management Interface when the core is built with the `clock_enables` option set to false.

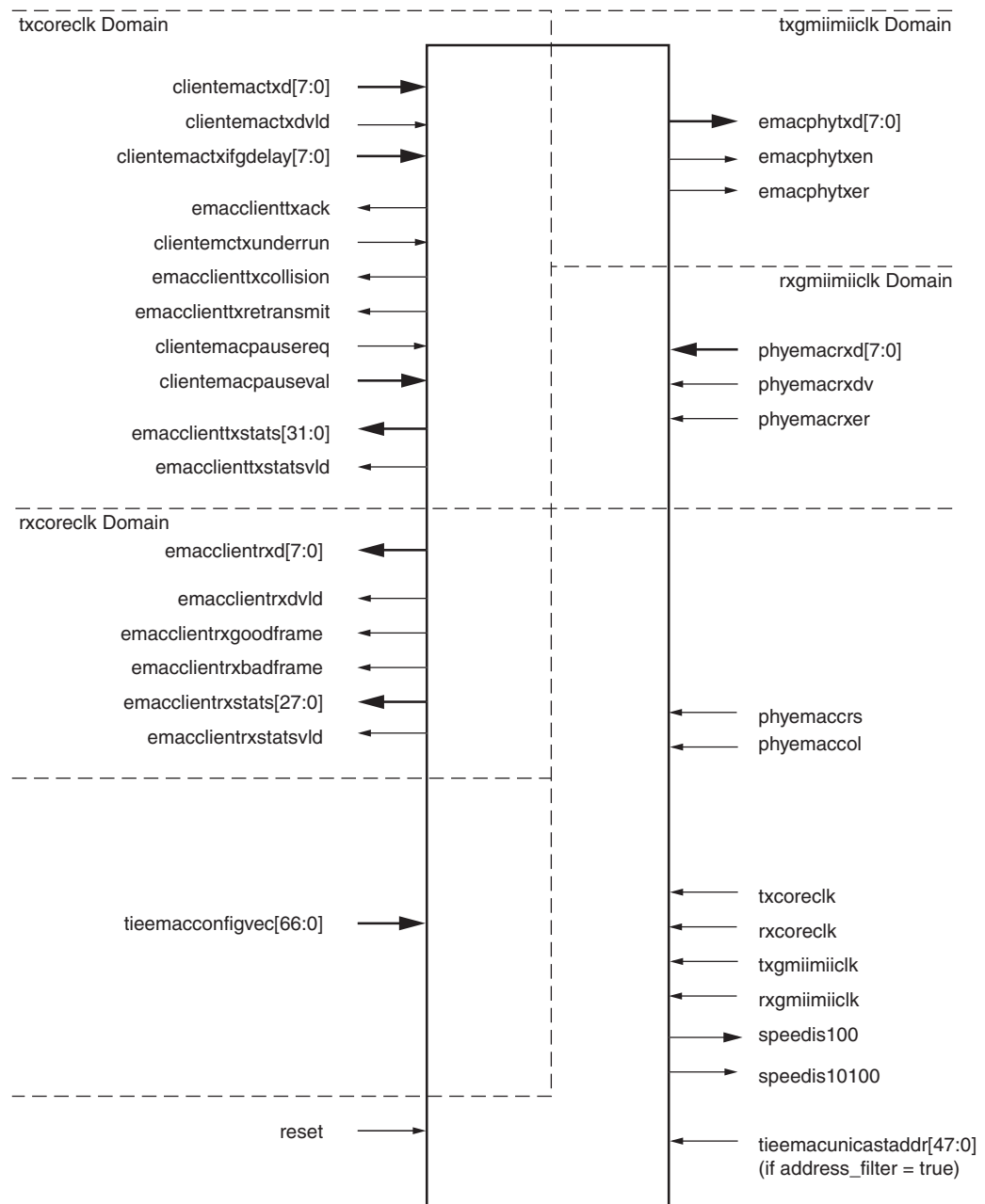


Figure 2-3: Component Pinout for MAC without Optional Management Interface (`clock_enables` = false)

Figure 2-4 shows the pinouts with the optional Management Interface when the core is built with the `clock_enables` option set to true.

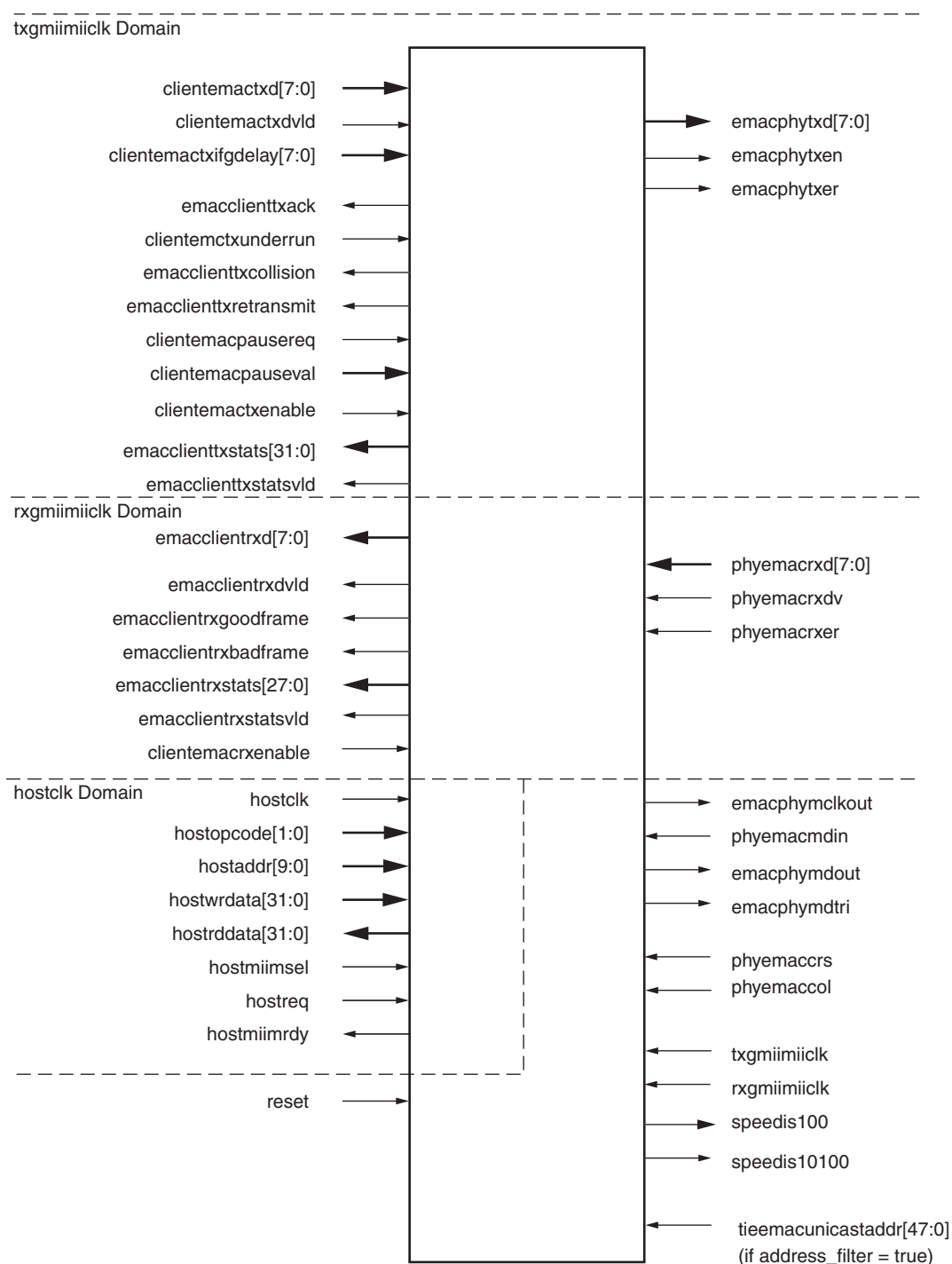


Figure 2-4: Component Pinout for MAC with Optional Management Interface (`clock_enables = true`)

Figure 2-5 shows the pinouts without the optional Management Interface when the core is built with the `clock_enables` option set to true.

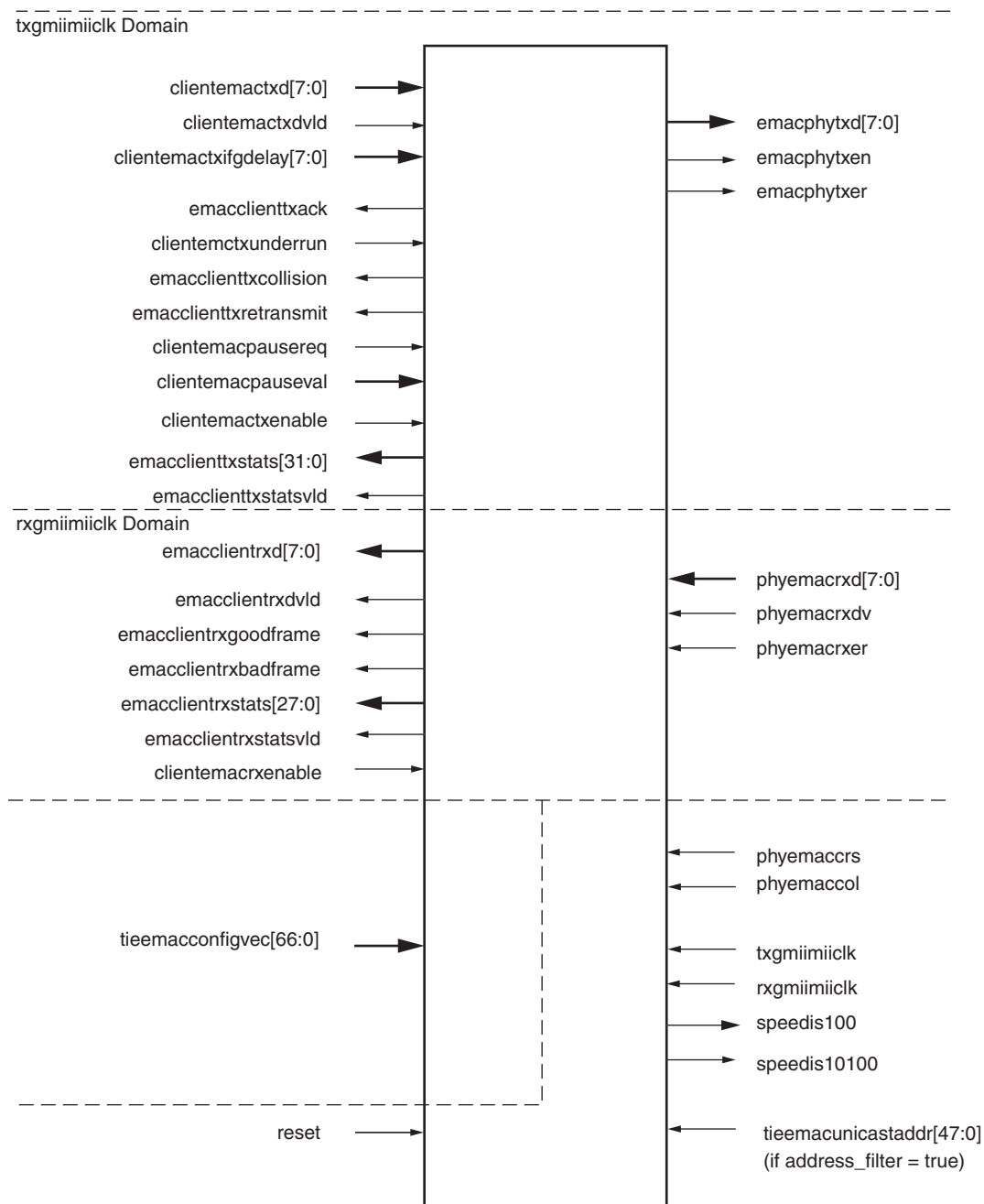


Figure 2-5: Component Pinout for MAC without Optional Management Interface (`clock_enables` = true)

Client Side Interface Signals

Table 2-1 describes the client-side transmit signals of the TEMAC core. These signals are used to transmit data from the client to the TEMAC core.

Table 2-1: Client Interface Signal Pins

Signal	Direction	Description
clientemactxd[7:0]	Input	Frame data to be transmitted is supplied on this port.
clientemactxdvld	Input	Control signal for clientemactxd port.
clientemactxifgdelay[7:0]	Input	Control signal for configurable interframe gap adjustment. See “Interframe Gap Adjustment: Full-Duplex Mode Only,” on page 54 for timing diagrams.
emacclienttxack	Output	Handshaking signal. Asserted when the current data on clientemactxd has been accepted. See “Transmitting Outbound Frames,” on page 47 for timing diagrams.
clientemactxenable	Input	If the core is built using the optional clock enable logic the number of clock resources is reduced by clocking the transmit client interface on the txgmiimiiclk input. At speeds below 1 Gbps this signal must toggle between ‘1’ and ‘0’ on alternate txgmiimiiclk cycles for correct operation of the core. At 1 Gbps it must be held high. See “Transmitting Outbound Frames,” on page 47 for timing diagrams.
clientemactxunderrun	Input	Asserted by client to force MAC core to corrupt the current frame.
emacclienttxcollision	Output	Asserted by the MAC core to signal a collision on the medium and that any transmission in progress should be aborted. Always ‘0’ when the MAC core is in full-duplex mode.
emacclienttxretransmit	Output	When asserted at the same time as the emacclienttxcollision signal, this signals to the client that the aborted frame should be resupplied to the MAC core for retransmission. Always ‘0’ when the MAC core is in full-duplex mode.
emacclienttxstats[31:0]	Output	This gives information on the last frame transmitted. See “Transmitter Statistics Vector,” on page 55 for vector contents.
emacclienttxstatsvld	Output	Asserted at end of frame transmission, indicating that the emacclienttxstats is valid.

Table 2-1: Client Interface Signal Pins (Continued)

Signal	Direction	Description
emacclientrx[d[7:0]	Output	Frame data received is supplied on this port.
emacclientrxdvld	Output	Control signal for the emacclientrx port.
emacclientrxgoodframe	Output	Asserted at end of frame reception to indicate that the frame should be processed by the MAC client. See “Normal Frame Reception,” on page 39.
emacclientrxbadframe	Output	Asserted at end of frame reception to indicate that the frame should be discarded by the MAC client. See “Frame Reception with Errors,” on page 42.
clientemacrxenable	Input	If the core is built using the optional clock enable logic the number of clock resources is reduced by clocking the receive client interface on the rxgmiimiick input. At speeds below 1 Gbps, this signal must toggle between ‘1’ and ‘0’ on alternate rxgmiimiick cycles for correct operation of the core. At 1 Gbps, it must be held high. See “Receiving Inbound Frames,” on page 39 for timing diagrams.
emacclientrxstats[27:0]	Output	Provides information about the last frame received. See “Receiver Statistics Vector,” on page 45 for the vector contents.
emacclientrxstatsvld	Output	Asserted at end of frame reception, indicating that the emacclientrxstats is valid.
clientemacpausereq	Input	Pause request: sends a pause frame down the link. See “Transmitting a Pause Control Frame,” on page 59.
clientemacpauseval[15:0]	Input	Pause value: inserted into the parameter field of the transmitted pause frame.

Management Interface Signals

Table 2-2 describes the Management Interface and support signals. These signals are used by the client to configure the MAC core and to read the status of configuration bits. See [“Using the Optional Management Interface,” on page 81.](#)

Table 2-2: Optional Management Interface Signal Pinout

Signal	Direction	Description
hostclk	Input	Clock for Management Interface.
hostopcode[1:0]	Input	Defines operation to be performed over MDIO interface. Bit 1 is also used in configuration register access. See “Using the Management Interface,” on page 88.
hostaddr[9:0]	Input	Address of register to be accessed.
hostwrdata[31:0]	Input	Data to write to register.
hostrddata[31:0]	Output	Data read from register.
hostmiimsel	Input	When asserted, the MDIO interface is accessed. When disasserted, the MAC internal configuration is accessed.
hostreq	Input	Used to signal a transaction on the MDIO interface. See “Using the Management Interface,” on page 88.
hostmiimrdy	Output	When high, the MDIO interface has completed any pending transaction and is ready for a new transaction.

Configuration Vector Signals

If the Management Interface is not present, the configuration of the core is carried out by a configuration vector. ([Table 2-3.](#))

Table 2-3: Alternative to the Optional Management Interface: Configuration Vector Signal Pinout

Signal	Direction	Description
tieemacconfigvec[66:0]	Input	The Configuration Vector is used to replace the functionality of the MAC Configuration Registers when the Management Interface is not used. See “Accessing Configuration without the Management Interface,” on page 94.

Address Filter Signals

If the optional address filter is included in the core, the user may specify a unicast address for the MAC by setting the `tieemacunicastaddr` signal. If the Management Interface is present, this can be overwritten by writing to the unicast address register. See [“Using the Management Interface,” on page 88](#).

Table 2-4: Address Filter Unicast Address

Signal	Direction	Description
<code>tieemacunicastaddr[47:0]</code>	Input	This vector is used to set the default address for the MAC. See “Address Filter,” on page 44 .

Clock, Speed Indication, and Reset Signals

[Table 2-5](#) describes the clock signals that are input to the core and the outputs that can be used to select between the three operating speeds. The clock signals are generated in the top-level example design provided with the core. See [“Clocking,” on page 117](#).

Table 2-5: Clock, Speed Indication and Reset Signals

Signal	Direction	Description
<code>txcoreclk</code>	Input	Only present when <code>clock_enables=false</code> . The clock for data transmission on the client side of the core. This is 125 MHz at 1 Gbps, 12.5 MHz at 100 Mbps and 1.25 MHz at 10 Mbps. This clock should be used to clock the client transmit circuitry. For more information see “Clocking,” on page 117 .
<code>rxcoreclk</code>	Input	Only present when <code>clock_enables=false</code> . The clock for the reception of data on the client side of the core. This is 125 MHz at 1 Gbps, 12.5 MHz at 100 Mbps and 1.25 MHz at 10 Mbps. This clock should be used to clock the client receiver circuitry. For more information, see “Clocking,” on page 117 .
<code>txgmiimiick</code>	Input	The clock for the transmission of data on the physical interface. This is 125 MHz at 1 Gbps, 25 MHz at 100 Mbps and 2.5 MHz at 10 Mbps. This clock should be used to clock the physical interface transmit circuitry. For more information, see “Clocking,” on page 117 . When <code>clock_enables=true</code> , this clock is used to clock the entire transmit side of the core.

Table 2-5: Clock, Speed Indication and Reset Signals

Signal	Direction	Description
rxgmiimiiclk	Input	The clock for the reception of data on the physical interface. This is 125 MHz at 1 Gbps, 25 MHz at 100 Mbps and 2.5 MHz at 10 Mbps. This clock should be used to clock the physical interface receive circuitry. For more information, see “Clocking,” on page 117 . When clock_enables=true, this clock is used to clock the entire receive side of the core.
speedis100	Output	This output is asserted when the core is operating at 100 Mbps. It is derived from either bits 30 and 31 of the MAC Speed Configuration register (See “Configuration Registers,” on page 81) if the optional Management Interface is present. If the Management Interface is not present, this is derived from configuration vector bits 65 and 66.
speedis10100	Output	This output is asserted when the core is operating at either 10 Mbps or 100 Mbps. It is derived from either bits 30 and 31 of the MAC Speed Configuration register (see “Configuration Registers,” on page 81) if the optional Management Interface is present. If the Management Interface is not present, this is derived from configuration vector bits 65 and 66.
reset	Input	Asynchronous reset for entire core. See “Reset Conditions,” on page 126 for more information on the reset circuit.

Physical Interface Signals

[Table 2-6](#) describes the GMII/MII signals of the MAC core. These are typically attached to a PHY module, either off-chip or internally integrated. The GMII is defined in *IEEE 802.3* clause 35. The GMII/MII physical interface, together with logic to convert these signals to RGMII format, is described in [“Using the Physical Side Interface,” on page 63](#)

Table 2-6: GMII/MII Interface Signal Pinout

Signal	Direction	Description
emacphytxd[7:0]	Output	Transmit data to PHY.
emacphytxen	Output	Data Enable control signal to PHY.
emacphytxer	Output	Error control signal to PHY.
phyemaccrs	Input	Control signal from PHY.

Table 2-6: GMII/MII Interface Signal Pinout (Continued)

Signal	Direction	Description
phyemaccol	Input	Control signal from PHY.
phyemacrx[7:0]	Input	Received data from PHY.
phyemacrxdv	Input	Data Valid control signal from PHY.
phyemacrxe	Input	Error control signal from PHY.
corehassgmii	Input	Tie this input high if the core is interfaced to the Ethernet 1000BASE-X PCS/PMA or SGMII core in SGMII mode. See “Integrating with the Ethernet 1000BASE-X PCS/PMA or SGMII Core,” on page 127 for more information.

Optional MDIO Signals

Table 2-7 describes the MDIO (MII Management) interface signals of the MAC core (see [“Using the MDIO Interface,”](#) on page 77). These signals are typically connected to the MDIO port of a PHY device, either off-chip or an SoC-integrated core. The MDIO format is defined in *IEEE 802.3* clause 22.

Table 2-7: MDIO Interface Signal Pinout

Signal	Direction	Description
emacphy_mclkout	Output	MDIO Management Clock: derived from hostclk on the basis of supplied configuration data when the optional Management Interface is used. See “Accessing MDIO via the TEMAC,” on page 92.
emacphy_mdin	Input	Input data signal for communication with PHY configuration and status. Tie high if unused.
emacphy_mdout	Output	Output data signal for communication with PHY configuration and status.
emacphy_mdtri	Output	Tristate control for MDIO signals; ‘0’ signals that the value on emacphy_mdout should be asserted onto the MDIO bus.

Generating the Core

This chapter provides information about configuring and generating the core using the CORE Generator™ tool.

GUI Interface

Figure 3-1 displays the TEMAC core customization screen.

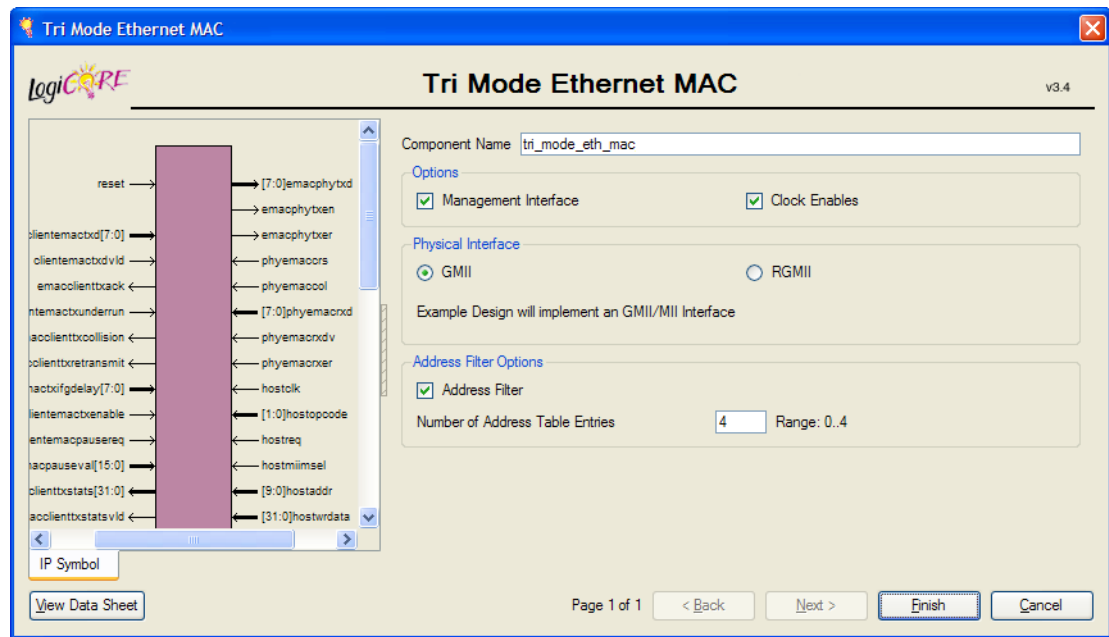


Figure 3-1: Core Customization Screen

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 “_”.

Management Interface

Select this option if you wish to include the optional Management Interface (see “Using the Optional Management Interface,” on page 81). If this option is not selected, the core will be generated with a configuration vector. The default is to have the Management Interface.

Clock Enables

Select this option to run the transmit and receive sections of the core on the clocks from the physical interface. This reduces the number of BUFGMUXes that are used by the core by 2.

Address Filter

It is possible to generate the core with an address filter. This will prevent the reception of frames that are not addressed to this MAC. See [“Address Filter,” on page 44](#).

Number of Address Table Entries

The Address Filter can be instantiated with an address table that holds up to 4 additional valid addresses. The user may select an integer between 0 and 4 to define the number of addresses that are present in the table.

Physical Interface

It is possible to select from two different physical interface choices for the core:

- GMII/MII - see [“Implementing External GMII,” on page 63](#).
- RGMII - see [“Implementing External RGMII,” on page 70](#).

The choice of physical interface will determine the content of the example design delivered with the core: the external GMII or RGMII will be added in the HDL top-level design file. There is no change in the core’s netlist for this option.

The default is the GMII physical interface.

Parameter Values in the XCO File

XCO file parameter names and their values are identical to the names and values shown in the GUI, except that underscore characters (_) are used instead of spaces. The text in an XCO file is case insensitive.

[Table 3-1](#) shows the XCO file parameters and values, and summarizes the GUI defaults. The following is an example of the CSET parameters in an XCO file:

```
CSET component_name = abc123
CSET address_filter = true
CSET management_interface = true
CSET clock_enables = false
CSET physical_interface = gmii
CSET number_of_address_table_entries = 4
```

Table 3-1: XCO File Values and Default Values

Parameter	XCO File Values	Default GUI Setting
component_name	ASCII text starting with a letter and based upon the following character set: a..z, 0..9 and _	blank
address_filter	One of the following keywords: true, false	true

Table 3-1: XCO File Values and Default Values

Parameter	XCO File Values	Default GUI Setting
number_of_address_table_entries	Integer in the range 0 - 4	4
clock_enables	One of the following keywords: true, false	true
management_interface	One of the following keywords: true, false	true
physical_interface	One of the following keywords: gmii, rgmii	gmii

Output Generation

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

- the netlist file
- supporting CORE Generator files
- release notes and other documentation
- subdirectories containing example design files
- scripts to run the core through the back-end tools and to simulate the core using the Mentor Graphics® ModelSim® and the Cadence® IUS simulators

See “[CORE Generator Directory Structure](#),” on page 121 for definitions of all output files.

Designing with the Core

This chapter provides general guidelines for creating designs using the TEMAC core, including a detailed description of each interface to the core. For information about special design considerations, for example, clocking schemes, see [“Clocking and Reset,” on page 117](#). To work with the FIFO provided in the example design included with the TEMAC core, see [Appendix A, “Using the Client Side FIFO.”](#) For more information about the example design see the *Tri-Mode Ethernet MAC Getting Started Guide*.

General Design Guidelines

This section describes the steps required to turn a TEMAC core into a fully-functioning design integrated with user application logic. Its important to recognize that not all designs will require all the design steps listed in this chapter. The following discusses the design steps required for various implementations. Follow the logic design guidelines in this manual carefully.

Design Steps

Generate the core from the CORE Generator (see [Chapter 3, “Generating the Core”](#)).

Using the HDL Example Design as User Top-level

See the *Tri-Mode Ethernet MAC Getting Started Guide*.

- Edit the HDL example design file produced by the CORE Generator to add user logic and any other I/Os required. Add/change clocking scheme.
- Synthesize the entire design. For a VHDL design, the Xilinx Synthesis Tool (XST) script and project file in the `/implement` directory may be adapted to include the user's HDL files. For a verilog design, the XST script file and the `implement` script in the `/implement` directory may be adapted.
- Run the `implement` script in the `/implement` directory to create a top-level netlist, which includes the TEMAC core netlist. The script may also run the Xilinx tools `map`, `par`, and `bitgen`, creating a bitstream that can be downloaded to a Xilinx device.
- Simulate the entire design using the demonstration test bench provided in the `/simulation` directory.
- Download the bitstream to a Virtex-5™, Virtex-4, Virtex-II, Virtex-II Pro, Spartan™-3, Spartan-3E or Spartan-3A device.

Using the HDL Example Design in a User Design

Generate the core from the CORE Generator (see [Chapter 3, “Generating the Core”](#)).

- Edit the HDL example design file produced by the CORE Generator to remove unnecessary IOBs, pipeline registers, Digital Clock Managers (DCMs), and anything else not required by the user. These may need to be replicated within the user top-level design.
- Add an interface to the HDL example design so that it may be instantiated in the user design.
- Synthesize the entire design, including the same files used for default implementation.
- Run the Xilinx tools **map**, **par**, and **bitgen** to create a bitstream that can be downloaded to a Xilinx device. Care must be taken to constrain the design correctly, and the UCF produced by CORE Generator should be used as the basis for the user's UCF. See [Chapter 9, "Constraining the Core"](#) for more information.
- Simulate the entire design using the demonstration test bench provided in the `/test` directory.
- Download the bitstream to a Virtex-5™, Virtex-4, Virtex-II, Virtex-II Pro, Spartan™-3, Spartan-3E or Spartan-3A device.

Understand Signal Pipelining

Pipeline registers are used in the HDL example design provided with the core *only* to allow the core interfaces to be interfaced cleanly to the IOBs on the selected device; these registers create artificial latency on some inputs and outputs in the example design file. Because a user design will most likely connect to the core interfaces on the same FPGA fabric, the pipeline registers will probably not be required in a user design and can be safely removed if the user plans to add interface registers to their own logic.

Register All I/Os

To simplify timing and increase system performance in an FPGA design, register all I/Os. All inputs and outputs from the user application should come from, or connect to, a flip-flop inside the user application. It may not be possible to register the signal on all paths; however, doing so simplifies timing analysis and makes it easier for the Xilinx tools to place and route the design.

Recognize Timing Critical Signals

The UCF provided with the core identifies the timing critical signals and the timing constraints that should be applied.

Use Supported Design Flows

The XST/ISE 9.1i/Mentor ModelSim or Cadence IUS design flow is supported for the TEMAC core.

Make Only Allowed Modifications

The TEMAC core should not be modified by the user, as they may cause adverse effects on system timing and protocol compliance. Supported user configurations of the TEMAC core can only be made by the selecting options from the CORE Generator screen when the core is generated. For more information, see [Chapter 3, "Generating the Core."](#)

Using the Client Side Data Path

This chapter provides a detailed description of the client-side data-flow interface. The definitions and abbreviations used in this chapter are described in [Table 5-1](#).

Table 5-1: Abbreviations Used in Timing Diagrams

Abbreviation	Definition
DA	Destination address; 6 bytes
SA	Source address; 6 bytes
L/T	Length/type field; 2 bytes
FCS	Frame check sequence; 4 bytes

Receiving Inbound Frames

The client interface is designed for maximum flexibility in matching to a client switching fabric or network processor interface.

The data pathway is 8 bits wide in both the transmit and receive directions. If the core is generated with the `clock_enable` option set to false, each pathway is synchronous to `txcoreclk` and `rxcoreclk` respectively. If the core is generated with the `clock_enable` option set to true, each pathway is synchronous to `txgmiiimiclk` and `rxgmiiimiclk`. This gives completely independent full-duplex operation.

Normal Frame Reception

Without Clock Enables

[Figure 5-1](#) shows the timing of a normal inbound frame transfer when the core is generated without the optional clock enables. 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 `emacclientrxgoodframe` signal to indicate that the frame was successfully received and that the frame should be analyzed by the client.

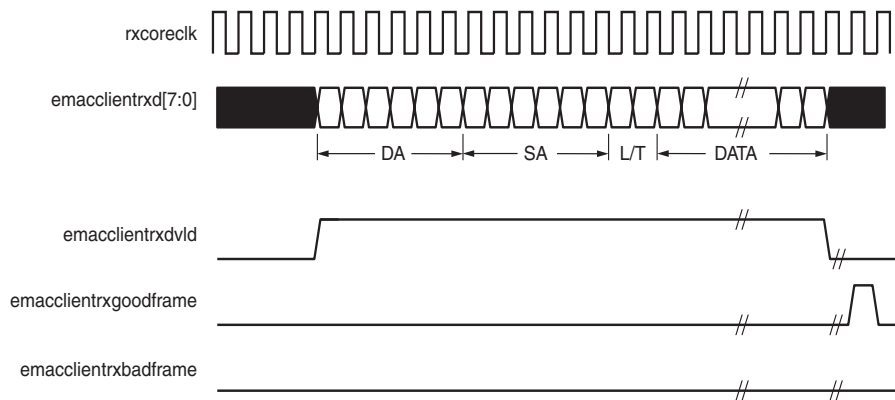


Figure 5-1: Normal Frame Reception

Frame parameters (destination address, source address, length/type and optionally FCS) are supplied on the data bus according to the timing diagram. The abbreviations are described in [Table 5-1](#).

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, then this padding will not be passed to the client in the data payload. The exception to this is in the case where FCS passing is enabled. See ["Client-Supplied FCS Passing"](#) on page 42.

Therefore, when client-supplied FCS passing is disabled, `emacclientrxdvld = '0'` between frames for the duration of the padding field (if present), the FCS field, carrier extension (if present), the interframe gap following the frame, and the preamble field of the next frame. When client-supplied FCS passing is enabled, `emacclientrxdvld = '0'` between frames for the duration of carrier extension (if present), the interframe gap, and the preamble field of the following frame.

Using Clock Enables

Figure 5-2 and Figure 5-3 show the timing of the reception of frames when the core is generated with the optional clock enable circuitry. Here the signals are synchronous to the rxgmiimiiclk input. At 1 Gbps, the client should hold the clientemacrxdenable line high (Figure 5-2). At slower speeds, the clientemacrxdenable line should be toggled on the rising edge of rxgmiimiiclk. This signal should be used to enable the client receiver logic. As 4 bits of data are transferred across the MII interface on each rising edge of rxgmiimiiclk at 10/100 Mbps, this gives 8 bits of valid data every second rxgmiimiiclk period.

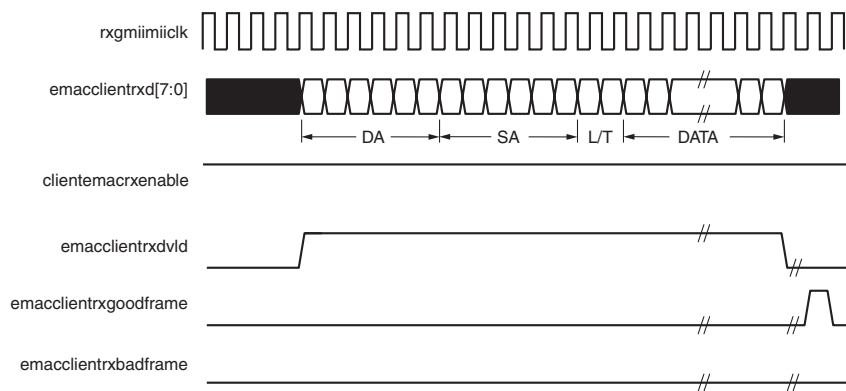


Figure 5-2: Normal Frame Reception at 1 Gbps with Optional Clock Enables

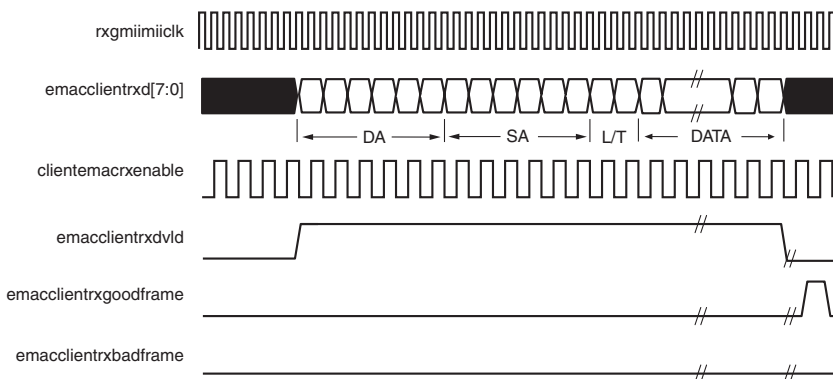


Figure 5-3: Normal Frame Reception at 10/100 Mbps with Optional Clock Enables

In the remainder of this section, the timing diagrams are shown for the non-clock enabled version of the core. The timing of the clock enabled core is identical to the given diagrams at 1 Gbps with rxgmiimiiclk replacing rxcoreclk and clientemacrxdenable held high. At 10/100 Mbps, the frequency of rxgmiimiiclk is double that of the illustrated rxcoreclk and the clientemacrxdenable signal is toggled on every rising edge of rxgmiimiiclk to provide the necessary data rate.

emacclientrxgoodframe and emacclientrxbadframe Timing

Although Figure 5-1 illustrates the emacclientrxgoodframe signal asserted shortly after the last valid data on emacclientrx, this is not always the case. The emacclientrxgoodframe or emacclientrxbadframe signals are asserted only after all frame checks are completed. This is after the FCS field has been received (and after reception of carrier extension, if present).

Therefore, either emacclientrxgoodframe or emacclientrxbadframe is asserted following frame reception at the beginning of the interframe gap.

Frame Reception with Errors

Figure 5-4 illustrates an unsuccessful frame reception (for example, a fragment frame or a frame with an incorrect FCS). In this case, the emacclientrxbadframe 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.

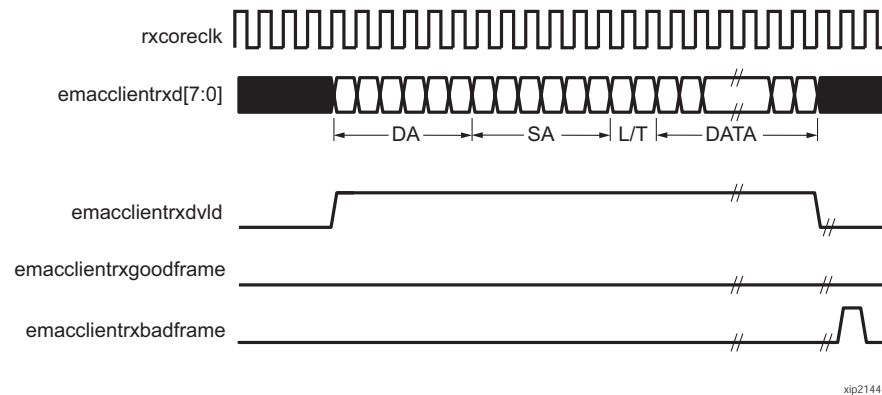


Figure 5-4: Frame Reception with Error

Client-Supplied FCS Passing

If the MAC core is configured to pass the FCS field to the client (see “Configuration Registers,” on page 81), it is handled as displayed in Figure 5-5.

In this case, any padding inserted into the frame to meet Ethernet minimum frame length specifications will be left intact and passed to the client.

Even though the FCS is passed up to the client, it is also verified by the MAC core, and `emacclientrxbadframe` asserted if the FCS check fails.

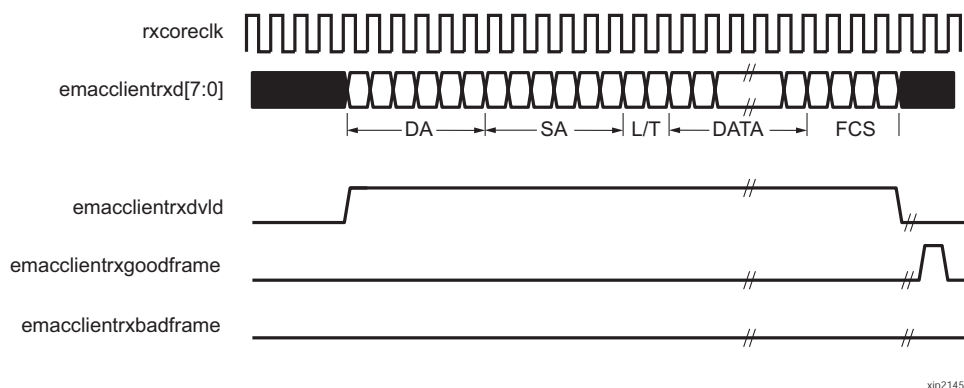


Figure 5-5: Frame Reception with In-Band FCS Field

VLAN Tagged Frames

The reception of a VLAN tagged frame (if enabled, see “[Configuration Registers](#),” on page 81) can be seen in Figure 5-6. The VLAN frame is passed to the client so that the frame may 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 may be found in *IEEE 802.3-2002* standard.

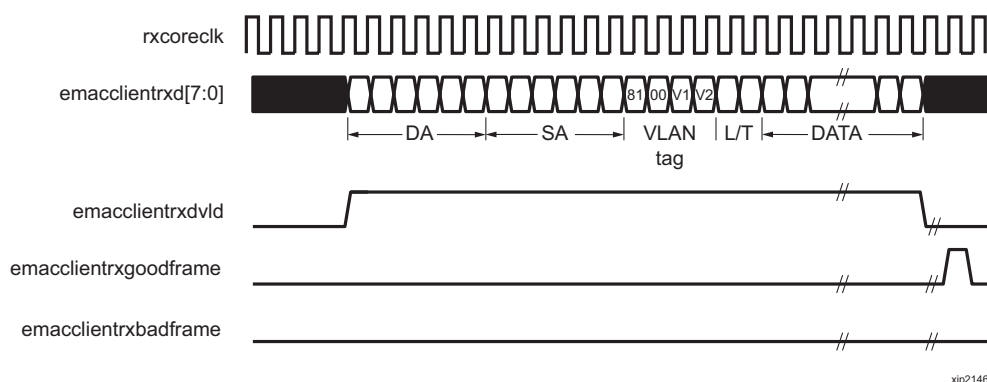


Figure 5-6: Reception of a VLAN Tagged Frame

Maximum Permitted Frame Length

The maximum legal length of a frame specified in *IEEE 802.3-2002* 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, `emacclientrxbadframe` 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](#),” on page 81.

Length/Type Field Error Checks

Enabled

Default operation is with the length/type error checking enabled. In this mode, the following checks are made on all frames received. If either of these checks fail, the frame is marked as BAD.

A value in the length/type field that is greater than or equal to decimal 46 but less than decimal 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 is checked to see that the data field is padded to exactly 46 bytes (so that the resultant frame is minimum frame size: 64 bytes total in length).

Furthermore, if padding is indicated (the length/type field is less than decimal 46) and “Client-Supplied FCS Passing” is disabled, then the length value in the length/type field will be used to deassert `emacclientrxdvld` after the indicated number of data bytes so that the padding bytes are removed from the frame.

Disabled

When the length/type error checking is disabled (see “Register Maps” on page 82) and the length/type field has a length interpretation, the MAC does not check the length value against the actual data length received. A frame containing only this error is marked as good. However, if the length/type field is less than decimal 46, the MAC will mark a frame as bad if it is not the minimum frame size of 64 bytes.

Furthermore, 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 `emacclientrxdvld`. Instead `emacclientrxdvld` will be deasserted before the start of the FCS field; in this way any padding will not be removed from the frame.

Address Filter

If the optional address filter is included in the core, the MAC is able to reject frames that do not contain a known address in their destination address field. If a frame is rejected, the `emacclientrxdvld` signal is not asserted for the duration of the frame. In addition neither `emacclientrxgoodframe` or `emacclientrxbadframe` are asserted at the end of the frame. The statistics vectors are still output with a valid pulse at the end of the rejected frame.

If the address filter is not in promiscuous mode, it will reject frames in which the destination address does not meet any of the following criteria:

- It is equal to the broadcast address defined in the *IEEE 802.3-2002* specification.
- It is equal to the pause multicast address defined in the *IEEE 802.3-2002* specification.
- The destination address field contains the Pause frame MAC source address specified in Receiver Configuration Word 0 and Word 1.
- It is equal to the MAC Unicast Address. When the optional Management Interface is present this is contained in the unicast address configuration registers (Table 8-9 and Table 8-10). If the Management Interface is not present the unicast address is input on the `tieemacunicastaddr` input.
- It matches any of the addresses stored in the MAC address table. The address table is only present when the MAC contains the optional Management Interface.

Receiver Statistics Vector

The statistics for the frame received are contained within the `emacclientrxstats` output. The vector is driven synchronously by the receiver clock, `rxcoreclk` (`rxgmiiimiclk` if `clock_enables = true`) following frame reception. Table 5-2 defines the bit field for the vector.

All bit fields, with the exception of `BYTE_VALID` are valid only when the `emacclientrxstatsvld` is asserted, as illustrated in Figure 5-7. `BYTE_VALID` is significant on every receiver clock cycle.

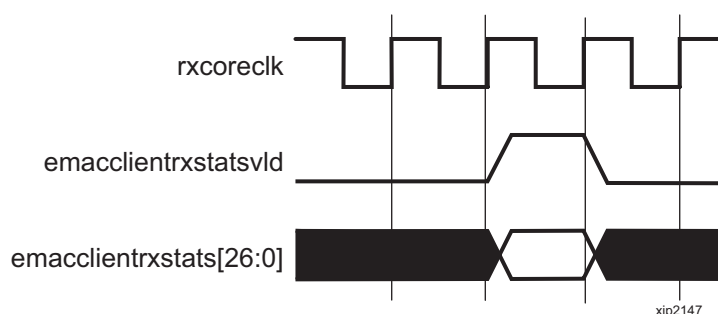


Figure 5-7: Receiver Statistics Vector Timing

Table 5-2: Bit Definition for the Receiver Statistics Vector

emacclientrx stats	Name	Description
27	ADDRESS_MATCH	If the optional address filter is included in the core, this bit is asserted if the address of the incoming frame matches one of the stored or pre-set addresses in the address filter. If the address filter is omitted from the core, or is configured in promiscuous mode, this line is held high.
26	ALIGNMENT_ERROR	Asserted at speeds below 1 Gbps if the frame contains an odd number of nibbles and the FCS for the frame is invalid.
25	LENGTH/TYPE Out of Range	If the length/type field contained a length value that did not match the number of MAC client data bytes received and the length/type field checks are enabled, then this bit is asserted. This bit is also asserted if the length/type field is less than 46, and the frame is not padded to exactly 64 bytes. This is independent of whether or not the length/type field checks are enabled.

Table 5-2: Bit Definition for the Receiver Statistics Vector (*Continued*)

emacclientrx stats	Name	Description
24	BAD_OPCODE	Asserted if the previous frame was error-free and 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).
23	FLOW_CONTROL_FRAME	Asserted if the previous frame was error-free, contained the special control frame identifier 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 supported PAUSE opcode, and was acted upon by the MAC.
22	BYTE_VALID	Asserted if a MAC frame byte (destination address to FCS inclusive) is in the process of being received. This is valid on every clock cycle. Do not use this as an enable signal to indicate that data is present on emacclientrxd[7:0].
21	VLAN_FRAME	Asserted if the previous frame contained a VLAN identifier in the length/type field when receiver VLAN operation is enabled.
20	OUT_OF_BOUNDS	Asserted if the previous frame exceeded the specified IEEE 802.3-2002 maximum legal length (see "Maximum Permitted Frame Length" on page 43). This is only valid if jumbo frames are disabled.
19	CONTROL_FRAME	Asserted if the previous frame contained the special control frame identifier in the length/type field.
18 down to 5	FRAME_LENGTH_COUNT	The length of the previous frame in number of bytes. The count will stick at 16368 for any jumbo frames larger than this value.
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.

Table 5-2: Bit Definition for the Receiver Statistics Vector (*Continued*)

emacclientrx stats	Name	Description
2	FCS_ERROR	Asserted if the previous frame received was correctly aligned but had an incorrect FCS value or the MAC detected error codes during frame reception.
1	BAD_FRAME ¹	Asserted if the previous frame received contained errors.
0	GOOD_FRAME ¹	Asserted if the previous frame received was error-free.

1. If the length/type field error checks are disabled, a frame which has an actual data length that does not match the length/type field value will be marked as a GOOD_FRAME providing no additional errors were detected. See ["Length/Type Field Error Checks" on page 44](#) for more information.

Transmitting Outbound Frames

Normal Frame Transmission

Without Clock Enables

The timing of a normal outbound frame transfer can be seen in [Figure 5-8](#). When the client wants to transmit a frame, it places the first column of data onto the `clientemactxd` port and asserts a '1' onto `clientemactxdvld`.

When the MAC core has read this first byte of data, it will assert the `emacclienttxack` signal; on the next and subsequent rising 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 taking `clientemactxdvld` low.

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 upon, rather than on separate ports.

The transmitter cannot guarantee that the minimum interframe gap will be output in half-duplex mode when clock enables are not selected. The gap may be larger than the specified

minimum of 12 bytes. If the 12-byte minimum is required, the clock enable option should be selected. This only applies to half-duplex mode.

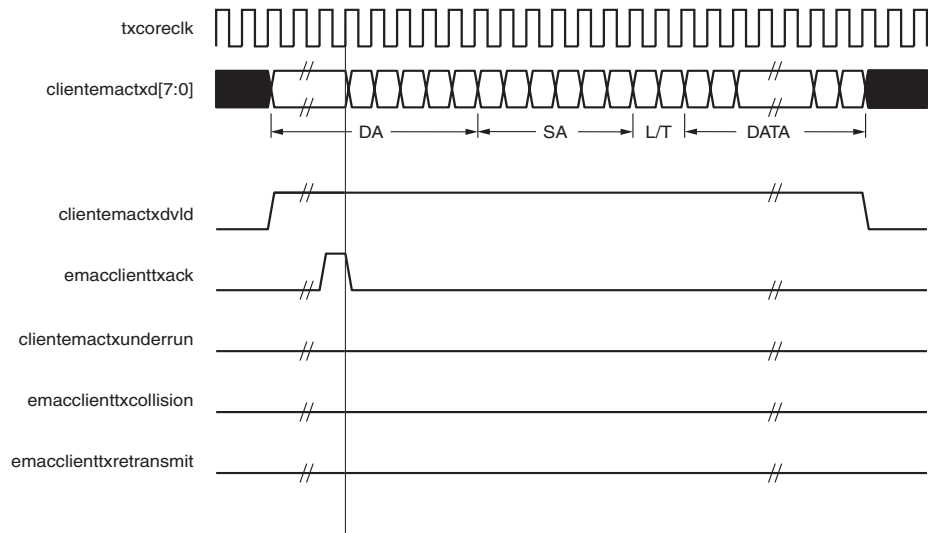


Figure 5-8: Normal Frame Transmission Across Client Interface

Using Clock Enables

If the core is generated with the optional clock enable circuitry, the client drives the **clientemactxenable** line high at 1000 Mbps and toggles it on every rising edge of **txgmiimiiclk** at slower speeds. Figure 5-9 and Figure 5-10 show normal frame transmission in this mode. The **clientemactxenable** line should be used to enable the client transmission circuitry.

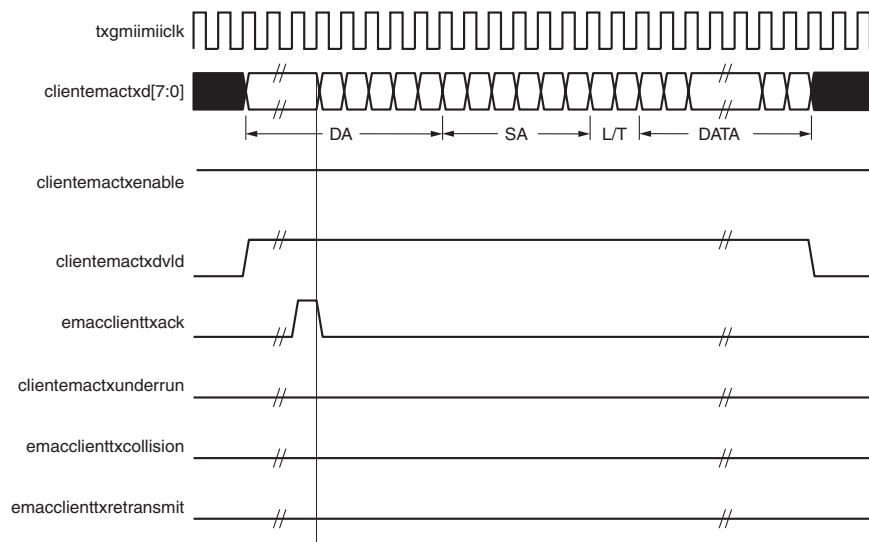


Figure 5-9: Normal Frame Transmission at 1000 Mbps with Optional Clock Enables

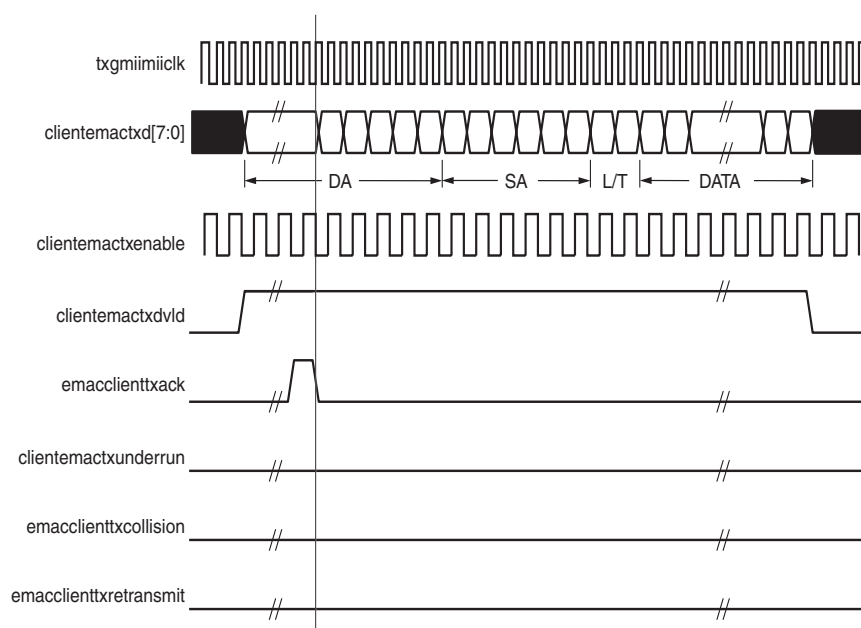


Figure 5-10: Normal Frame Transmission at 10/100 Mbps with Optional Clock Enables

In the remainder of this section, the timing diagrams are shown for the non-clock enabled version of the core. The timing of the clock enabled core is identical to the given diagrams at 1000 Mbps with `txgmiimiiclk` replacing `txcoreclk` and `clientemactxenable` held high. At 10/100 Mbps, the frequency of `txgmiimiiclk` is double that of the illustrated `txcoreclk` and the `clientemactxenable` signal is toggled on every rising edge of `txgmiimiiclk` to provide the necessary data rate.

Padding

When fewer than 46 bytes of data are supplied by the client to the MAC core, the transmitter module will add padding up to the minimum frame length. The exception to this is when the MAC core is configured for client-passed FCS; in this case the client must also supply the padding to maintain the minimum frame length.

Client-Supplied FCS Passing

If the MAC core is configured to have the FCS field passed in by the client, the transmission timing is as depicted in Figure 5-11. 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.

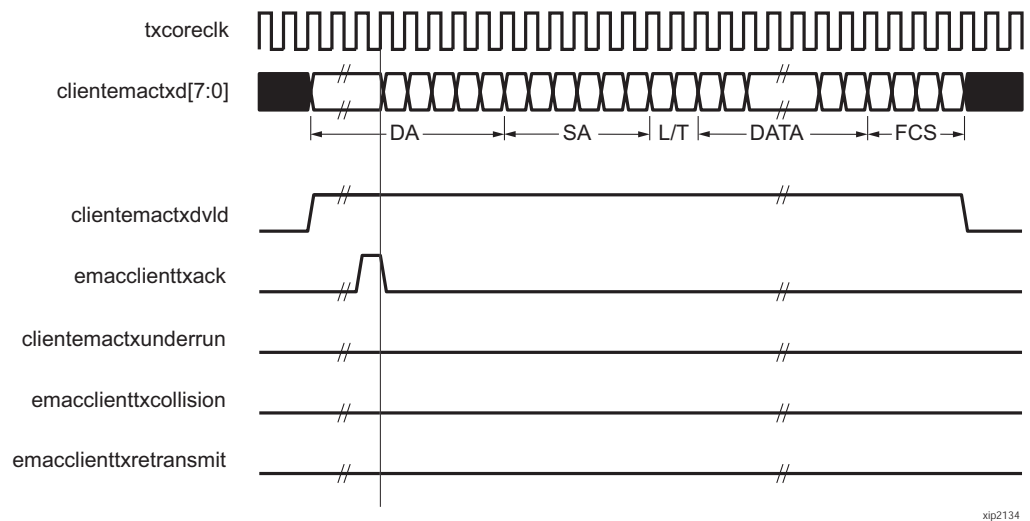


Figure 5-11: Frame Transmission with Client-supplied FCS

Client Underrun

Figure 5-12 shows the timing of an aborted transfer. This can occur, for example, if a FIFO connected to the client interface empties before a frame is completed. When the client asserts `clientemactxunderrun` during a frame transmission, the MAC core inserts an error code to corrupt the current frame and then falls back to idle transmission. It is the responsibility of the client to re-queue the aborted frame for transmission. To error the frame, the `clientemactxunderrun` signal may be asserted during the data transmission or up to 1 valid clock cycle after `clientemactxdvld` goes low.

When an underrun occurs, `clientemactxdvld` may be asserted on the clock cycle after the `clientemactxunderrun` assertion to request a new transmission.

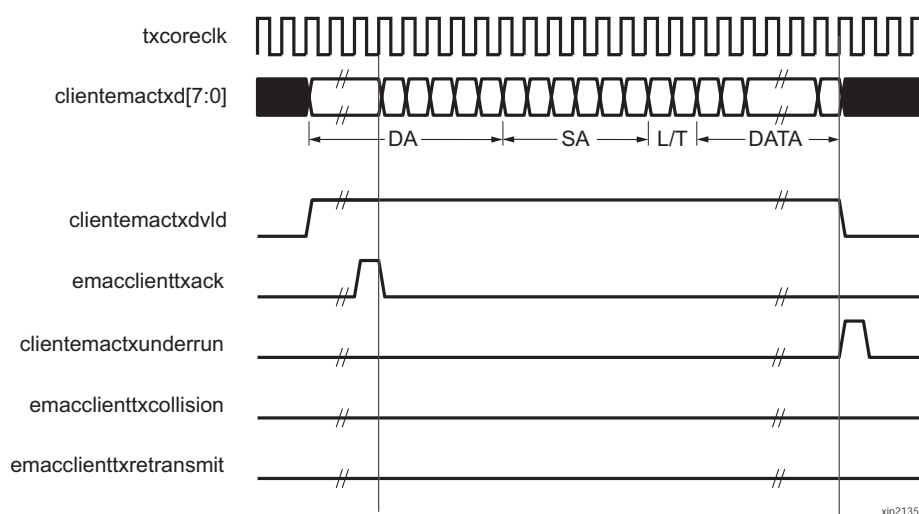


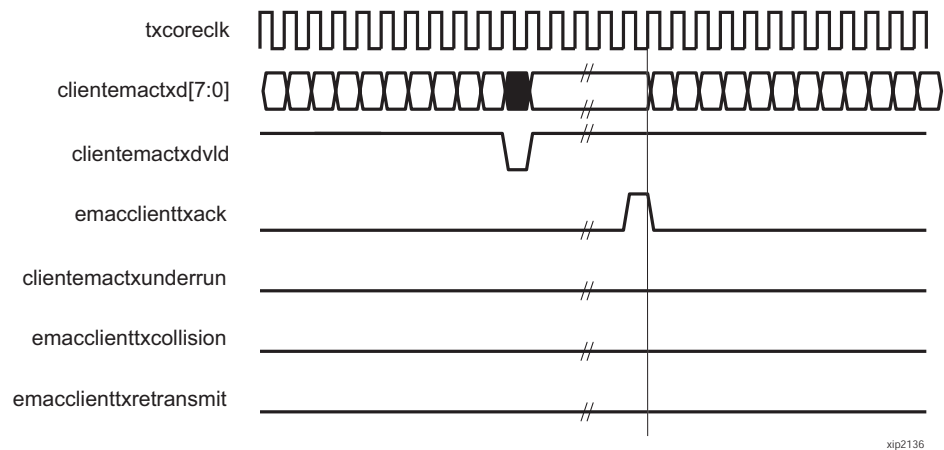
Figure 5-12: Frame Transmission with Underrun

Back-to-Back Transfers

Figure 5-13 shows the MAC client immediately ready to transmit a second frame of data following completion of its first frame. In this figure, the end of the first frame is shown on the left. On the clock cycle immediately following the final byte of the first frame, `clientemactxdvld` is taken low by the client, and is taken high one clock cycle later to indicate that the first byte of the destination address of the second frame is on `clientemactxd` awaiting transmission.

When the MAC core is ready to accept data, `emacclienttxack` is asserted and the transmission continues in the same manner as in the case of the single frame. The MAC core will defer the assertion of `emacclienttxack` appropriately to comply with inter-packet gap requirements and flow control requests.

If the MAC core is operating at 1 Gbps in half-duplex mode, the timing shown in Figure 5-13 is required to take advantage of frame bursting; the MAC core is only guaranteed to retain control of the medium if the `clientemactxdvld` signal is low for a single clock cycle. For more information on frame bursting, see *IEEE 802.3-2002*.

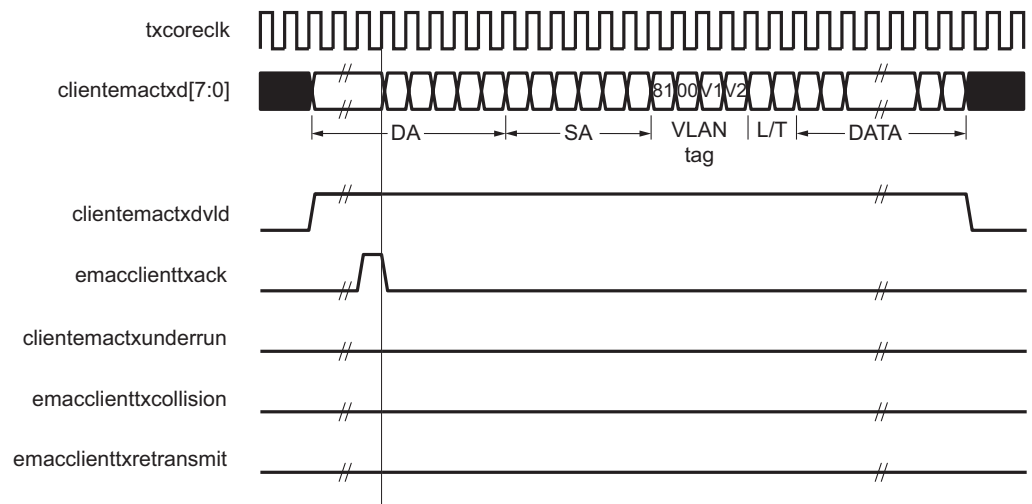


xip2136

Figure 5-13: Back-to-Back Frame Transmission

VLAN Tagged Frames

Transmission of a VLAN tagged frame (if enabled) can be seen in Figure 5-14. 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. More information on the contents of these two bytes can be found in *IEEE 802.3-2002*.



xip2137

Figure 5-14: Transmission of a VLAN Tagged Frame

Maximum Permitted Frame Length

The maximum legal length of a frame specified in *IEEE 802.3-2002* 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,”](#) on page 81.

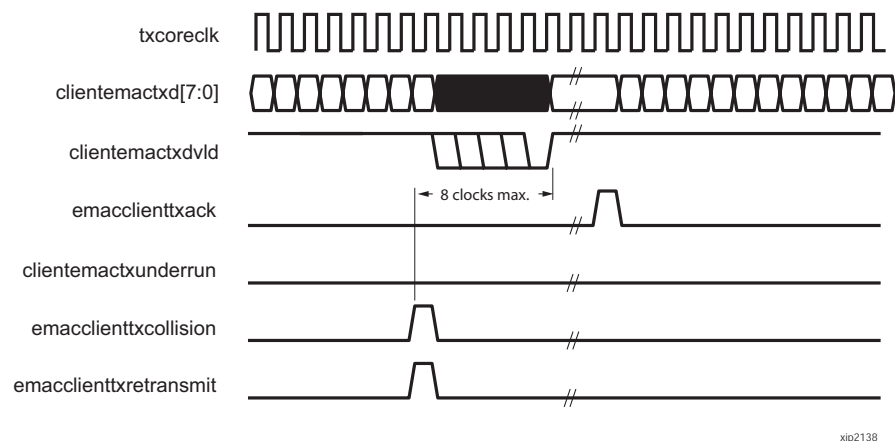
Frame Collisions: Half-Duplex Operation Only

In half-duplex Ethernet operation, collisions occur on the medium as a matter of course; this is how the arbitration algorithm is fulfilled. In the case of a collision, the MAC core signals to the client that data may need to be resupplied as follows.

If there is a collision, the `emacclienttxcollision` signal will be set to ‘1’ by the MAC core. If a frame is in progress, the client must abort the transfer and take `clientemactxdvld` to ‘0.’

If the `emacclienttxretransmit` signal is ‘1’ in the same clock cycle that the `emacclienttxcollision` signal is ‘1,’ the client must resubmit the previous frame to the MAC core for retransmission; `clientemactxdvld` must be asserted to the MAC core within 8 clock cycles of the `emacclienttxretransmit` signal in order to meet Ethernet timing requirements. See [Figure 5-15](#).

If the `emacclienttxretransmit` signal is ‘0’ in the same clock cycle that the `emacclienttxcollision` signal is ‘1,’ the number of retries for this frame has exceeded the Ethernet specification or the collision has been classed as late, and the frame should be dropped by the client. The client can then make any new frame available to the MAC for transmission without timing restriction. See [Figure 5-16](#).



xip2138

Figure 5-15: Collision Handling: Frame Retransmission Required

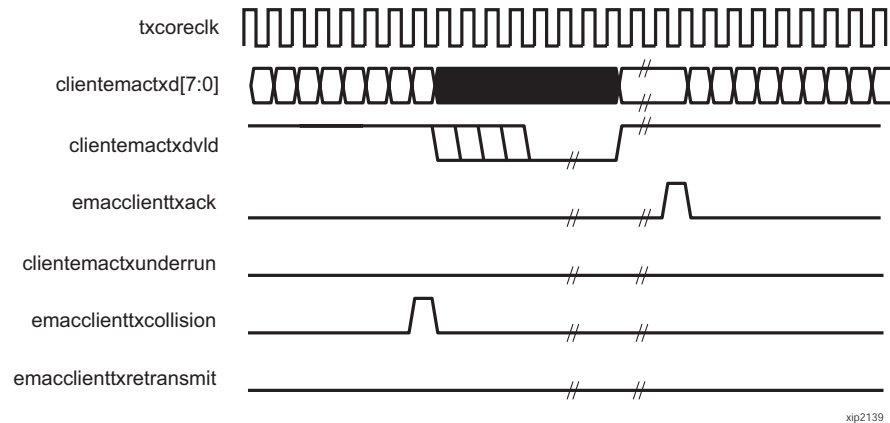


Figure 5-16: Collision Handling: No Frame Retransmission Required

Interframe Gap Adjustment: Full-Duplex Mode Only

A configuration bit in the transmitter control register (see “Configuration Registers,” on page 81) allows the user to control the length of the interframe gap transmitted by the MAC on the physical interface. If this function is selected, the MAC exerts back pressure on the client interface to delay the transmission of the next frame until the requested number of idle cycles has elapsed. The number of idle cycles is controlled by the value on the `clientemactxifgdelay` port seen at the start of frame transmission on the client interface. Figure 5-17 shows the MAC operating in this mode.

The transmitter will never separate frames by less than the minimum interframe gap specified in the IEEE 802.3-2002. This corresponds to 12 transmit clock cycles on the GMMI/MII interface. The value on the `clientemactxifgdelay` port must be larger than 12 to have an effect.

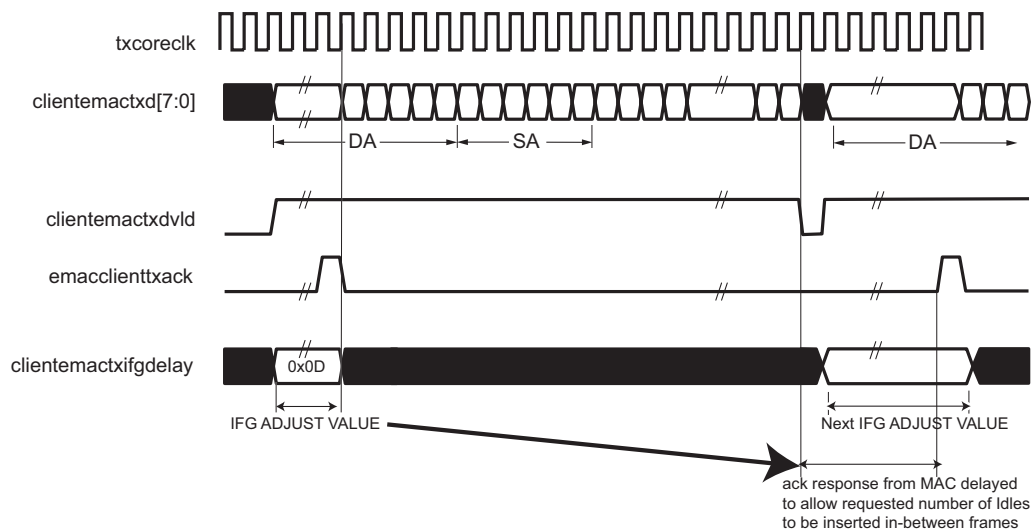


Figure 5-17: Interframe Gap Adjustment

Transmitter Statistics Vector

The statistics for the frame transmitted are contained within the `emacclienttxstats` output. The vector is driven synchronously by the transmitter clock following frame transmission. The bit field definition for the Vector is defined in [Table 5-3](#).

All bit fields, with the exception of `BYTE_VALID` are valid only when the `emacclienttxstatsvld` is asserted, as illustrated in [Figure 5-18](#). `BYTE_VALID` is significant on every transmitter clock cycle.

`emacclienttxstats` bits 28 down to 20 inclusive are for half-duplex only and will be set to logic 0 when operating in full-duplex mode.

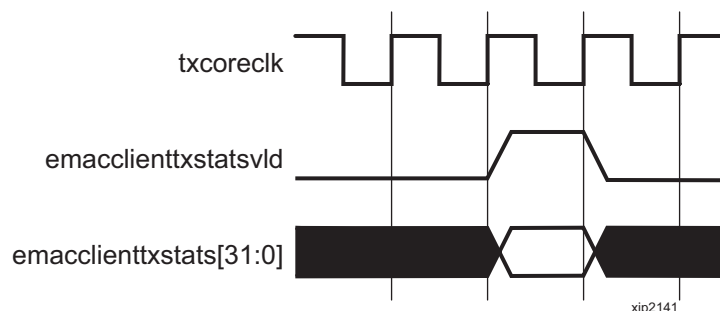


Figure 5-18: Transmitter Statistics Vector Timing

Table 5-3: Bit Definition for the Transmitter Statistics Vector

emacclienttxstats	Name	Description
31	PAUSE_FRAME_TRANSMITTED	Asserted if the previous frame was a pause frame that the MAC itself initiated in response to a <code>clientemacpausereq</code> assertion.
30	BYTE_VALID	Asserted if a MAC frame byte (DA to FCS inclusive) is in the process of being transmitted. This is valid on every clock cycle. Do not use this as an enable signal to indicate that data is present on <code>emacphytxd[7:0]</code> .
29	Reserved	Returns logic 0.
28 down to 25	TX_ATTEMPTS[3:0]	The number of attempts that have been made to transmit the previous frame. This is a 4-bit number: 0 should be interpreted as 1 attempt; 1 as 2 attempts, up until 15 as 16 attempts.

Table 5-3: Bit Definition for the Transmitter Statistics Vector (*Continued*)

emacclientxstats	Name	Description
24	Reserved	Returns logic 0.
23	EXCESSIVE_COLLISION	Asserted if a collision has been detected on each of the last 16 attempts to transmit the previous frame.
22	LATE_COLLISION	Asserted if a late collision occurred during frame transmission.
21	EXCESSIVE_DEFERRAL	Asserted if the previous frame was deferred for an excessive amount of time as defined by the constant "maxDeferTime" in IEEE 802.3-2002.
20	TX_DEFERRED	Asserted if transmission of the frame was deferred.
19	VLAN_FRAME	Asserted if the previous frame contained a VLAN identifier in the length/type field when transmitter VLAN operation is enabled.
18 down to 5	FRAME_LENGTH_COUNT	The length of the previous frame in number of bytes. The count will stick at 16368 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 contained 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 a broadcast address in the destination address field.
0	SUCCESSFUL_FRAME	Asserted if the previous frame was transmitted without error.

Using Flow Control

This chapter describes the operation of the flow control logic of the TEMAC core. The flow control block is designed to clause 31 of the *IEEE 802.3-2002* 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"](#) on page 81.)

Overview of Flow Control

Flow Control Requirement

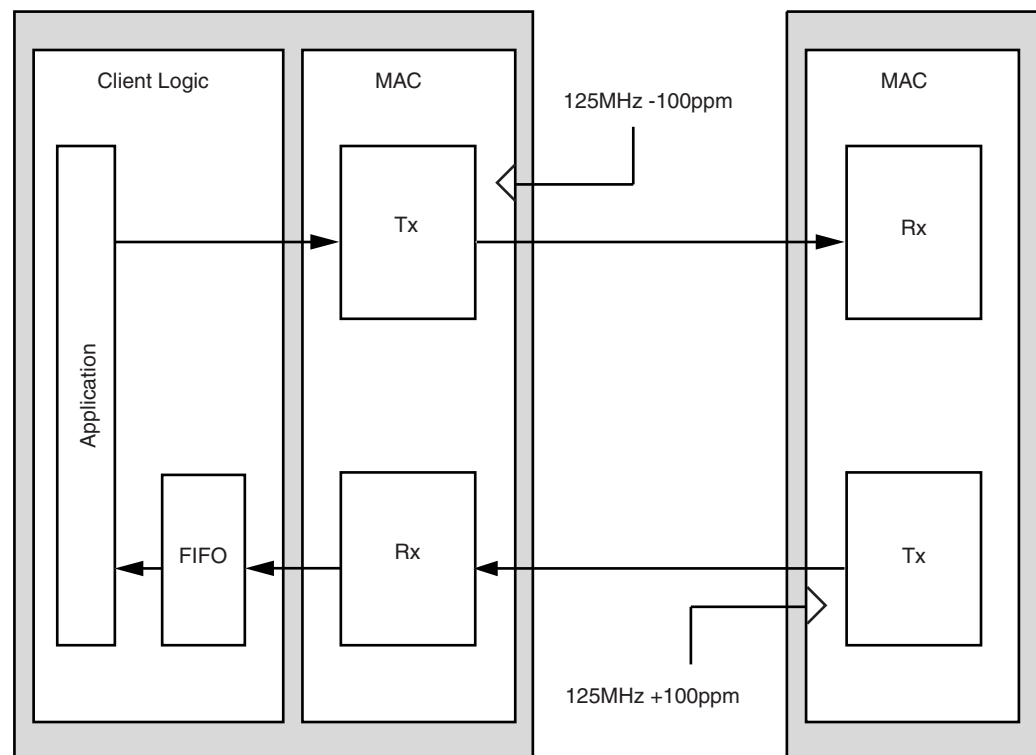


Figure 6-1: The Requirement for Flow Control

Figure 6-1 illustrates the requirement for Flow Control at 1 Gbps. The MAC on the right side of the figure has a reference clock slightly faster than the nominal 125 MHz. The MAC on the left side of the figure has a reference clock slightly slower than the nominal 125 MHz. This results in the MAC on the left side of the figure not being able to match the full line rate of the MAC on the right side (due to clock tolerances). The MAC at the left is illustrated as performing a loopback implementation, which results in the FIFO filling up over time. Without Flow Control, this FIFO will eventually fill and overflow, resulting in the corruption or loss of ethernet frames. Flow Control is one solution to this problem.

Flow Control Basics

A MAC may transmit a Pause Control frame to request that its link partner cease transmission for a specific period of time. For example, the left MAC in Figure 6-1 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 right MAC of Figure 6-1 may cease transmission after receiving the Pause Control frame transmitted by the left MAC. In a well designed system, the right MAC ceases transmission before the client FIFO of the left MAC overflows to provide time to empty the FIFO to a safe level before resuming normal operation. This practice 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 standard. Control frames are identified from other frame types by a defined value placed into the length/type field (the MAC Control Type code). Figure 6-2 illustrates control frame format.

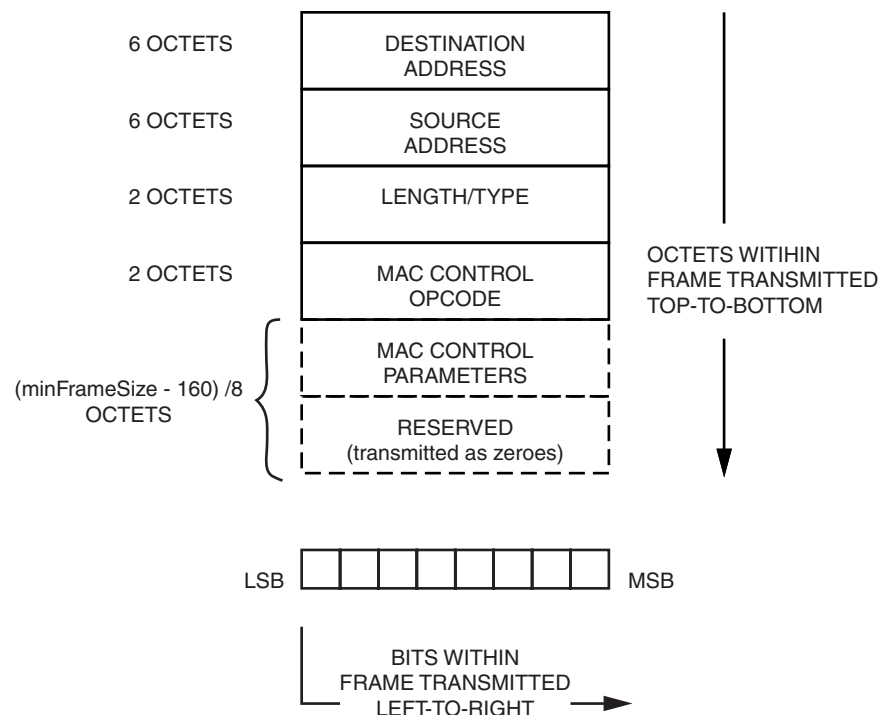


Figure 6-2: MAC Control Frame Format

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). At 1 Gbps, a single *pause_quantum* corresponds to 512 ns. At 100 Mbps, a single *pause_quantum* corresponds to 5120 ns, and at 10 Mbps, a single *pause_quantum* corresponds to 51200 ns.

Flow Control Operation of the TEMAC

Transmitting a Pause Control Frame

Core-initiated Pause Request

If the TEMAC core is configured to support transmit flow control, the client may initiate a flow control frame by asserting `clientemacpausereq` while the pause value is on the `clientemacpauseval` bus. If the core is generated with the `clock_enable` option set to false, these signals are synchronous to `txcoreclk`. Figure 6-3 illustrates this timing.

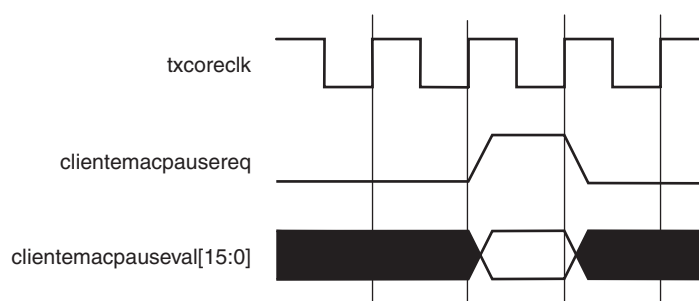


Figure 6-3: Pause Request Timing

If the MAC core is generated with the `clock_enable` option set to true, the signals are synchronous to `txgmiimiclk`. Figure 6-4 illustrates this timing.

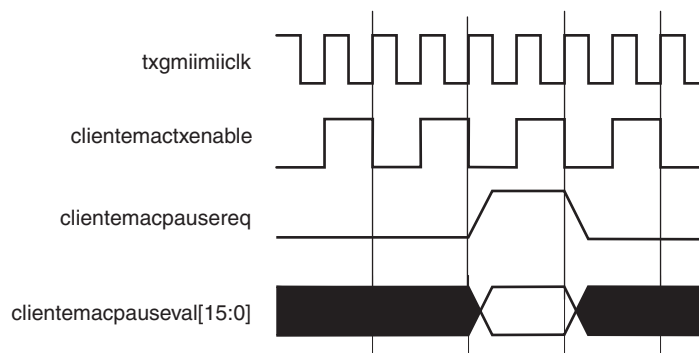


Figure 6-4: Pause Request Timing with Clock Enables

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 6-2](#)):

- The destination address used is an *IEEE 802.3* 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" on page 81](#)).
- The value sampled from the `clientmacpauseval[15:0]` port at the time of the `clientmacpausereq` 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, 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 `clientmacpausereq` signal is asserted numerous times in a short time period (before the control pause frame transmission has had a chance to begin), only a single pause control frame will be transmitted. The `clientmacpauseval[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 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 ["Transmitting Outbound Frames" on page 47](#)).

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 6-2](#). It must pass all standard receiver frame checks (e.g. 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 `emacclientrxbadframe` signal asserted.

Pause Frame Reception Disabled

When pause control reception is disabled, an error free control frame is received through the client interface with the `emacclientrxgoodframe` signal asserted. In this way, the frame is passed to the client logic for interpretation (see ["Client-initiated Response to a Pause Request" on page 61](#)).

Pause Frame Reception Enabled

When pause control reception is enabled 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* globally assigned multicast address or the configurable Pause Frame MAC Address (see ["Configuration Registers" on page 81](#)).
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 the previously listed 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 `emacclientrxgoodframe` asserted. It is then the responsibility of the MAC client logic to decode, act on (if required) and drop this control frame.

If all the previously listed 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 `clientemactxack` 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 `emacclientrxbadframe` 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 in the length/type field should be dropped by the receiver client logic. All Control frames are indicated by `emacclientrxstats` bit 19 (see ["Receiver Statistics Vector" on page 45](#)).

Client-initiated Response to a Pause Request

For maximum flexibility, flow control logic can be disabled in the core 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 `emacclientrxgoodframe` 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 6-1](#). To summarize the example, 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 to occupancy threshold (7/8 occupancy is used in this description). 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 `clientemacpauseval[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). 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

`clientemacpauseval[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 6-5 illustrates the FIFO occupancy over time.

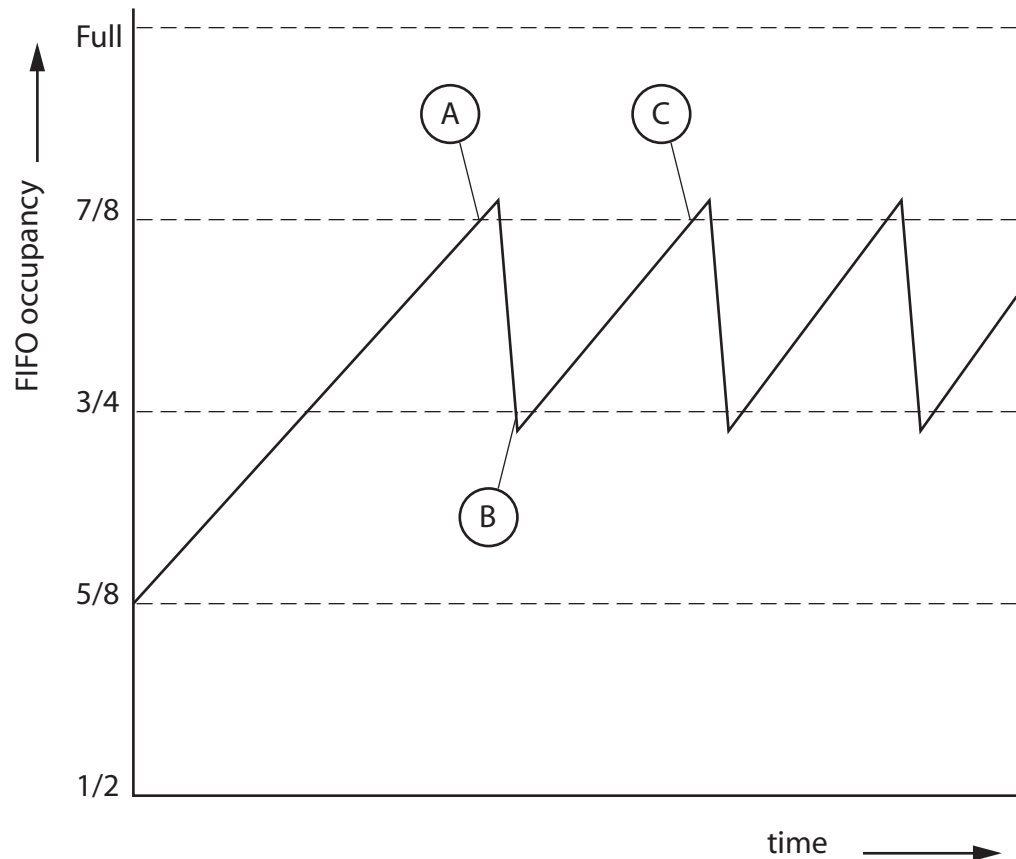


Figure 6-5: Flow Control Implementation Triggered from FIFO Occupancy

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.

Using the Physical Side Interface

The HDL example design supplied with the core implements an external GMII/MII or RGMII interface. These are typically attached to an external PHY device. For more information about the example design files, see the *Tri-Mode Ethernet MAC Getting Started Guide*.

Implementing External GMII

GMII/MII Transmit Interface

Virtex-II Pro, Virtex-II, Spartan-3, Spartan-3E, and Spartan-3A Devices

[Figure 7-1](#) shows a block diagram of the GMII/MII transmit interface. The signal names in the figure match those in the HDL example design. There are two transmit clock inputs to the chip:

- `gtx_clk` is a user-supplied 125 MHz clock. This is used to derive the core and `gmii/mii` clocks when running at 1 Gbps.
- `mii_tx_clk` is provided by the PHY and runs at 25 MHz when the device is running at 100 Mbps and at 2.5 MHz when it is operating at 10 Mbps.

The clock generator module takes the clock inputs and generates `tx_gmii_mii_clk_int`. This runs at 125 MHz, 25 MHz, or 2.5 MHz depending on the MAC operating speed. This is used to drive the GMII/MII logic in the example design and the core. See ["Clocking" on page 117](#) for more information on the clock generator circuit.

[Figure 7-1](#) shows that the output transmitter signals are registered in device IOBs before driving them to the device pads. The logic required to forward the transmitter clock is also shown: this uses an IOB output double-data-rate (DDR) register so that the clock signal produced incurs exactly the same delay as the data and control signals. This clock signal, `gmii_tx_clk`, is inverted with respect to `tx_gmii_mii_clk_int` so that the rising edge of `gmii_tx_clk` will occur in the center of the data valid window, therefore maximizing setup and hold times across the interface.

The half-duplex signals `gmii_col` and `gmii_crs` are asynchronous to the transmit clock. These are routed through PADs and IOBs and then input to the core.

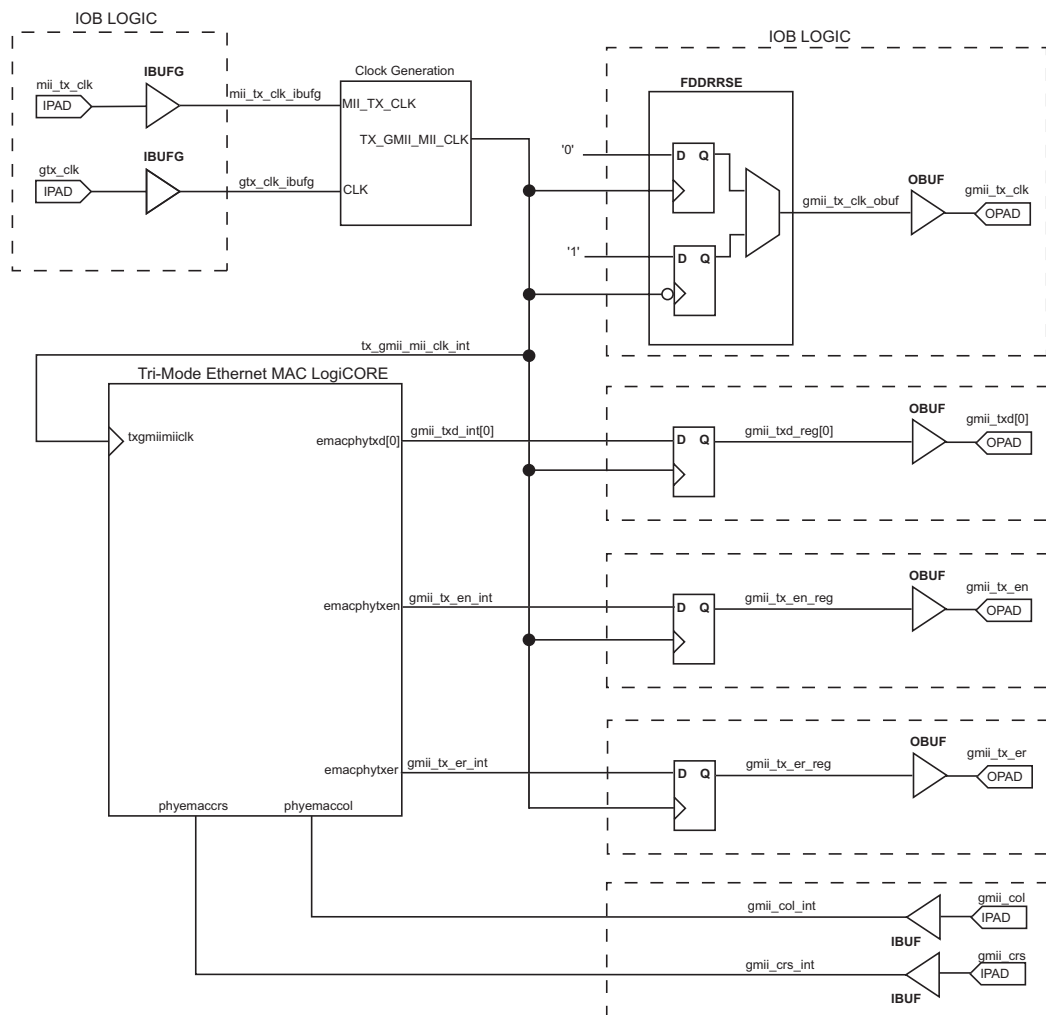


Figure 7-1: External GMII/MII Transmit Interface

Virtex-4 and Virtex-5 Devices

Figure 7-2 illustrates how to implement a GMII/MII transmit interface when either a Virtex-4 or Virtex-5 device is selected. An ODDR component is used instead of an FDDRSE component to generate a transmitter clock. This is designed so that the rising edge of `gmii_tx_clk` will occur in the center of the data valid window, therefore maximizing setup and hold times across the interface.

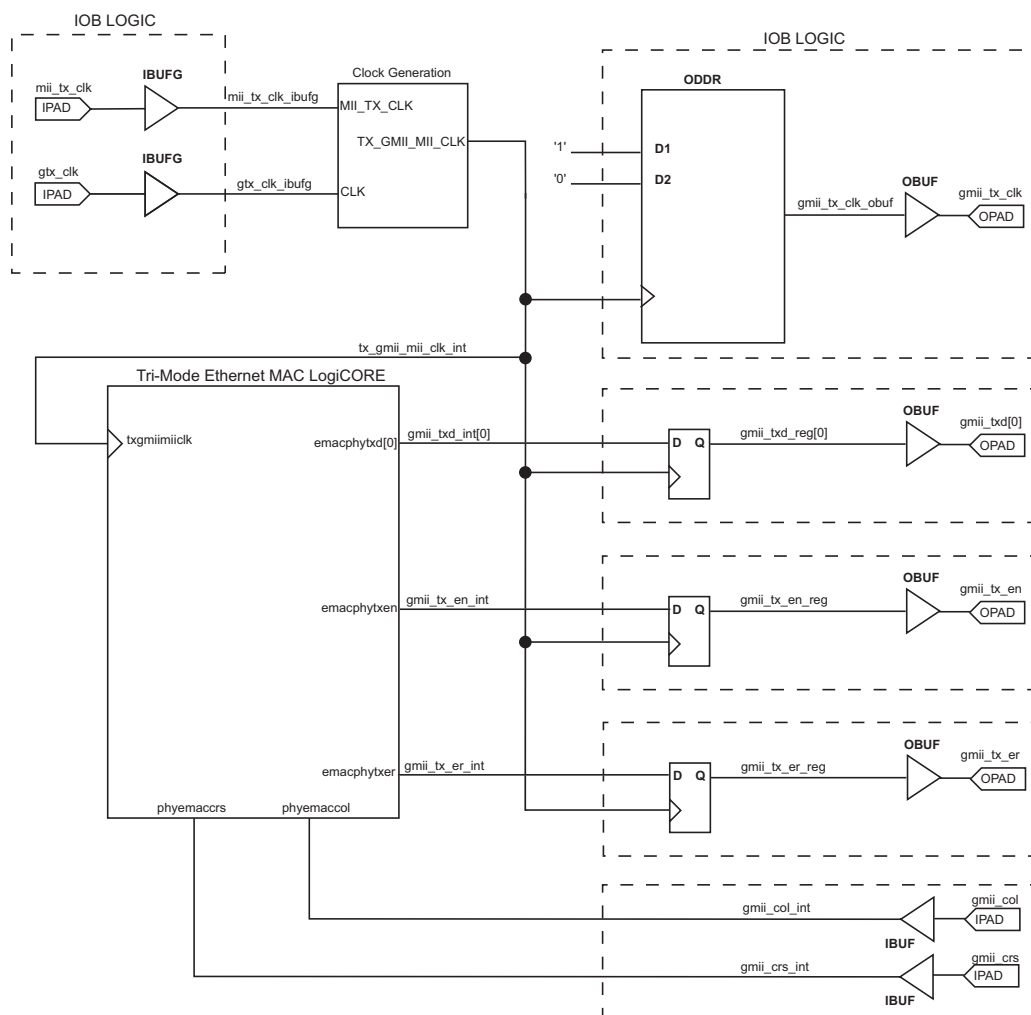


Figure 7-2: External GMII/MII Transmit Interface in a Virtex-4/Virtex-5 Device

GMII/MII Receive Interface

Virtex-II and Virtex-II Pro Devices

Figure 7-3 shows how to implement an external GMII/MII receiver interface.

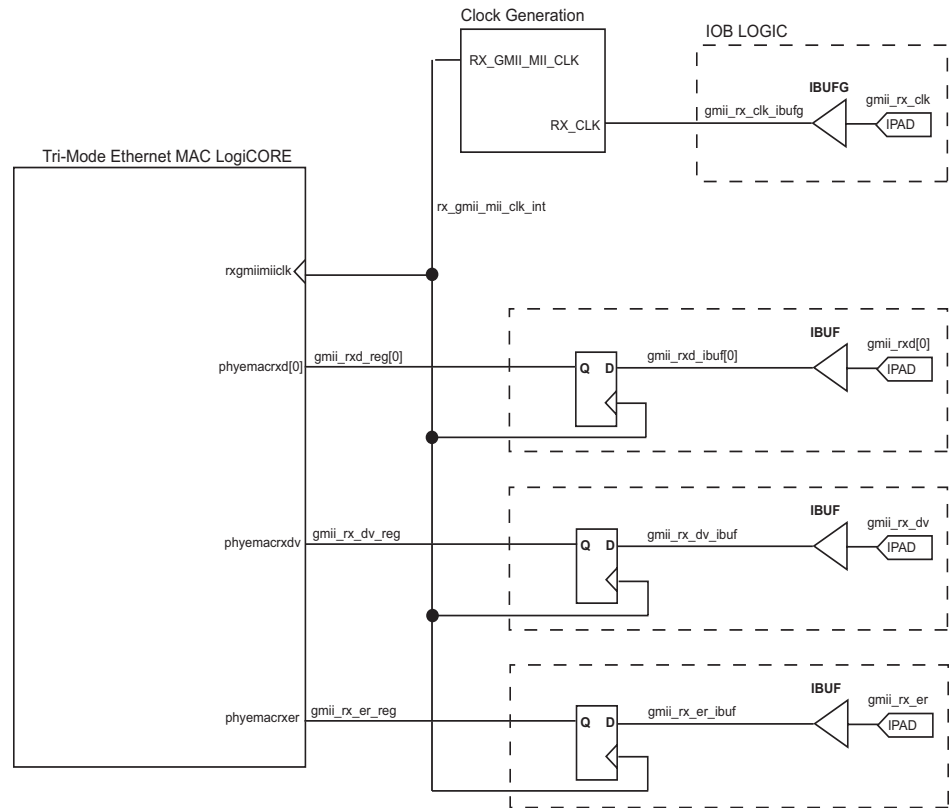


Figure 7-3: External GMII/MII Receive Interface

The clock generator module derives `rx_gmii_mii_clk` from the `gmii_rx_clk` input from the PHY. This runs at 125 MHz, 25 MHz or 2.5 MHz, depending on the speed, and is used to clock the GMII/MII logic in the HDL example design and the core. For more information on the clock generator module, see "Clocking" on page 117.

Figure 7-3 shows that the input receiver signals are registered in device IOBs before driving them to the device pads.

Spartan-3, Spartan-3E, and Spartan-3A Devices

The logic described in the previous section does not meet the input setup and hold requirements for GMII with Spartan-3, Spartan-3E, and Spartan-3A devices. In these devices, a DCM must be used on the `gmii_rx_clk` clock path as illustrated in Figure 7-4. This is performed by the example design delivered with the core (all signal names and logic match).

Phase shifting may then be applied to the DCM to fine-tune the setup and hold times at the GMII IOB input flip-flops; fixed-phase shift is applied to the DCM via the example UCF for the example design. For more information, see "Calculating the DCM Phase Shift" in Appendix D.

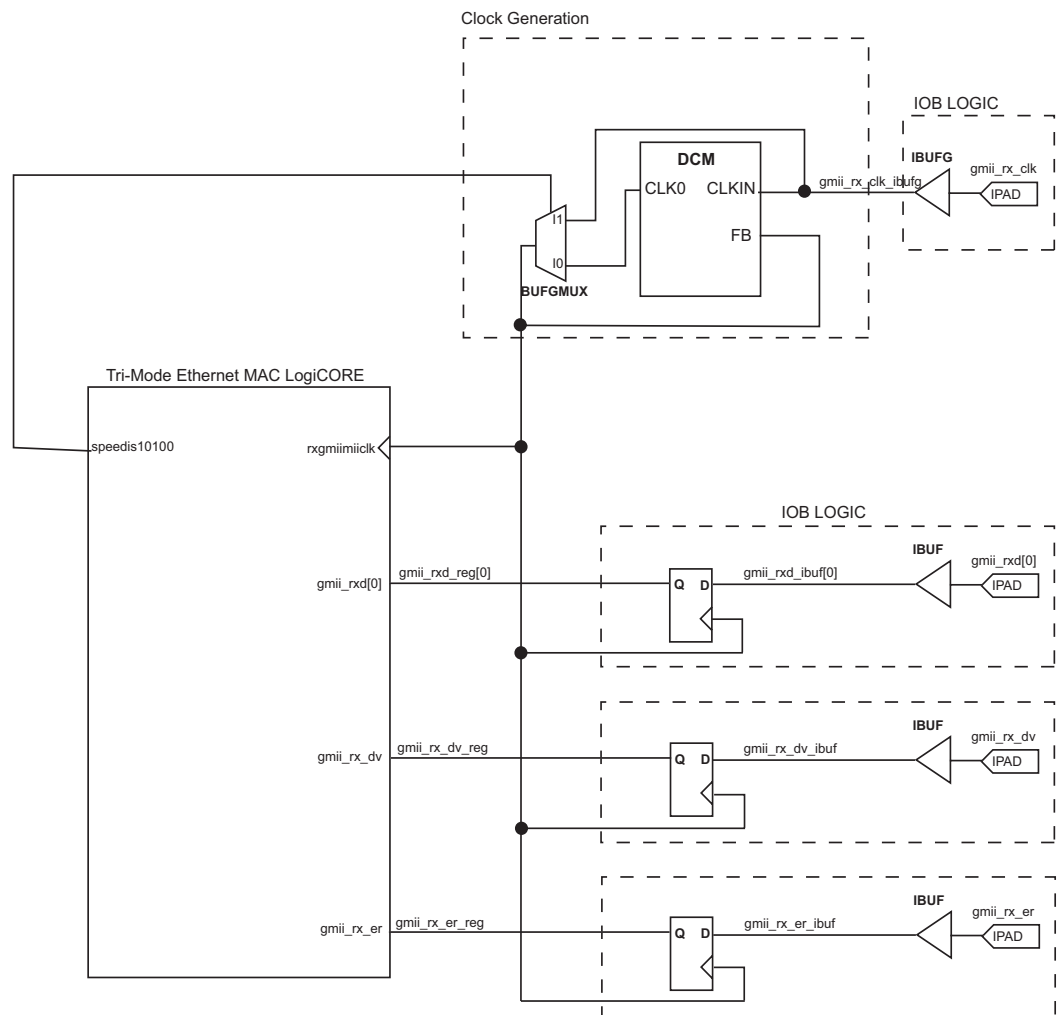


Figure 7-4: GMII/MII Receive Logic for Spartan-3, Spartan-3E, and Spartan-3A Devices

As there are only 8 BUFGMUXes on the Spartan-3 device, the clock multiplexing for the txgmiiimiclk is performed in the FPGA fabric (see [Figure 10-15](#)). The Spartan-3 device should always be reset after a speed change in order to avoid the core entering an undefined state due to a glitch on the txgmiiimiclk signal.

Virtex-4 Devices

The logic described in "GMII/MII Receive Interface" on page 66 does not meet the input setup and hold requirements for GMII with Virtex-4 devices. An IDELAY component may be used on the clock, data and control paths, as illustrated in Figure 7-5. These can be used to either shift the input clock `gmii_rx_clk` or shift the data and control signals to meet the setup and hold requirements and to allow for any bus skew across the data and control inputs. The IDELAY components are used in fixed delay mode, where the attribute `IOBDELAY_VALUE` determines the tap delay value. An IDELAYCTRL primitive must be instantiated for this mode of operation. See the *Virtex-4 User Guide* for more information about using the IDELAYCTRL and IDELAY components.

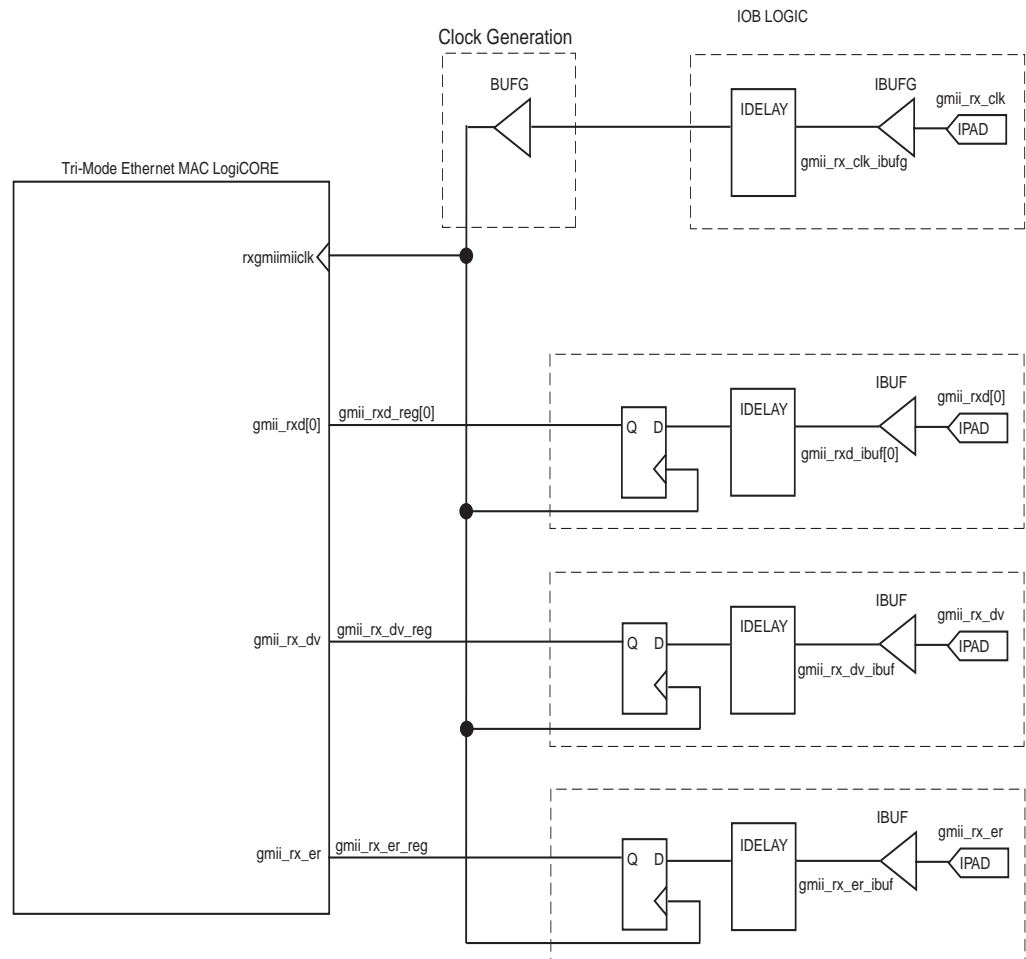


Figure 7-5: GMII/MII Receive Logic for Virtex-4 Devices

Virtex-5 Devices

The logic described in "GMII/MII Receive Interface" on page 66 does not meet the input setup and hold requirements for GMII with Virtex-5 devices. An IODELAY component may be used on the data and control signals, as illustrated in Figure 7-6. These can be used to either shift the input clock `gmii_rx_clk` or the data and control signals to meet the setup and hold requirements and to allow for any bus skew across the data and control inputs. The IODELAY components are used in fixed delay mode, where the attribute `IDELAY_VALUE` determines the tap delay value. An IDELAYCTRL primitive must be instantiated for this mode of operation. See the *Virtex-5 User Guide* for more information about using the IDELAYCTRL and IODELAY components.

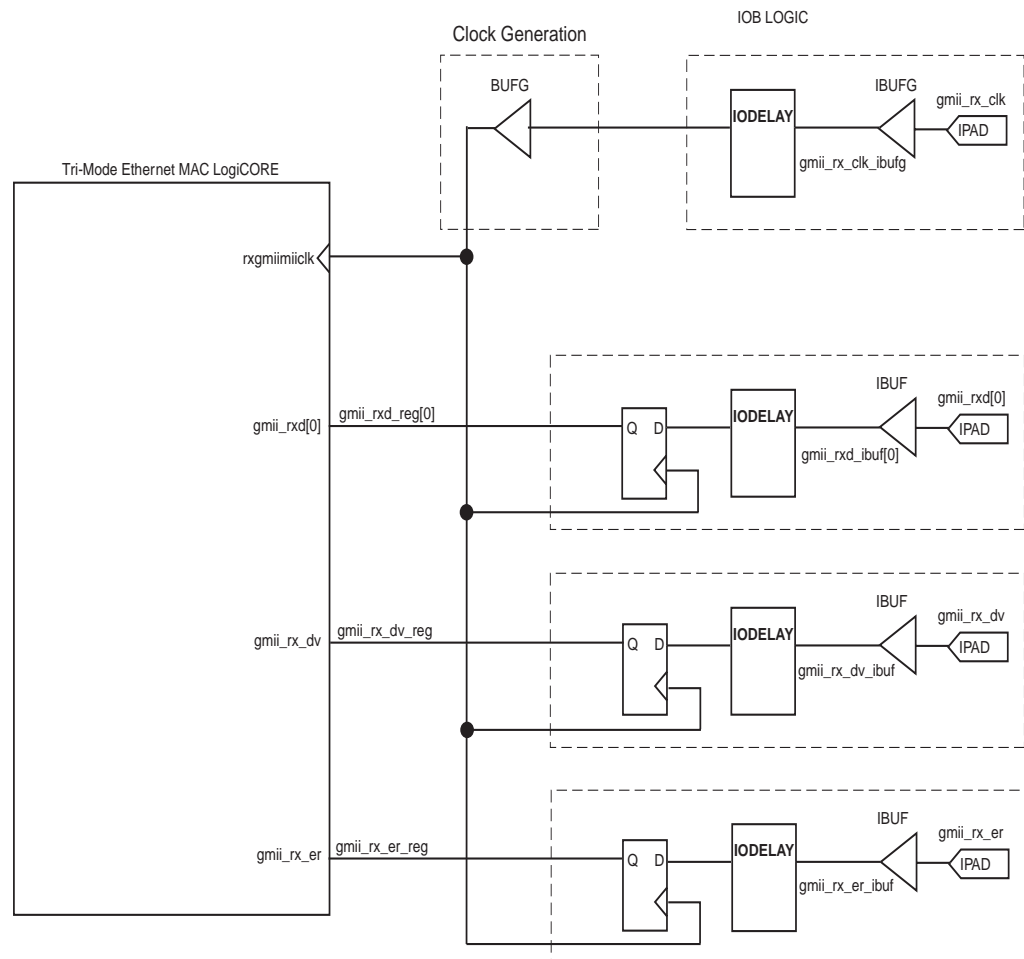


Figure 7-6: GMII/MII Receive Logic for Virtex-5 Devices

Implementing External RGMII

RGMII Transmit Interface

Virtex-II Pro, Virtex-II, Spartan-3, and Spartan-3A Devices

The RGMII interface is designed to the RGMII V2.0 specification. Figure 7-7 shows a block diagram of the RGMII transmit interface in a Virtex-II device. The signal names in the figure match those in the HDL example design. There is one transmit clock input to the chip, `gtx_clk`. This is a 125 MHz clock that is used to generate the RGMII transmit clock output at all speeds. For more information on the clock generator module, see "Clocking" on page 117.

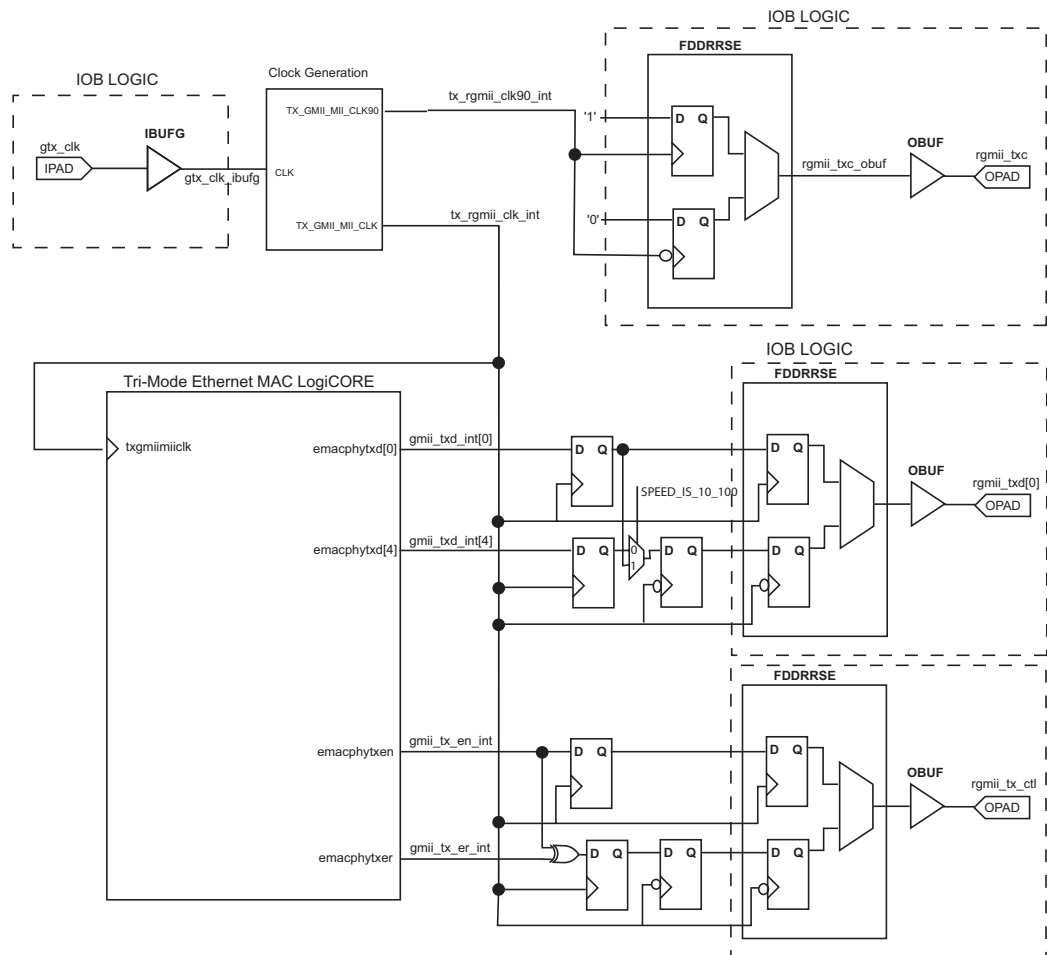


Figure 7-7: External RGMII Transmit Interface

The output transmitter signals are registered on `tx_rgmii_clk_int`, in the FPGA fabric, including the encoded `rgmii_tx_ctl_int` signal, derived from the logical `xor` of `gmii_tx_en_int` and `gmii_tx_er_int`. The signals to be transmitted on the `rgmii` falling clock edge are then registered on the falling edge clock. This ensures that the data is presented to the double data rate registers at the correct time. Finally the transmitter signals are registered by an IOB output double-data-rate (DDR) register before being driven to output pads.

The logic required to forward the transmitter clock is also shown: this uses an IOB output double-data-rate (DDR) register so that the clock signal produced incur exactly the same delay as the data and control signals. At 1 Gbps this clock signal, `tx_rgmii_clk90_int`, is phase shifted by 90 degrees in the clock generator module with respect to `tx_rgmii_clk_int`, so that the rising edge of `rgmii_txc` will occur in the center of the data valid window. This maximizes setup and hold times across the interface, as specified in the Reduced Gigabit Media Independent interface (RGMII) Version 2.0 specification.

Virtex-4 Devices

Figure 7-8 illustrates how to use the physical transmitter interface of the core to create an external RGMII in a Virtex-4 family device. The signal names and logic shown on the figure match those delivered with the example design when the RGMII is chosen.

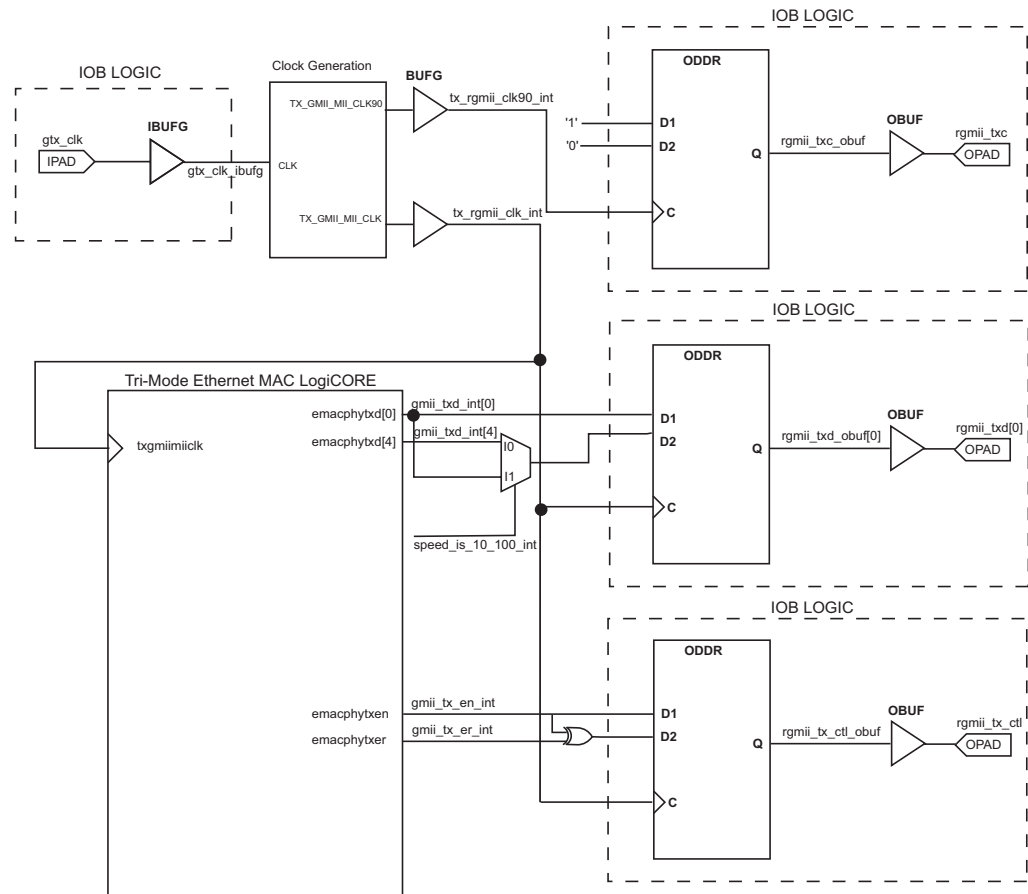


Figure 7-8: External RGMII Transmit Interface in a Virtex-4 Device

Figure 7-8 shows that the output transmitter signals are registered in the IOBs in ODDR components. These components convert the input signals into one double data rate signal. These signals are then output through OBUFs before being driven to output pads.

The logic required to forward the transmitter clock is also shown. This uses an ODDR register so that the clock signal produced incur exactly the same delay as the data and control signals. At 1 Gbps this clock signal, `tx_rgmii_clk90_int`, is phase shifted by 90 degrees in the clock generator module with respect to `tx_rgmii_clk_int`, so that the rising edge of `rgmii_txc` will occur in the center of the data valid window. This

maximizes setup and hold times across the interface, as specified in the Reduced Gigabit Media Independent interface (RGMI) Version 2.0 specification.

Virtex-5 Devices

The example design provided for a Virtex-5 device is significantly different from the other families, as it has been designed to use new architecture features in order to reduce the number of global clocking resources required. Figure 7-9 shows how an external RGMII physical interface has been created for a Virtex-5 device.

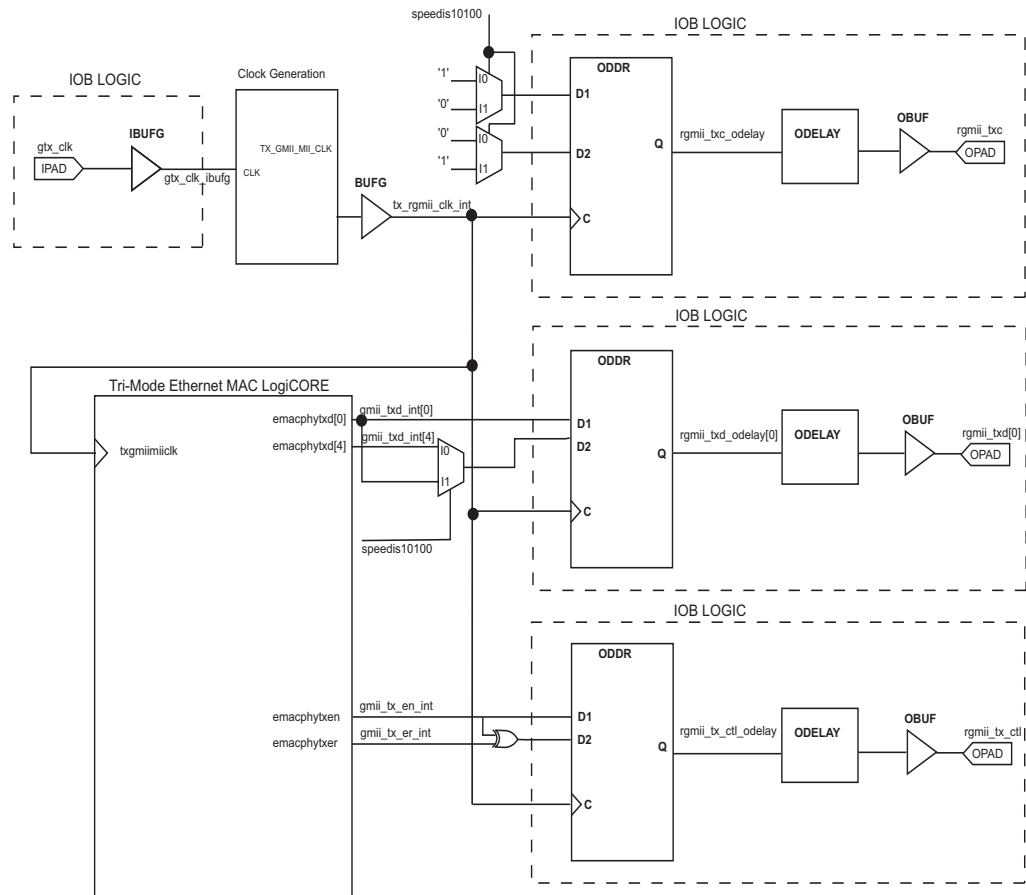


Figure 7-9: External RGMII Transmit Interface in a Virtex-5 Device

At 1 Gbps the clock signal, `rgmii_tx_odelay`, is phase shifted by 90 degrees with respect to `tx_clk_int` by the `IODELAY` component, so that the rising edge of `rgmii_txc` will occur in the center of the data valid window. This will maximize the setup and hold times across the interface, as specified in the Reduced Gigabit Media Independent interface (RGMII) Version 2.0 specification. The `IODELAY` component is used in fixed delay mode, where the attribute `ODELAY_VALUE` determines the tap delay value. An `IDELAYCTRL` primitive must be instantiated for this mode of operation. Refer to the *Virtex-5 User Guide* for more information on the use of `IDELAYCTRL` and `IODELAY` components.

For 100 Mbps and 10 Mbps, the clock signal `rgmii_tx_odelay` is inverted with respect to `tx_clk_int` so that the rising edge of the clock is approximately in the middle of the data window.

The RGMII data/control signals are routed through `IODELAY` components with an `ODELAY_VALUE` of zero to provide similar path delays to that of the clock signal.

RGMII Receiver Interface

Virtex-II Pro, Virtex-II, Spartan-3, and Spartan-3A Devices

Figure 7-10 shows a block diagram of the RGMII receiver interface in a Virtex-II device. The input receiver signals are registered in device IOBs on rising and falling edges of `gmii_rx_clk_bufg`. The signals are then registered inside the FPGA fabric before a final register stage to synchronize signals to the rising edge clock. In order to achieve the required setup and hold times across the interface, the clock generator uses a DCM with a phase shift to adjust the clock relative to the data. See "Clocking" on page 117 and "Calculating the DCM Phase Shift," on page 145.

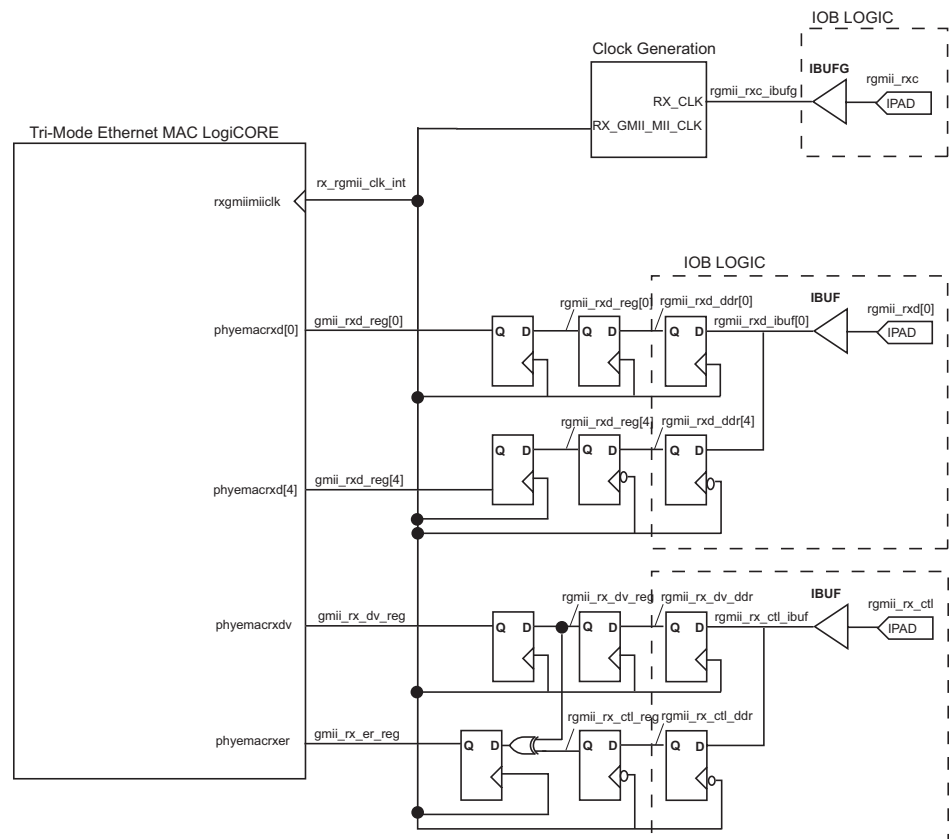


Figure 7-10: External RGMII Receive Interface

Virtex-4 Devices

Figure 7-11 illustrates how to use the physical receiver interface of the core to create an external RGMII in a Virtex-4 family device. The signal names and logic shown on the figure match those delivered with the example design when the RGMII is selected.

Figure 7-11 shows that the input receiver signals are registered in the IOBs in IDDR components. These components convert the input double data rate signals into GMII specification signals. The `gmii_rx_er_reg` signal is derived in the FPGA fabric from the outputs of the control IDDR component.

IDELAY components can be used to phase-shift the input RGMII clock, data and control signals to meet the setup and hold margins. The IDELAY components are used in fixed delay mode, where the attribute `IOBDELAY_VALUE` determines the tap delay value. An

IDELAYCTRL primitive must be instantiated for this mode of operation. See the *Virtex-4 User Guide* for more information about using the IDELAYCTRL and IDELAY components.

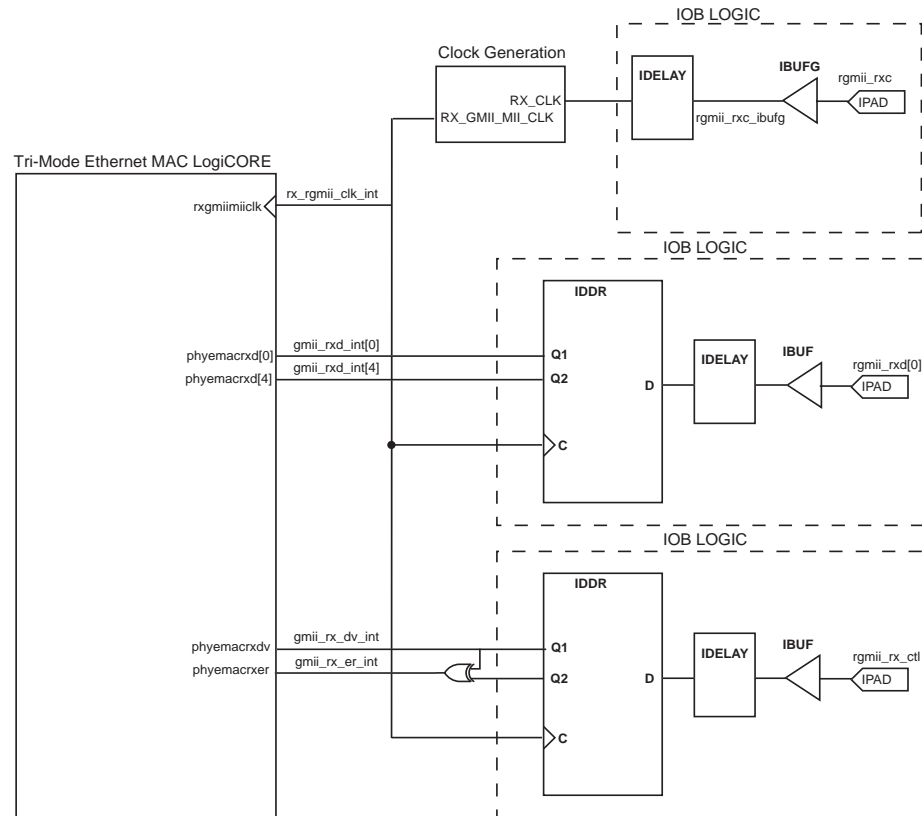


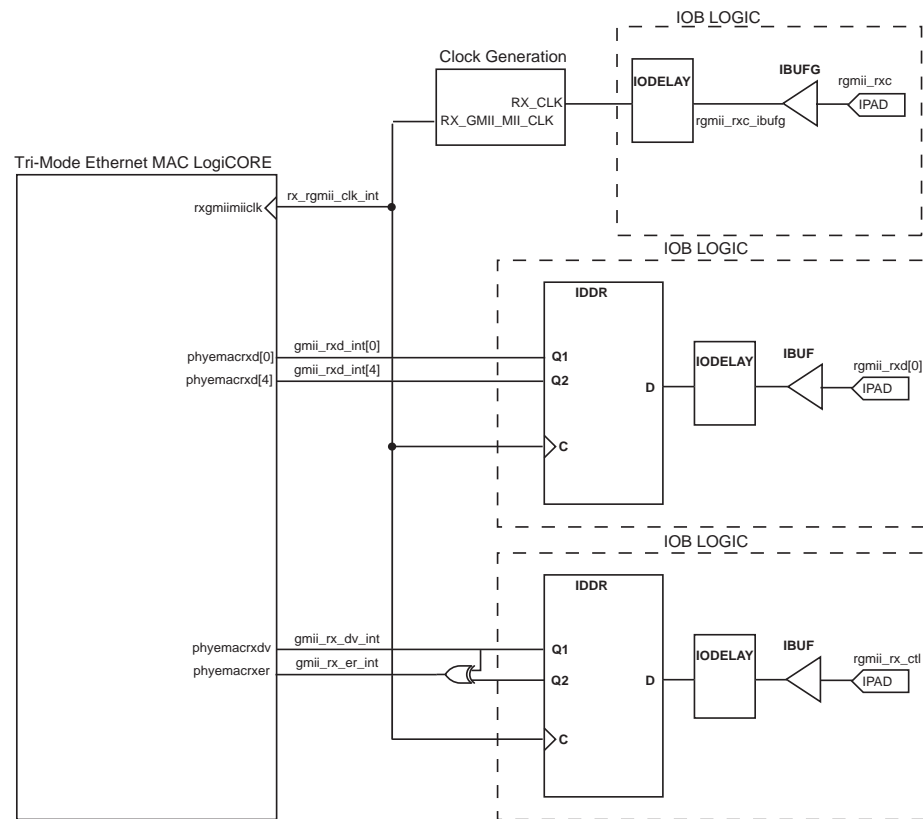
Figure 7-11: External RGMII Receive Interface in Virtex-4 Devices

Virtex-5 Devices

Figure 7-12 illustrates how to use the physical receiver interface of the core to create an external RGMII in a Virtex-5 family device. The signal names and logic shown on the figure match those delivered with the example design when the RGMII is selected.

Figure 7-12 shows that the input receiver signals are registered in the IOBs in IDDR components. These components convert the input double data rate signals into GMII specification signals. The gmii_rx_er_reg signal is derived in the FPGA fabric from the outputs of the control IDDR component.

IDELAY components can be used to phase-shift the input RGMII clock, data and control signals to meet the setup and hold margins. The IDELAY components are used in fixed delay mode, where the attribute IDELAY_VALUE determines the tap delay value. An IDELAYCTRL primitive must be instantiated for this mode of operation. See the *Virtex-5 User Guide* for more information about using the IDELAYCTRL and IDELAY component.



RGMII Inband Status Decoding Logic

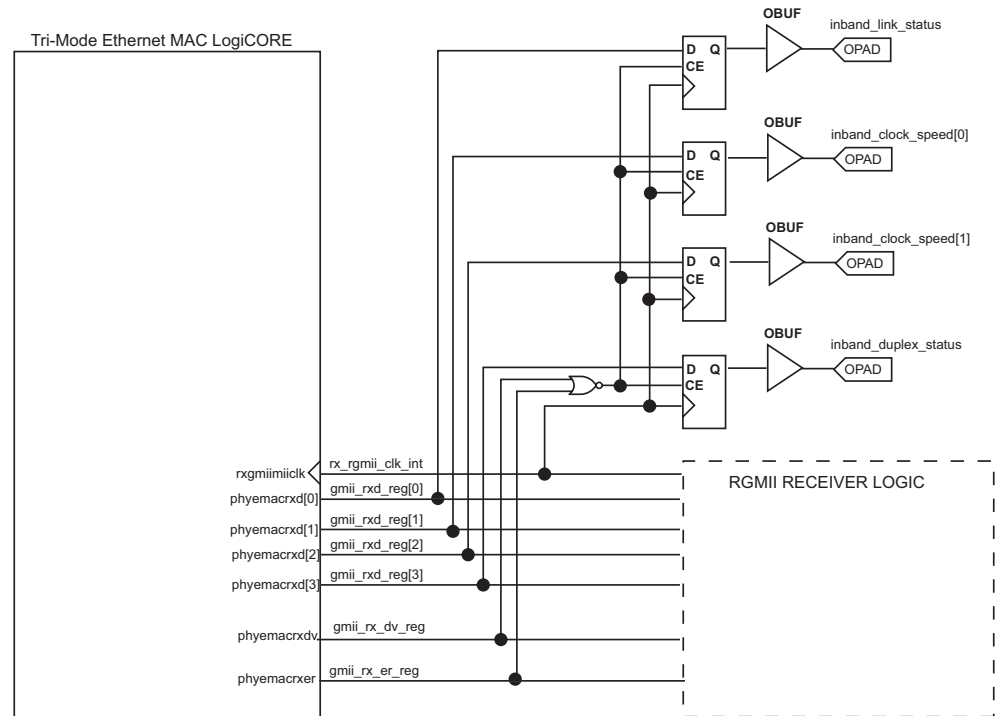


Figure 7-13: RGMII Inband Status Logic

The Inband Status decoding logic is common to all device families. [Figure 7-13](#) illustrates how to decode the RGMII inband status information, that is received through the RGMII interface between frames, in a Virtex-II family device. The signal names and logic shown in the figure exactly match those delivered with the example design when the RGMII is chosen. If other families are chosen, equivalent primitives and logic specific to that family will automatically be used in the example design.

Using the MDIO Interface

This interface is accessed through the optional Management Interface (see [“Accessing MDIO via the TEMAC,” on page 92](#)) and is typically connected to the MDIO port of a physical layer device (PHY) to access its configuration and status registers. The MDIO format is defined in *IEEE 802.3* clause 22.

Connecting the MDIO to an Internally Integrated PHY

The MDIO ports of the TEMAC core can be connected to the MDIO ports of an internally integrated physical layer device. For example, the MDIO port of the Ethernet 1000BASE-X PCS/PMA or SGMII from Xilinx (see [“Integrating with the Ethernet 1000BASE-X PCS/PMA or SGMII Core,” on page 127](#)).

Connecting the MDIO to an External PHY

The MDIO ports of the TEMAC core can be connected to the MDIO of an external PHY. In this situation, `phyemacmdin`, `emacphyumdout` and `emacphyumdtri` must be connected to a Tri-State buffer to create a bidirectional wire, `mdio`. This Tri-State buffer can be either external to the FPGA, or internally integrated by using an IOB IOBUF component with an appropriate SelectIO™ standard for the external PHY. (This is illustrated in Figure 7-14.)

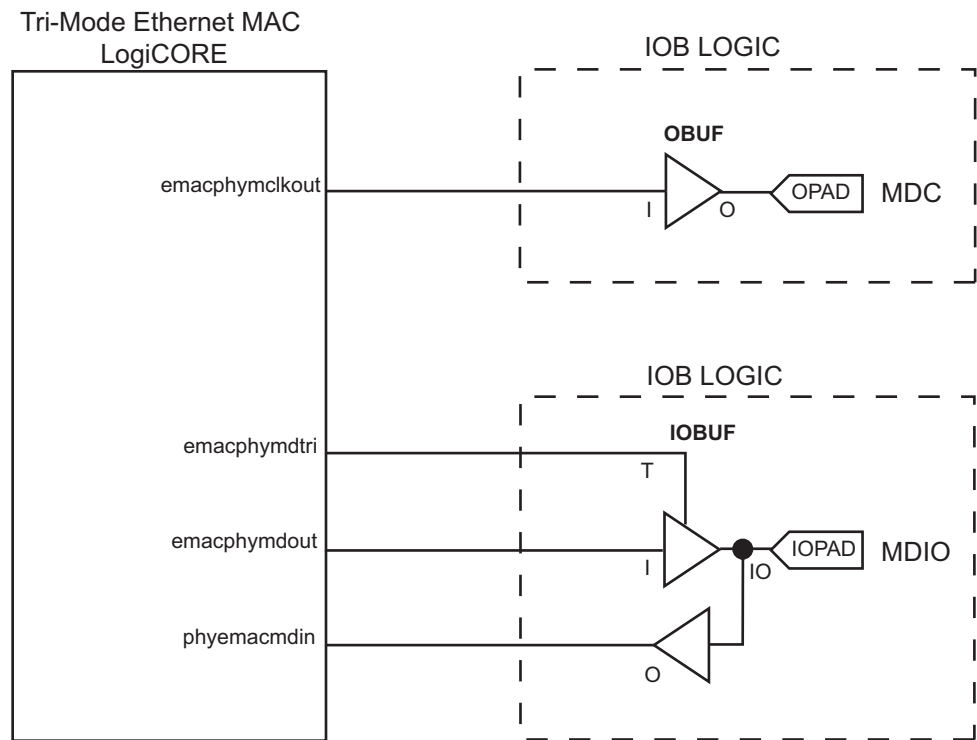


Figure 7-14: External MDIO Interface

Connecting the MDIO to an External and Internal PHY

The MDIO can connect to more than one device. If an internal PHY is present but the device is also connected to external devices via the MDIO, an arbitration circuit is required. An example circuit is shown in Figure 7-15. Both PHY devices must be assigned an unique physical address.

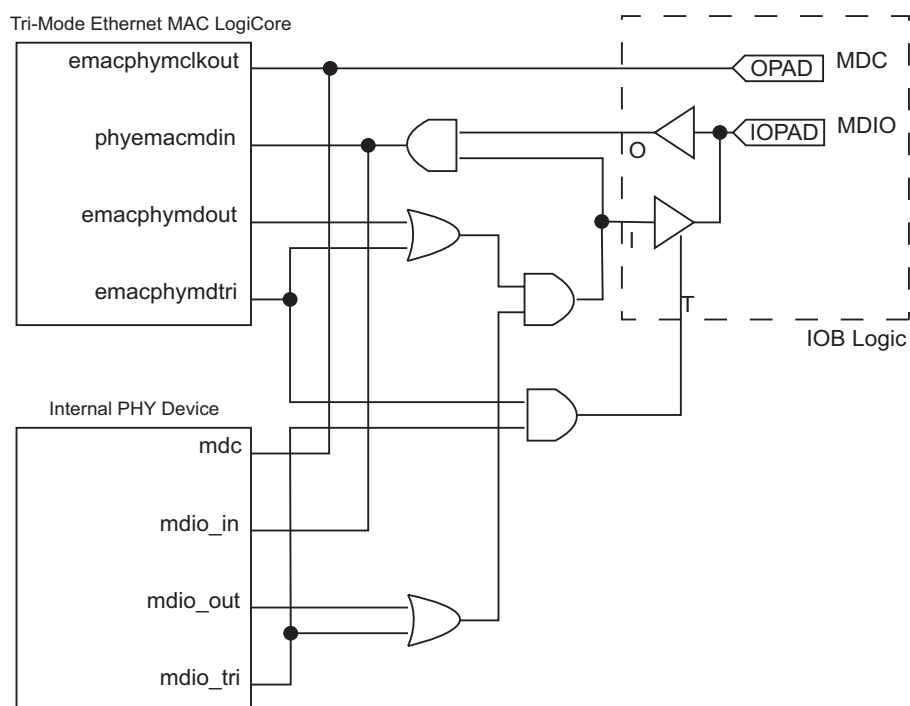


Figure 7-15: Internal and External MDIO Interfaces

Configuration and Status

Using the Optional Management Interface

The Management Interface is a processor-independent interface with standard address, data, and control signals. It may be used as is, or a wrapper (not supplied) may be applied to interface to common bus architectures.

This interface is used for:

- Configuration of the MAC core
- Access through the MDIO interface to the management registers located in the PHY attached to the MAC core

The Management Interface is accessed differently depending on the type of transaction; a truth table showing which access method is required for each transaction type is shown in [Table 8-1](#). These access methods are described in the following sections.

Table 8-1: Management Interface Transaction Types

Transaction	HOST_MIIM_SEL	HOST_ADDR[9]
Configuration	0	1
MIIM access	1	X

hostclk

The Management Interface clock, `hostclk`, is used to derive the MDIO clock, `emacphyclkout`. To save on clock resources, it may be tied to the user supplied 125 MHz input clock.

Configuring the MAC core to derive the MDC signal from this clock is detailed in [“Accessing MDIO via the TEMAC,” on page 92](#).

Configuration Registers

After power up or reset, the client may reconfigure the core parameters from their defaults, such as flow control support. Configuration changes can be written at any time. Both the receiver and transmitter logic will only respond to configuration changes during inter-frame gaps. The exceptions to this are the configurable 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 8-2](#). As can be seen, the address has some implicit don't care bits; any access to an

address in the ranges shown will perform a 32-bit read or write from the same configuration word.

Table 8-2: Configuration Registers

Address	Description
0x200-0x23F	Receiver Configuration (Word 0)
0x240-0x27F	Receiver Configuration (Word 1)
0x280-0x2BF	Transmitter Configuration
0x2C0-0x2FF	Flow Control Configuration
0x300-0x31F	MAC Speed Configuration
0x320-0x33F	Reserved
0x340-0x37F	Management Configuration
0x380-0x383	Unicast Address (Word 0) (if address filter is present)
0x384-0x387	Unicast Address (Word 1) (if address filter is present)
0x388-0x38B	Address Table Configuration (Word 0) (if address filter is present)
0x38C-0x38F	Address Table Configuration (Word 1) (if address filter is present)
0x390-0x3BF	Address Filter Mode (if address filter is present)

Register Maps

The register contents for the two receiver configuration words can be seen in [Table 8-3](#) and [Table 8-4](#).

Table 8-3: Receiver Configuration Word 0

Bit	Default Value	Description
31-0	All 0's	<p>Pause frame MAC Source 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.</p> <p>The address is ordered so the first byte transmitted/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.</p>

Table 8-4: Receiver Configuration Word 1

Bit	Default Value	Description
15-0	All 0's	Pause frame MAC Source Address [47:32] See description in Table 8-3 .
24-16	N/A	Reserved
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 ," on page 44. When this bit is set to '0,' the length/type field checks will be performed: this is normal operation.
26	0	Half Duplex If '1,' the receiver will operate in half-duplex mode. If '0,' the receiver will operate in full-duplex mode.
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 " Client-Supplied FCS Passing ," on page 50. When it is '0,' the client will not be passed the FCS. In both cases, the FCS will be verified on the frame.
30	0	Jumbo Frame Enable When this bit is set to '1,' the MAC receiver will accept frames over the specified <i>IEEE 802.3-2002</i> maximum legal length. When this bit is '0,' the MAC will only accept frames up to the specified maximum.
31	0	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.

The register contents for the Transmitter Configuration Word are described in [Table 8-5](#).

Table 8-5: Transmitter Configuration Word

Bit	Default Value	Description
24-0	N/A	Reserved
25	0	Interframe Gap Adjust Enable If '1,' the transmitter will read the value on the port clientemactxifgdelay at the start of frame transmission and adjust the interframe gap following the frame accordingly (see “Interframe Gap Adjustment: Full-Duplex Mode Only,” on page 54). If '0,' the transmitter will output a minimum interframe gap of at least twelve clock cycles, as specified in <i>IEEE 802.3-2002</i> .
26	0	Half Duplex If '1,' the transmitter will operate in half-duplex mode.
27	0	VLAN Enable When this bit is set to '1,' the transmitter will recognize the transmission of VLAN tagged frames.
28	1	Transmit Enable When this bit is '1,' the transmitter is operational. When it is '0,' the transmitter is disabled.
29	0	In-band FCS Enable When this bit is '1,' the MAC transmitter will expect the FCS field to be passed in by the client as described in “Client-Supplied FCS Passing,” on page 50. When this bit is '0,' the MAC transmitter will append padding as required, compute the FCS 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 specified <i>IEEE 802.3-2002</i> maximum legal length. When this bit is '0,' the MAC will only send frames up to the specified maximum.
31	0	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.

The register contents for the Flow Control Configuration Word are described in [Table 8-6](#).

Table 8-6: 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 Control Frame,” on page 60. When this bit is '0,' received flow control frames will always be passed up to the client.

Table 8-6: Flow Control Configuration Word

Bit	Default Value	Description
30	1	Flow Control Enable (TX) When this bit is '1,' asserting the <code>clientemacpausereq</code> signal sends a flow control frame out from the transmitter as described in "Transmitting a Pause Control Frame," on page 59 . When this bit is '0,' asserting the <code>clientemacpausereq</code> signal has no effect.
31	N/A	Reserved

The register contents for the Management Configuration Word are described in [Table 8-7](#).

Table 8-7: Management Configuration Word

Bits	Default Value	Description
5-0	All 0's	Clock Divide[5:0] See "Accessing MDIO via the TEMAC," on page 92 .
6	0	MDIO Enable When this bit is '1,' the MDIO interface can be used to access attached PHY devices. When this bit is '0,' the MDIO interface is disabled and the MDIO signals remain inactive. A write to this bit will only take effect if Clock Divide is set to a non-zero value.
31-7	N/A	Reserved

The register contents for the MAC Speed Configuration Word are described in [Table 8-8](#).

Table 8-8: MAC Speed Configuration Word

Bits	Default Value	Description
29-0	N/A	Reserved
31-30	"10"	MAC Speed Configuration "00" - 10 Mbps "01" - 100 Mbps "10" - 1 Gbps

Note: The setting of the MAC Speed Configuration register is not affected by a reset.

The register contents for the two unicast address registers are described in [Table 8-9](#) and [Table 8-10](#).

Table 8-9: Unicast Address (Word 0)

Bits	Default Value	Description
31-0	tieemacunicastaddr[31 downto 0]	Address filter unicast address[31:0]. This address is used by the MAC to match against the destination address of any incoming frames. The address is ordered so the first byte transmitted/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 8-10: Unicast Address (Word 1)

Bits	Default Value	Description
15-0	tieemacunicastaddr[47 downto 32]	Address filter unicast address[47:32]. See description in Table 8-9 .
31-16	N/A	Reserved

In addition to the unicast address, broadcast address and pause addresses, the address filter can be programmed to respond to 4 separate addresses. These are stored in an address table in the address filter. See “[Address Filter](#),” on page 44. [Table 8-11](#) and [Table 8-12](#) show how the contents of the table are set.

Table 8-11: Address Table Configuration (Word 0)

Bits	Default Value	Description
31-0	All 0s	MAC Address[31:0]. The address that is to be written to the address table. The address is ordered so the first byte transmitted/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 8-12: Address Table Configuration (Word 1)

Bits	Default Value	Description
15-0	All 0s	MAC Address[47:32] See description in Table 8-11 .
17-16	All 0s	The location in the address table that the MAC address is to be written to or read from. There are up to 4 entries in the table (Location 0 to 3).
22-18	N/A	Reserved
23	0	Read not write This bit is set to '1' to read from the address table. If it is set to '1,' the contents of the table entry that is being accessed by bits 17-16 will be output on the hostrddata bus in consecutive cycles (Least Significant Word first). If it is set to '0,' the data on bits 15-0 is written into the table at the address specified by bits 17-16.
31-24	N/A	Reserved

The contents of the address filter mode register are described in [Table 8-13](#).

Table 8-13: Address Filter Mode

Bits	Default Value	Description
31	1	Promiscuous Mode If this bit is set to '1,' the address filter is set to operate in promiscuous mode. All frames will be passed to the receiver client regardless of the destination address.
30-0	N/A	Reserved

Using the Management Interface

Accessing Configuration

Writing to the configuration registers through the Management Interface is depicted in [Figure 8-1](#). When accessing the configuration registers (for example, when `hostaddr[9] = '1'` and `hostmiimsel = '0'`), the upper bit of `hostopcode` functions as an Active Low write enable signal. The lower `hostopcode` bit is a *don't care* bit.

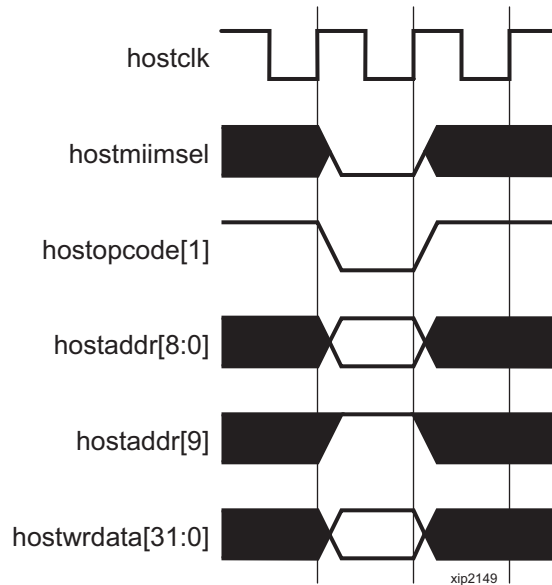


Figure 8-1: Configuration Register Write Timing

Reading from the configuration register words is similar, but the upper `hostopcode` bit should be '1,' as shown in [Figure 8-2](#). In this case, the contents of the register appear on `hostrddata` the `hostclk` edge after the register address is asserted onto `hostaddr`.

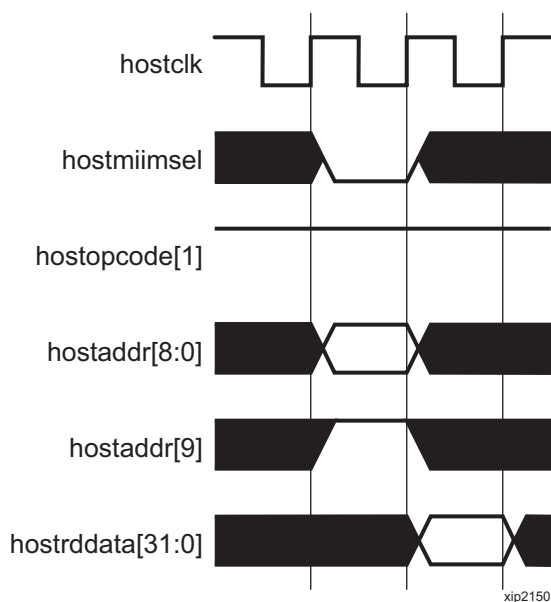


Figure 8-2: Configuration Register Read Timing

Accessing the Address Table

Writing and reading to the address table configuration words is less straightforward. To write to a specific entry in the table, the user must first write the least significant 32-bits of the address into the address table configuration (Word 0) register. The user then writes the most significant 16-bits together with the location in the table (bits 17-16) to the address table configuration (Word 1) register with bit 23 (read not write) set to '0.' This is shown in

Figure 8-3. Although it is shown in the figure, there is no requirement for the two writes to be on adjacent cycles.

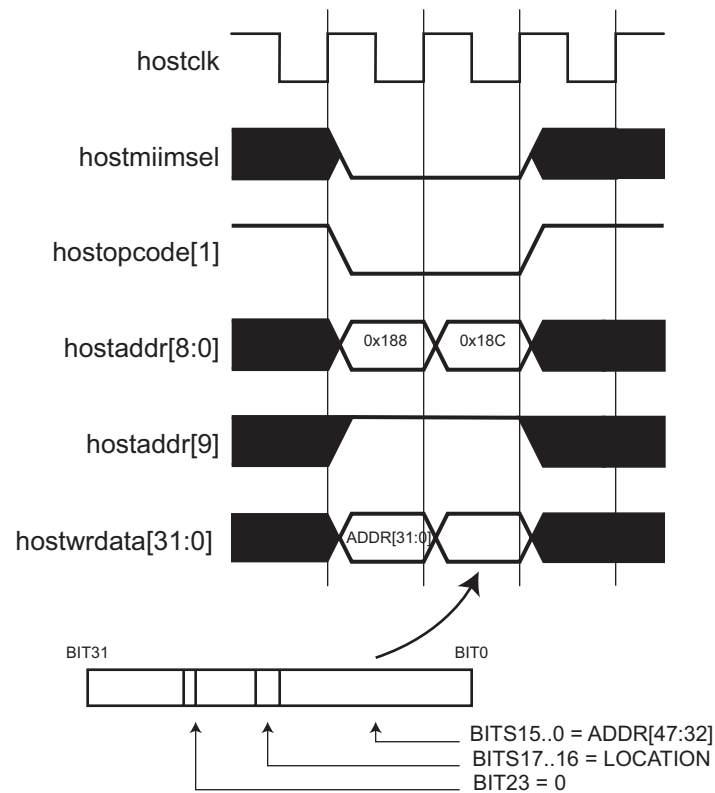


Figure 8-3: Address Table Write Timing

To read from the address table the user writes to the address table configuration register (Word 1) with the location set to the desired table entry and bit 23 set to '1.' On the next cycle the least significant word appears on the `hostwrddata` bus. One cycle afterwards the

most significant 16-bits are output on the lower 16 bits of the bus. This is shown in Figure 8-4.

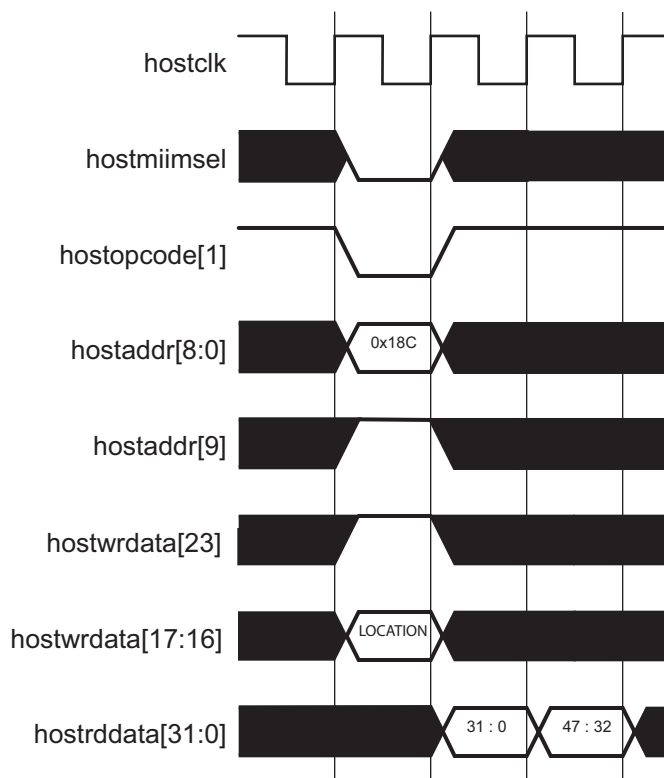


Figure 8-4: Address Table Read Timing

MDIO Interface

Introduction to MDIO

The MDIO interface for 1 Gbps operation and slower speeds is defined in *IEEE 802.3* clause 22. This is a two wire interface consisting of a clock, *mdc*, and a shared serial data line, *mdio*. This interface is typically connected to the MDIO ports of a physical layer device (PHY) to access its configuration and status registers.

There are two different transaction types of MDIO for write and read; they are described in the next sections. The following abbreviations apply for the remainder of this chapter:

- **PRE** - preamble
- **ST** - start of frame
- **OP** - operation code
- **PHYAD** - PHY address
- **REGAD** - Register address
- **TA** - turnaround.

Write Transaction

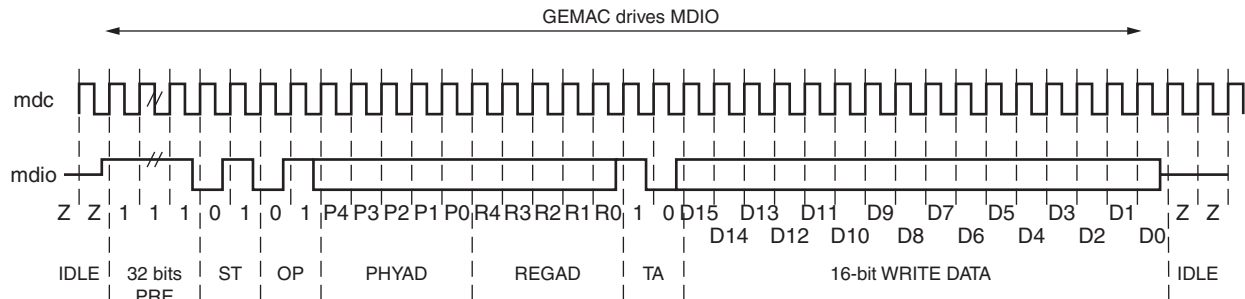


Figure 8-5: MDIO Write Transaction

Figure 8-5 shows a Write transaction across the MDIO; this is defined by OP="01". The addressed PHY (PHYAD) device takes the 16-bit word in the data field and writes it to the register at REGAD.

Read Transaction

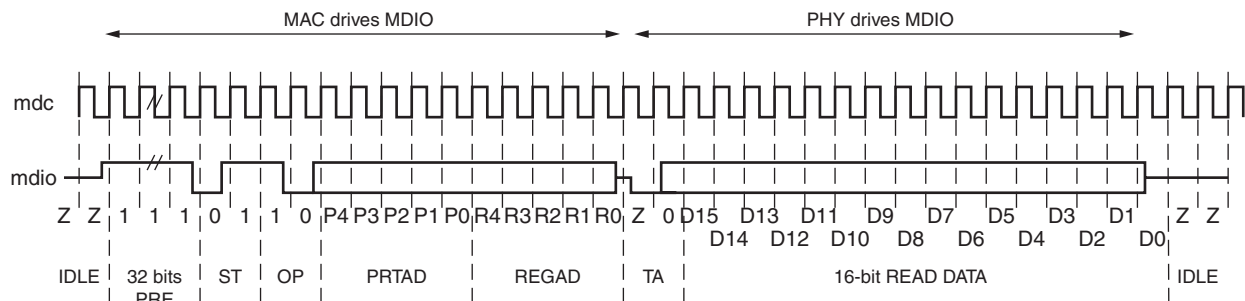


Figure 8-6: MDIO Read Transaction

Figure 8-6 shows a Read transaction; this is defined by OP="10". The addressed PHY (PHYAD) device returns the 16-bit word from the register at REGAD.

For details of the register map of PHY layer devices and a fuller description of the operation of the MDIO Interface itself, see *IEEE 802.3-2002*.

Accessing MDIO via the TEMAC

The Management Interface is also used to access the MDIO interface of the MAC core. The MDIO interface supplies a clock to the connected PHY, *mdc*. This clock is derived from the *hostclk* signal using the value in the *Clock Divide[4:0]* configuration register. The frequency of *mdc* is given by the following equation:

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

The frequency of *mdc* given by this equation should not exceed 2.5 MHz in order to comply with the *IEEE 802.3-2002* specification for this interface. 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.

For details of the register map of PHY layer devices and a fuller description of the operation of the MDIO interface itself, see *IEEE 802.3-2002*.

Access to the MDIO interface through the Management Interface is depicted in the timing diagram in Figure 8-7.

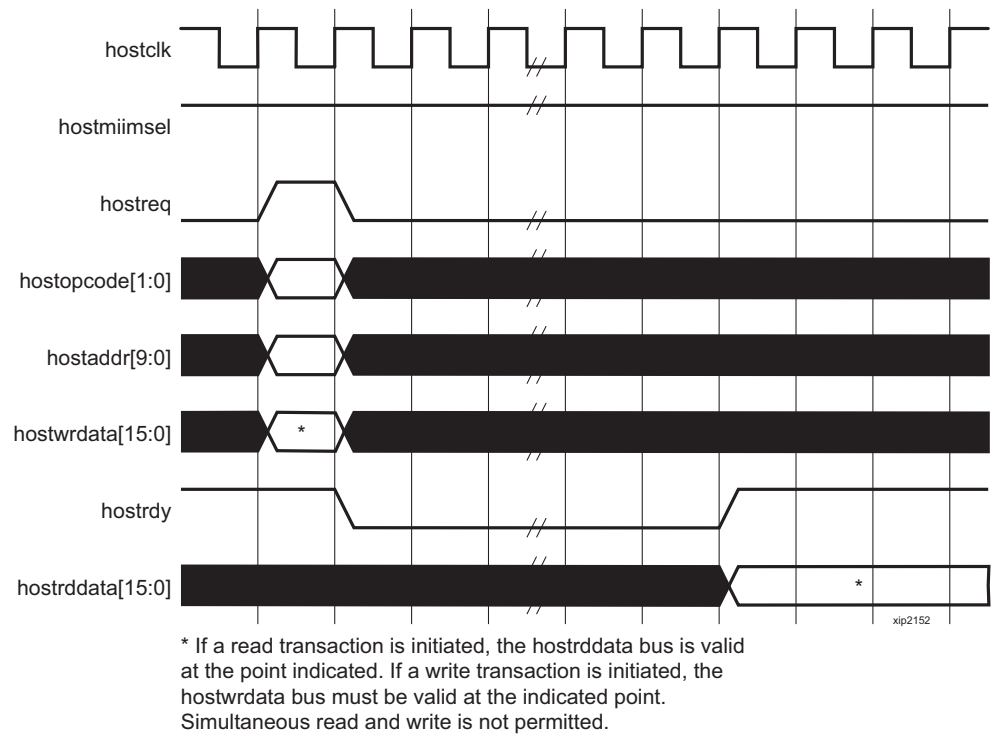


Figure 8-7: MDIO Access Through Management Interface

For MDIO transactions, the following points apply:

- hostmiimsel is '1'
- hostopcode[1:0] maps to the OP (opcode) field of the MDIO frame
- hostaddr maps to the two address fields of the MDIO frame; PHYAD is hostaddr[9:5], and REGAD is host_addr[4:0]
- hostwrdata[15:0] maps into the data field of the MDIO frame when performing a write operation
- The data field of the MDIO frame maps into hostrddata[15:0] when performing a read operation

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

The MAC core then deasserts the hostmiimrdy signal while the transaction across the MDIO is in progress. When the transaction across the MDIO interface has been completed, the hostmiimrdy signal will be asserted by the MAC core; if the transaction is a read, the data will also be available on the hostrddata[15:0] bus at this time.

For the TEMAC port definition of the MDIO, see “Optional MDIO Signals,” on page 32. In addition, see the following sections:

- “Connecting the MDIO to an Internally Integrated PHY” on page 77
- “Connecting the MDIO to an External PHY” on page 78
- “Connecting the MDIO to an External and Internal PHY” on page 78

Accessing Configuration without the Management Interface

If the optional Management Interface is omitted from the core, all of relevant configuration signals are brought out of the core. These signals are bundled into the `tiemacconfigvec` signal. The bit mapping of the configuration vector signal is defined in Table 8-14. See the corresponding entry in the configuration register tables for the full description of each signal.

These configuration vector signals can be changed by the user at any time; however, with the exception of the reset and the flow control configuration signals, they will not take effect until the current frame has completed transmission or reception.

The Clock heading denotes which clock domain the configuration signal is registered into before use by the core. It is not necessary to drive the signal from this clock domain.

Configuration Vector Description

Table 8-14: Configuration Vector Bit Definition

Bit(s)	Configuration Register cross reference	Clock	Description
47 ... 0	“Receiver Configuration Word 0” bits 31-0 and “Receiver Configuration Word 1” bits 15-0	rxcoreclk (rxgmiimiiclk if clock_enables = true)	Pause frame MAC Source Address[47:0] This address is 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. 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 bite [47:0] as 0xFFEEDDCBBAA.
48	“Receiver Configuration Word 1” bit 26	rxcoreclk (rxgmiimiiclk if clock_enables = true)	Receiver Half Duplex If ‘1,’ the receiver operates in half-duplex mode. If ‘0,’ the receiver operates in full-duplex mode.
49	“Receiver Configuration Word 1” bit 27	rxcoreclk (rxgmiimiiclk if clock_enables = true)	Receiver VLAN Enable When this bit is set to ‘1,’ VLAN tagged frames are accepted by the receiver.
50	“Receiver Configuration Word 1” bit 28	rxcoreclk (rxgmiimiiclk if clock_enables = true)	Receiver Enable If set to ‘1,’ the receiver block is operational. If set to ‘0,’ the block ignores activity on the physical interface RX port.

Table 8-14: Configuration Vector Bit Definition (Continued)

Bit(s)	Configuration Register cross reference	Clock	Description
51	“Receiver Configuration Word 1” bit 29	rxcoreclk (rxgmiimiiclk if clock_enables = true)	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 “Client-Supplied FCS Passing,” on page 42 . 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.
52	“Receiver Configuration Word 1” bit 30	rxcoreclk (rxgmiimiiclk if clock_enables = true)	Receiver Jumbo Frame Enable When this bit is ‘0,’ the receiver will not pass frames longer than the maximum legal frame size specified in <i>IEEE 802.3-2002</i> (“Maximum Permitted Frame Length,” on page 53). When it is ‘1,’ the receiver will not have an upper limit on frame size.
53	“Receiver Configuration Word 1” bit 31	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.
54	“Transmitter Configuration Word” bit 25	txcoreclk (txgmiimiiclk if clock_enables = true)	Transmitter Interframe Gap Adjust Enable If ‘1,’ and the MAC is set to operate in full-duplex mode, then the transmitter will read the value of the clientemactxifgdelay port and set the Interframe Gap accordingly. If ‘0,’ the transmitter will always insert at least the legal minimum interframe gap.
55	“Transmitter Configuration Word” bit 26	txcoreclk (txgmiimiiclk if clock_enables = true)	Transmitter Half Duplex If ‘1,’ the transmitter operates in half-duplex mode. If ‘0,’ the transmitter operates in full-duplex mode.

Table 8-14: Configuration Vector Bit Definition (*Continued*)

Bit(s)	Configuration Register cross reference	Clock	Description
56	“Transmitter Configuration Word” bit 27	txcoreclk (txgmiimiiclk if clock_enables = true)	Transmitter VLAN Enable When this bit is set to ‘1,’ the transmitter allows the transmission of VLAN tagged frames.
57	“Transmitter Configuration Word” bit 28	txcoreclk (txgmiimiiclk if clock_enables = true)	Transmitter Enable When this bit is ‘1,’ the transmitter will be operational. When it is ‘0,’ the transmitter is disabled.
58	“Transmitter Configuration Word” bit 29	txcoreclk (txgmiimiiclk if clock_enables = true)	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 “Client-Supplied FCS Passing,” on page 50. When it is ‘0,’ the MAC transmitter will append padding as required, compute the FCS and append it to the frame.
59	“Transmitter Configuration Word” bit 30	txcoreclk (txgmiimiiclk if clock_enables = true)	Transmitter Jumbo Frame Enable When this bit is ‘1,’ the MAC transmitter will allow frames larger than the maximum legal frame length specified in <i>IEEE 802.3-2002</i> to be sent. When set to ‘0,’ the MAC transmitter will only allow frames up to the legal maximum to be sent.
60	“Transmitter Configuration Word” bit 31	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.

Table 8-14: Configuration Vector Bit Definition (Continued)

Bit(s)	Configuration Register cross reference	Clock	Description
61	“Flow Control Configuration Word” bit 29	txcoreclk (txgmiimiiclk if clock_enables = true)	Transmit Flow Control Enable When this bit is ‘1,’ asserting the clientemacpausereq signal causes the MAC core to send a flow control frame out from the transmitter as described in “Transmitting a Pause Control Frame,” on page 59. When this bit is ‘0,’ asserting the clientemacpausereq signal will have no effect.
62	“Flow Control Configuration Word” bit 30	rxcoreclk (rxgmiimiiclk if clock_enables = true)	Receive Flow Control Enable When this bit is ‘1,’ received flow control frames will inhibit the transmitter operation as described in “Receiving a Pause Control Frame,” on page 60. When it is ‘0,’ received flow frames are passed up to the client.
63	“Receiver Configuration Word 1” bit 25	rxcoreclk (rxgmiimiiclk if clock_enables = true)	Length/Type Error Check Disable When this bit is ‘1,’ the core will not perform the length/type field error checks as described in “Length/Type Field Error Checks,” on page 44. When it is set to ‘0,’ the length/type field checks will be performed; this is normal operation.

Table 8-14: Configuration Vector Bit Definition (*Continued*)

Bit(s)	Configuration Register cross reference	Clock	Description
64	“Address Filter Mode” bit 31	rxcoreclk (rxgmiimiiclk if clock_enables = true)	Address Filter Enable When this bit is ‘0,’ the address filter is enabled. If it is set to ‘1,’ the address filter will operate in promiscuous mode.
66 downto 65	“MAC Speed Configuration Word” bits 31 downto 30	txcoreclk and rxcoreclk (txgmiimiiclk and rxgmiimiiclk if clock_enables = true)	MAC Speed

Constraining the Core

This chapter defines the constraint requirements of the TEMAC core. An example UCF is provided with the HDL example design to provide samples of constraint requirements for the design. See the *Tri-Mode Ethernet MAC Getting Started Guide* for more information.

Required Constraints

Device, Package, and Speedgrade Selection

The TEMAC can be implemented in Virtex-II, Virtex-II Pro, Spartan-3, Spartan-3E, Spartan-3A, Virtex-4, and Virtex-5 devices with the following attributes:

- Large enough to accommodate the core
- Contains a sufficient number of IOBs
- -4 speed grade for Virtex-II, -5 speed grade for Virtex-II Pro, Spartan-3, Spartan-3E and Spartan-3A, -10 speed grade for Virtex-4, and -1 speed grade for Virtex-5

I/O Location Constraints

No specific I/O location constraints are required.

Placement Constraints

With the exception of Virtex-4 and Virtex-5, the constraints file contains placement information for the global clock buffers. These are provided as an example only and may be removed. However, in RGMII, it is recommended that all the transmitter clock buffers are confined to the same bank of BUFGs.

Timing Constraints

Example(s) are in the UCF delivered with the HDL example design for the core.

PERIOD(s) for Clock nets

GMII Clock Constraints

If an external GMII interface is implemented then the following constraints should be applied.

gmii_rx_clk

The gmii_rx_clk signal is connected to the rxgmiiimiclk input of the TEMAC in the example design that is provided with the core. In order for the core to operate correctly at 1 Gbps, this must be constrained to run at 125 MHz.

```
NET "gmii_rx_clk*" TNM_NET = "clk_rx";
TIMEGRP "rx_clock" = "clk_rx";
TIMESPEC "TS_rx_clk" = PERIOD "rx_clock" 8000 ps HIGH 50 %;
```

rx_clk_int

If the clock_enables option is set to false, the rx_clk_int signal must be constrained to run at 125 MHz for 1 Gbps operation. This is connected to the rxcoreclk input of the TEMAC, in addition to driving the user receive logic. If the clock_enables option is set to true, the constraint should not be present.

```
NET "rx_clk_int" TNM_NET = "clk_rx_core";
TIMEGRP "rx_clock_core" = "clk_rx_core";
TIMESPEC "TS_rx_clk_core" = PERIOD "rx_clock_core" 8000 ps HIGH 50 %;
```

tx_gmii_mii_clk

The tx_gmii_mii_clk signal is connected to the txgmiiimiclk input of the TEMAC. This signal must be constrained for a frequency of 125 MHz for 1 Gbps operation.

```
NET "tx_gmii_mii_clk*" TNM_NET = "clk_tx_gmii";
TIMEGRP "tx_clock_gmii" = "clk_tx_gmii";
TIMESPEC "TS_tx_clk_gmii" = PERIOD "tx_clock_gmii" 8000 ps HIGH 50 %;
```

tx_clk_int

If the clock_enables option is set to false, the tx_clk_int signal drives the txcoreclk input of the core and the users transmit logic. This signal must be constrained to run at 125 MHz for 1 Gbps operation. If the clock_enables option is set to true, the constraint should not be present.

```
NET "tx_clk_int" TNM_NET = "clk_tx_core";
TIMEGRP "tx_clock_core" = "clk_tx_core";
TIMESPEC "TS_tx_clk_core" = PERIOD "tx_clock_core" 8000 ps HIGH 50 %;
```

RGMII Clock Constraints

If an external RGMII interface is implemented, the following constraints should be applied.

rx_rgmii_clk_int

The rx_rgmii_clk_int signal is connected to the rxgmiiimiclk input of the TEMAC in the example design that is provided with the core. In order for the core to operate correctly at 1 Gbps, this must be constrained to run at 125 MHz.

```
NET "rx_rgmii_clk_int" TNM_NET = "clk_rx";
TIMEGRP "rx_clock" = "clk_rx";
TIMESPEC "TS_rx_clk" = PERIOD "rx_clock" 8000 ps HIGH 50 %;
```

rx_clk_int

If the clock_enables option is set to false, the rx_clk_int signal must be constrained to run at 125 MHz for 1 Gbps operation. This is connected to the rxcoreclk input of the TEMAC, in addition to driving the user receive logic. If the clock_enables option is set to true, the constraint should not be present.

```
NET "rx_clk_int" TNM_NET = "clk_rx_core";
TIMEGRP "rx_clock_core" = "clk_rx_core";
TIMESPEC "TS_rx_clk_core" = PERIOD "rx_clock_core" 8000 ps HIGH 50 %;
```

tx_clk180

The tx_clk180 signal is used to generate the transmit clocks at 10 Mbps and 100 Mbps for the RGMII interface. This should be constrained to run at 125 MHz.

Note: This period constraint does not exist for Virtex-5 devices, as this clock is not generated.

```
NET **tx_clk180" TNM_NET = "clk_tx";
TIMEGRP "tx_clock" = "clk_tx";
TIMESPEC "TS_tx_clk" = PERIOD "tx_clock" 8000 ps HIGH 50 %;
```

tx_rgmii_clk

The tx_rgmii_clk signal is connected to the txgmiiimiclk input of the TEMAC. This signal must be constrained for a frequency of 125 MHz for 1 Gbps operation.

```
NET "tx_rgmii_clk" TNM_NET = "clk_tx_gmii";
TIMEGRP "tx_clock_gmii" = "clk_tx_gmii";
TIMESPEC "TS_tx_clk_gmii" = PERIOD "tx_clock_gmii" 8000 ps HIGH 50 %;
```

tx_clk_int

If the clock_enables option is set to false, the tx_clk_int signal drives the txcoreclk input of the core and the users transmit logic. This signal must be constrained to run at 125 MHz for 1 Gbps operation. If the clock_enables option is set to true, the constraint should not be present.

```
NET "tx_clk_int" TNM_NET = "clk_tx_core";
TIMEGRP "tx_clock_core" = "clk_tx_core";
TIMESPEC "TS_tx_clk_core" = PERIOD "tx_clock_core" 8000 ps HIGH 50 %;
```

refclk_bufg

For Virtex-4 and Virtex-5 devices, an additional constraint is provided in the UCF for the IDELAYCTRL reference clock. This clock is constrained to run at 200 MHz, but may be relaxed for Virtex-5 devices within the guidelines described in the *Virtex-5 User Guide* for IDELAYCTRL components.

```
NET **refclk_bufg" TNM_NET = "clk_ref_clk";
TIMEGRP "ref_clk" = "clk_ref_clk";
TIMESPEC "TS_ref_clk" = PERIOD "ref_clk" 5000 ps HIGH 50 %;
```

Management Clock Constraints

host_clk

The host_clk signal must be constrained to run at the desired frequency.

```
NET "host_clk" TNM_NET = "host_clk";
TIMEGRP "host" = "host_clk" EXCEPT "mdio_logic";
TIMESPEC "TS_host_clk" = PERIOD "host" 10000 ps HIGH 50 %;
```

MDIO Logic

The MDIO logic is driven from the MDC clock. This is output from the core as emacphymclkout. The following constraints must be applied for the MDIO logic to operate correctly.

```
INST "trimac_core?BU2?U0?TRIMAC_INST?MANIFGEN?MANAGEN?PHY?ENABLE_REG" TNM = "mdc_rising";
INST "trimac_core?BU2?U0?TRIMAC_INST?MANIFGEN?MANAGEN?PHY?READY_INT" TNM = "mdc_rising";
INST "trimac_core?BU2?U0?TRIMAC_INST?MANIFGEN?MANAGEN?PHY?STATE_COUNT*" TNM = "mdc_rising";
INST "trimac_core?BU2?U0?TRIMAC_INST?MANIFGEN?MANAGEN?PHY?MDIO_TRISTATE" TNM = "mdc_falling";
INST "trimac_core?BU2?U0?TRIMAC_INST?MANIFGEN?MANAGEN?PHY?MDIO_OUT" TNM = "mdc_falling";
TIMEGRP "mdio_logic" = "mdc_rising" "mdc_falling";

TIMESPEC "TS_mdio1" = PERIOD "mdio_logic" 400 ns;
TIMESPEC "TS_mdio2" = FROM "mdc_rising" TO "mdc_falling" 200 ns;
```

Timespecs for Critical Logic

Signals must cross clock domains at certain points in the core. These are described in the following section.

Configuration Logic

When the optional Management Interface is used with the core (see “Using the Optional Management Interface,” on page 81), configuration information is written synchronously to hostclk. Receiver configuration data must be transferred onto the rxcoreclk clock domain for use with the receiver; transmitter configuration data must be transferred onto the txcoreclk domain for use with the transmitter. The following UCF syntax targets this logic and a timing ignore attribute (TIG) is applied. It does not matter when configuration changes take place; the current configurations are sampled between frames by both the receiver and transmitter.

```
INST "trimac_core?BU2?U0?TRIMAC_INST?MANIFGEN?MANAGEN?CONF?RX0_OUT*" TNM="config_to_rx";
INST "trimac_core?BU2?U0?TRIMAC_INST?MANIFGEN?MANAGEN?CONF?RX1_OUT*" TNM="config_to_rx";
INST "trimac_core?BU2?U0?TRIMAC_INST?MANIFGEN?MANAGEN?CONF?FC_OUT_29" TNM="config_to_rx";
TIMESPEC "TS_config_to_rx" = FROM "config_to_rx" TO "rx_clock" TIG;

INST "trimac_core?BU2?U0?TRIMAC_INST?MANIFGEN?MANAGEN?CONF?TX_OUT*" TNM="config_to_tx";
INST "trimac_core?BU2?U0?TRIMAC_INST?MANIFGEN?MANAGEN?CONF?FC_OUT_30" TNM="config_to_tx";
TIMESPEC "TS_config_to_tx" = FROM "config_to_tx" TO "tx_clock_gmii" TIG;
```

Timespecs for Reset Logic within the Core

Internally, the core is divided up into clock/reset domains, which group together elements with common clock and reset signals. The reset circuit provides controllable skews on the reset nets within the design. More information on the operation and rationale behind this circuit can be found in Ken Chapman’s Xilinx TechXclusive, “Get Smart About Reset” at:

www.xilinx.com/support/techxclusives/global-techX19.htm

The following UCF syntax identifies the relevant reset logic and ensures that the reset signals do not cause set-up or hold violations in the circuit:

```
NET "trimac_core?BU2?U0?TRIMAC_INST?RXRSTGENNOEN?SYNC_RX_RESET_I?RESET_OUT*" MAXDELAY=6100 ps;
NET "trimac_core?BU2?U0?TRIMAC_INST?TXRSTGENNOEN?SYNC_TX_RESET_I?RESET_OUT*" MAXDELAY=6100 ps;
NET "trimac_core?BU2?U0?TRIMAC_INST?INT_GMII_MII_RX_RESET" MAXDELAY=6100 ps;
NET "trimac_core?BU2?U0?TRIMAC_INST?RXGMIIIRSTGENEN?SYNC_GMII_MII_RX_RESET_I?RESET_OUT*" MAXDELAY=6100 ps;
NET "trimac_core?BU2?U0?TRIMAC_INST?SYNC_GMII_MII_TX_RESET_I?RESET_OUT*" MAXDELAY=6100 ps;
NET "trimac_core?BU2?U0?TRIMAC_INST?G_SYNC_MGMT_RESET?SYNC_MGMT_RESET_HOST_I?RESET_OUT*" MAXDELAY=6100 ps;
```

Note: The lastline is only required when the optional Management Interface is used.

Note: The first three lines are only required when the clock enables option is not selected.

Constraints when Implementing an External GMII

The constraints defined in this section are implemented in the UCF for the example design delivered with the core. Sections from this UCF are copied into the following descriptions to act as an example. These should be studied in conjunction with the HDL source code for the example design and with the description given in “Implementing External GMII,” on page 63.

GMII IOB Constraints

The following constraints target the flip-flops that are inferred in the top-level HDL file for the example design; constraints are set to ensure that these are placed in IOBs.

```
INST "*gmii_txd_reg*" IOB = true;
```



```

INST "gmii_tx_en_reg"    IOB = true;
INST "gmii_tx_er_reg"    IOB = true;

INST "rx_dv_to_mac*"     IOB = true;
INST "rx_dv_to_mac"      IOB = true;
INST "rx_er_to_mac"      IOB = true;

```

The GMII is a 3.3 volt signal level interface. The 3.3 volt LVTTL SelectIO standard is the default for Virtex-II devices; the following constraints may be added without harm. The 3.3 volt LVTTL SelectIO standard is not the default for Virtex-5, Virtex-4, Virtex-II Pro, Spartan-3, Spartan-3E and Spartan-3A devices. Use the following constraints with the device IO Banking rules:

```

INST "gmii_txd<?>"      IOSTANDARD = LVTTL;
INST "gmii_tx_en"        IOSTANDARD = LVTTL;
INST "gmii_tx_er"        IOSTANDARD = LVTTL;

INST "gmii_rxd<?>"      IOSTANDARD = LVTTL;
INST "gmii_rx_dv"        IOSTANDARD = LVTTL;
INST "gmii_rx_er"        IOSTANDARD = LVTTL;

INST "gmii_tx_clk"       IOSTANDARD = LVTTL;
INST "gmii_rx_clk"       IOSTANDARD = LVTTL;

```

In addition, the example design provides pad locking on the GMII for several families. This is provided as a guideline only; there are no specific I/O location constraints for this core.

GMII Input Setup/Hold Timing

Figure 9-1 and Table 9-1 illustrate the setup and hold time window for the input GMII signals. This is the worst-case data valid window presented to the FPGA device pins.

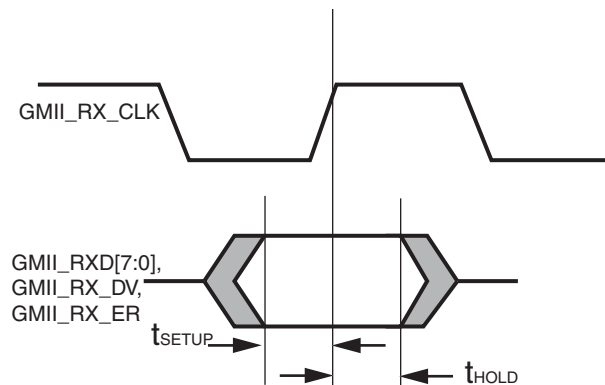


Figure 9-1: Input GMII Timing

Observe that there is a 2 ns data valid window which is presented across the GMII input bus. This must be correctly sampled by the FPGA devices.

Table 9-1: Input GMII Timing

Symbol	Min	Max	Units
t_{SETUP}	2.00	-	ns
t_{HOLD}	0.00	-	ns

Virtex-II, and Virtex-II Pro Devices

These families have input delay elements (which are always of a fixed delay) that are automatically inserted by the Xilinx tools and are set to provide a zero-hold time. These

input delays will automatically meet input setup and hold timing on the GMII without any specific constraints.

Spartan-3, Spartan-3E, and Spartan-3A Devices

The GMII design uses a DCM on the receiver clock domain for Spartan-3, Spartan-3E and Spartan-3A devices. Phase-shifting is then applied to the DCM to align the resultant clock so that it will correctly sample the 2 ns GMII data valid window at the input flip-flops.

The fixed phase shift is applied to the DCM using the following UCF syntax.

```
INST *gmii_rxc_dcm CLKOUT_PHASE_SHIFT = FIXED;
INST *gmii_rxc_dcm PHASE_SHIFT = -30;
```

The value of `PHASE_SHIFT` is preconfigured in the example designs to meet the setup and hold constraints for the example GMII pinout in the particular device. The setup/hold timing which is achieved after place-and-route is reported in the data sheet section of the TRCE report (created by the implement script). A further explanation of these numbers is detailed in [“Understanding Timing Reports for GMII Setup/Hold timing”](#).

For customers fixing their own pinout, the setup and hold figures reported in the TRCE report can be used to initially setup the approximate DCM phase shift. [Appendix D, “Calculating the DCM Phase Shift”](#) describes a more accurate method for fixing the phase shift by using hardware measurement of a unique PCB design.

Virtex-4 Devices

The GMII design uses IDELAY components on the receiver clock, data and control signals for Virtex-4 devices. A fixed tap delay can be applied to either delay the data and control signals or delay the clock so that the data/control are correctly sampled by the `gmii_rx_clk` clock at the IOB flip-flop, meeting GMII setup and hold timing.

The choice of delaying data/control or clock is dependant upon a number of factors, not least being the required shift. There are trade-offs to be made with either choice: Delaying the clock is clock period specific as we move the clock to line up each edge with data from the following edge. Delaying the data/control introduces more jitter which degrades the overall setup/hold window. The interface timing report in the two cases is also quite different and for this reason this is discussed in [“Understanding Timing Reports for GMII Setup/Hold timing”](#).

The following constraint shows an example of setting the delay value for one of these IDELAY components. All bits can be adjusted individually, if desired, to compensate for any PCB routing skew.

```
INST *gmii_interface/delay_gmii_rx_dv IOBDELAY_VALUE = 53;
```

The value of `IOBDELAY_VALUE` is preconfigured in the example designs to meet the setup and hold constraints for the example GMII pinout in the particular device. The setup/hold timing which is achieved after place-and-route is reported in the data sheet section of the TRCE report (created by the implement script).

When IDELAY or IODELAY primitives are instantiated with a fixed delay attribute, an IDELAYCTRL component must be also instantiated to continuously calibrate the individual input delay elements. The IDELAYCTRL module requires a reference clock, which is assumed to be an input to the example design delivered by CORE Generator. The most efficient way to use the IDELAYCTRL module is to lock the placement of the instance to the clock region of the device where the IDELAY/IODELAY components are placed. An example LOC constraint for the IDELAYCTRL module is shown commented out in the UCF. See the *Virtex-4 User Guide* and code comments for more information.

Virtex-5 devices

The GMII design uses IODELAY components on the receiver clock, data and control signals for Virtex-5 devices. A fixed tap delay can be applied to either delay the data and control signals or delay the clock so that the data/control are correctly sampled by the `gmii_rx_clk` clock at the IOB flip-flop, meeting GMII setup and hold timing.

The choice of delaying data/control or clock is dependant upon a number of factors, not least being the required shift. There are trade-offs to be made with either choice: Delaying the clock is clock period specific as we move the clock to line up each edge with data from the following edge. Delaying the data/control introduces more jitter which degrades the overall setup/hold window. The interface timing report in the two cases is also quite different and for this reason this is discussed in [“Understanding Timing Reports for GMII Setup/Hold timing”](#).

The following constraint shows an example of setting the delay value for one of these IODELAY components. All bits can be adjusted individually, if desired, to compensate for any PCB routing skew.

```
INST *gmii_interface/delay_gmii_rx_dv IDELAY_VALUE = 33;
```

The value of IDELAY_VALUE is preconfigured in the example designs to meet the setup and hold constraints for the example GMII pinout in the particular device. The setup/hold timing which is achieved after place-and-route is reported in the data sheet section of the TRCE report (created by the implement script).

When IDELAY or IODELAY primitives are instantiated with a fixed delay attribute, an IDELAYCTRL component must be also instantiated to continuously calibrate the individual input delay elements. The IDELAYCTRL module requires a reference clock, which is assumed to be an input to the example design delivered by CORE Generator. The most efficient way to use the IDELAYCTRL module is to lock the placement of the instance to the clock region of the device where the IDELAY/IODELAY components are placed. An example LOC constraint for the IDELAYCTRL module is shown commented out in the UCF. See the *Virtex-5 User Guide* and code comments for more information.

Understanding Timing Reports for GMII Setup/Hold timing

Spartan-3 Devices

Setup and Hold results for the GMII input bus can be found in the data sheet section of the Timing Report.

The Clock Generation logic includes a BUFGMUX which provides a path for the pre-DCM clock, see [Figure 7-4](#). When this BUFGMUX is present the timing engine will use the non-DCM clock path resulting in false setup and hold numbers. It is therefore necessary to edit the example design HDL to remove this non-DCM path if the setup/hold numbers are to be used for DCM phase adjustment.

Once this is done the results are self-explanatory and it is easy to see how they relate to [Figure 9-1](#). Here follows an example for the GMII report from a Spartan-3A device. The implementation requires 1.965 ns of setup: this is less than the 2 ns required and so there is slack. The implementation requires -0.007 ns of hold: this is less than the 0 ns required and so there is slack.

```
Data Sheet report:
-----
All values displayed in nanoseconds (ns)
```

Setup/Hold to clock gmii_rx_clk

Source	Setup to clk (edge)	Hold to clk (edge)	Internal Clock(s)	Clock Phase
gmii_rx_dv	1.958 (R)	-0.014 (R)	rx_gmii_mii_clk_int	-2.812
gmii_rx_er	1.940 (R)	-0.007 (R)	rx_gmii_mii_clk_int	-2.812
gmii_rxd<0>	1.957 (R)	-0.012 (R)	rx_gmii_mii_clk_int	-2.812
gmii_rxd<1>	1.958 (R)	-0.014 (R)	rx_gmii_mii_clk_int	-2.812
gmii_rxd<2>	1.965 (R)	-0.022 (R)	rx_gmii_mii_clk_int	-2.812
gmii_rxd<3>	1.940 (R)	-0.007 (R)	rx_gmii_mii_clk_int	-2.812
gmii_rxd<4>	1.958 (R)	-0.014 (R)	rx_gmii_mii_clk_int	-2.812
gmii_rxd<5>	1.961 (R)	-0.017 (R)	rx_gmii_mii_clk_int	-2.812
gmii_rxd<6>	1.965 (R)	-0.022 (R)	rx_gmii_mii_clk_int	-2.812
gmii_rxd<7>	1.956 (R)	-0.011 (R)	rx_gmii_mii_clk_int	-2.812

Virtex-II or Virtex-II Pro Devices

Setup and Hold results for the GMII input bus can be found in the data sheet section of the Timing Report. The results are self-explanatory and it is easy to see how they relate to [Figure 9-1](#). Here follows an example for the GMII report from a Virtex-II device. The implementation requires 1.835 ns of setup: this is less than the 2 ns required and so there is slack. The implementation requires -0.226 ns of hold: this is less than the 0 ns required and so there is slack.

Data Sheet report:

All values displayed in nanoseconds (ns)

Setup/Hold to clock gmii_rx_clk

Source	Setup to clk (edge)	Hold to clk (edge)	Internal Clock(s)	Clock Phase
gmii_rx_dv	1.820 (R)	-0.281 (R)	rx_gmii_mii_clk_int	0.000
gmii_rx_er	1.770 (R)	-0.226 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<0>	1.821 (R)	-0.283 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<1>	1.833 (R)	-0.295 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<2>	1.790 (R)	-0.253 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<3>	1.789 (R)	-0.252 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<4>	1.834 (R)	-0.296 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<5>	1.829 (R)	-0.291 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<6>	1.793 (R)	-0.255 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<7>	1.835 (R)	-0.296 (R)	rx_gmii_mii_clk_int	0.000

Virtex-4 or Virtex-5 Devices with Delayed Data/Control

Setup and Hold results for the GMII input bus can be found in the data sheet section of the Timing Report. The results are self-explanatory and it is easy to see how they relate to [Figure 9-1](#). Here follows an example for the GMII report from a Virtex-5 device. The implementation requires 1.962 ns of setup: this is less than the 2 ns required and so there is slack. The implementation requires -0.008 ns of hold: this is less than the 0 ns required and so there is slack.

Data Sheet report:

All values displayed in nanoseconds (ns)

Setup/Hold to clock gmii_rx_clk

Source	Setup to clk (edge)	Hold to clk (edge)	Internal Clock(s)	Clock Phase
gmii_rx_dv	1.955 (R)	-0.017 (R)	rx_gmii_mii_clk_int	0.000
gmii_rx_er	1.962 (R)	-0.031 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<0>	1.949 (R)	-0.013 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<1>	1.944 (R)	-0.009 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<2>	1.947 (R)	-0.012 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<3>	1.942 (R)	-0.008 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<4>	1.950 (R)	-0.015 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<5>	1.962 (R)	-0.026 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<6>	1.957 (R)	-0.022 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<7>	1.952 (R)	-0.020 (R)	rx_gmii_mii_clk_int	0.000

Virtex-4 or Virtex-5 Devices with Delayed Clock

Setup and Hold results for the GMII input bus can be found in the data sheet section of the Timing Report. However, depending on how the setup/hold requirements have been met the results can initially look strange and it is not immediately obvious how they relate to [Figure 9-1](#). Here follows an example for the GMII report from a Virtex-4 device where the clock has been delayed to meet the setup/hold requirements.

Data Sheet report:

All values displayed in nanoseconds (ns)

Setup/Hold to clock gmii_rx_clk

Source	Setup to clk (edge)	Hold to clk (edge)	Internal Clock(s)	Clock Phase
gmii_rx_dv	-6.198 (R)	7.526 (R)	rx_gmii_mii_clk_int	0.000
gmii_rx_er	-6.225 (R)	7.554 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<0>	-6.149 (R)	7.484 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<1>	-6.152 (R)	7.486 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<2>	-6.206 (R)	7.532 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<3>	-6.207 (R)	7.533 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<4>	-6.134 (R)	7.476 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<5>	-6.134 (R)	7.476 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<6>	-6.170 (R)	7.506 (R)	rx_gmii_mii_clk_int	0.000
gmii_rxd<7>	-6.170 (R)	7.506 (R)	rx_gmii_mii_clk_int	0.000

The implementation requires -6.134 ns of setup. [Figure 9-2](#) illustrates that this represents a figure of 1.866 ns relative to the following rising edge of the clock (since the IDELAY has acted to delay the clock by an entire period when measured from the input flip-flop). This is less than the 2 ns required and so there is slack.

The implementation requires 7.554 ns of hold. [Figure 9-2](#) illustrates that this represents a figure of -0.446 ns relative to the following rising edge of the clock (since the IDELAY has

acted to delay the clock by an entire period when measured from the input flip-flop). This is less than the 0 ns required and so there is slack.

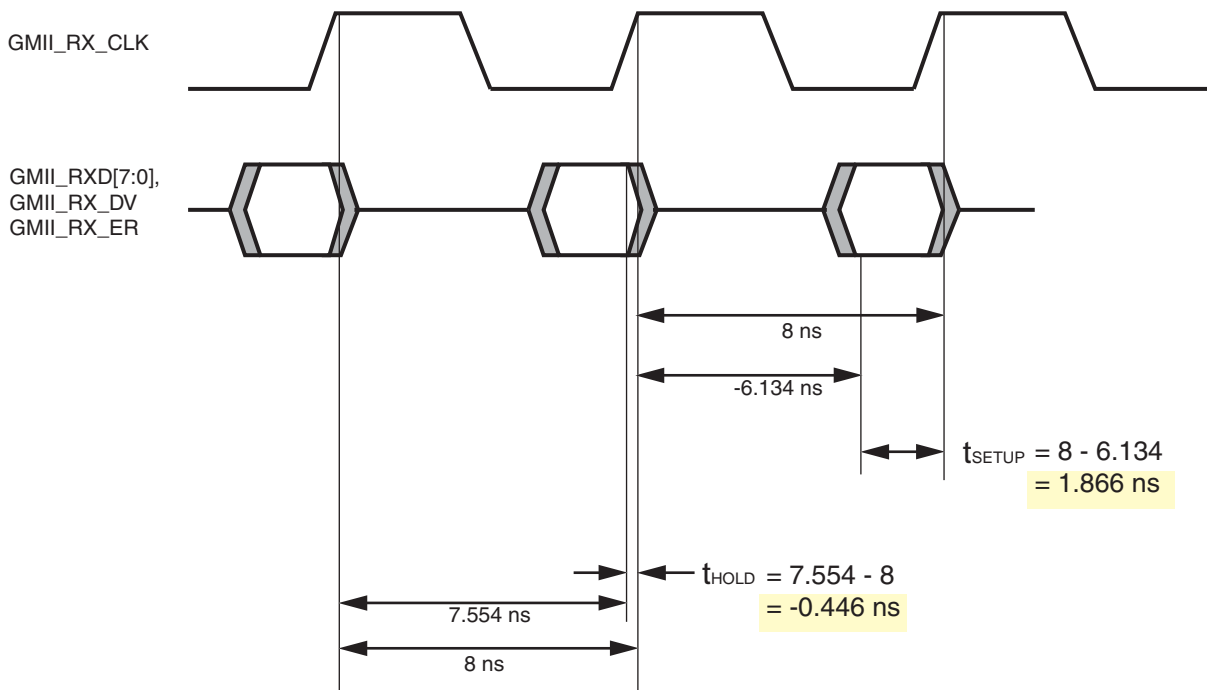


Figure 9-2: Timing Report Setup/Hold

Constraints when Implementing an External RGMII

The constraints defined in this section are implemented in the UCF for the example design delivered with the core. Sections from this UCF are copied into the descriptions below to act as an example. These should be studied in conjunction with the HDL source code for the example design and with the description given in “Implementing External RGMII,” on page 70.

RGMII IOB Constraints

The following constraints target the flip-flops that are inferred in the top level HDL file for the example design; constraints are set to ensure that these are placed in IOBs. The DDR register constraints are not present for a Virtex-4 or Virtex-5 device where DDR components are instantiated rather than inferred.

```
INST "rgmii_rxd_ddr*"      IOB = true;
INST "rgmii_rx_dv_ddr*"   IOB = true;
INST "rgmii_rx_ctl_ddr*"  IOB = true;

INST "inband_link_status" IOB = true;
INST "inband_clock_speed*" IOB = true;
INST "inband_duplex_status" IOB = true;
```

The RGMII v2.0 is a 1.5 volt signal-level interface. The 1.5 volt HSTL Class I SelectIO standard is used for RGMII interface pins. Use the following constraints with the device IO Banking rules. The IO slew rate is set to fast to ensure that the interface can meet setup and hold times.

```
INST "rgmii_txd<?>"      IOSTANDARD = HSTL_I;
```

```

INST "rgmii_tx_ctl"      IOSTANDARD = HSTL_I;
INST "rgmii_rxd<?>"     IOSTANDARD = HSTL_I;
INST "rgmii_rx_ctl"     IOSTANDARD = HSTL_I;

INST "rgmii_txc"        IOSTANDARD = HSTL_I;
INST "rgmii_rxc"        IOSTANDARD = HSTL_I;

INST "rgmii_txd<?>"     SLEW = FAST;
INST "rgmii_tx_ctl"     SLEW = FAST;
INST "rgmii_txc"        SLEW = FAST;

```

In addition, the example design provides pad locking on the RGMII for several families. This is provided as a guideline only; there are no specific I/O location constraints for this core.

RGMII Input Setup/Hold Timing

Figure 9-3 and Table 9-2 illustrate the setup and hold time window for the input RGMII signals. This is the worst-case data valid window presented to the FPGA device pins.

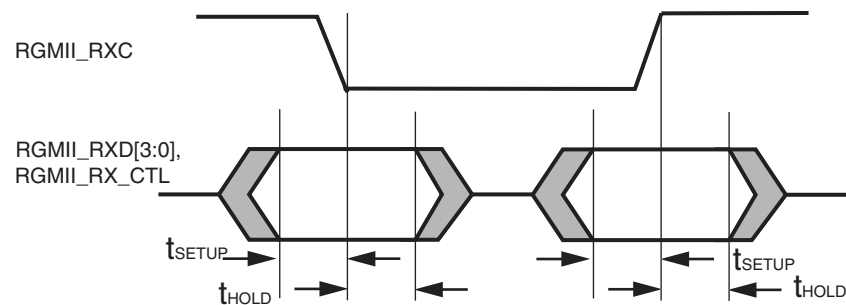


Figure 9-3: Input RGMII Timing

Observe that there is a 2 ns data valid window which is presented across the RGMII input bus. This must be correctly sampled on both clock edges by the FPGA devices.

Table 9-2: Input RGMII Timing

Symbol	Min	Typical	Units
t_{SETUP}	1.0	2.0	ns
t_{HOLD}	1.0	2.0	ns

For RGMII, the lower data bits, `rgmii_rxd[3:0]`, should be sampled internally on the rising edge of `rgmii_rxc`, and the upper data bits, `rgmii_rxd[7:4]`, should be sampled internally on the falling edge of `rgmii_rxc`.

Virtex-II, Virtex-II Pro, Spartan-3, Spartan-3E, and Spartan-3A Devices

The RGMII design uses a DCM on the receiver clock domain for all devices except Virtex-4 and Virtex-5. Phase-shifting is then applied to the DCM to align the resultant clock so that it will correctly sample the 2 ns RGMII data valid window at the input flip-flops.

The fixed phase shift is applied to the DCM using the following UCF syntax.

```

INST *gmii_rxc_dcm CLKOUT_PHASE_SHIFT = FIXED;
INST *gmii_rxc_dcm PHASE_SHIFT = 10;

```

The value of `PHASE_SHIFT` is preconfigured in the example designs to meet the setup and hold constraints for the example RGMII pinout in the particular device. The setup/hold

timing which is achieved after place-and-route is reported in the data sheet section of the TRCE report (created by the implement script).

For customers fixing their own pinout, the setup and hold figures reported in the TRCE report can be used to initially setup the approximate DCM phase shift. [Appendix D, “Calculating the DCM Phase Shift”](#) describes a more accurate method for fixing the phase shift by using hardware measurement of a unique PCB design.

Virtex-4 Devices

The RGMII design uses IDELAY components on the receiver clock, data and control signals for Virtex-4 devices. A fixed tap delay can be applied to either delay the data and control signals or delay the clock so that the data/control are correctly sampled by the `rgmii_rxc` clock at the IOB IDDR registers, meeting RGMII setup and hold timing.

The choice of delaying data/control or clock is dependant upon a number of factors, not least being the required shift. There are trade-offs to be made with either choice: Delaying the clock is clock period specific as we move the clock to line up each edge with data from the following edge. Delaying the data/control introduces more jitter which degrades the overall setup/hold window. The interface timing report in the two cases is also quite different and for this reason this is discussed in [“Understanding Timing Reports for RGMII Setup/Hold timing”](#).

The following constraint shows an example of setting the delay value for one of these IDELAY components. Data/Control bits can be adjusted individually, if desired, to compensate for any PCB routing skew.

```
INST *gmii_interface/delay_rgmii_rx_ctl IOBDELAY_VALUE = 40;
```

The value of IOBDELAY_VALUE is preconfigured in the example designs to meet the setup and hold constraints for the example RGMII pinout in the particular device. The setup/hold timing which is achieved after place-and-route is reported in the data sheet section of the TRCE report (created by the implement script).

When IDELAY or IODELAY primitives are instantiated with a fixed delay attribute, an IDELAYCTRL component must be also instantiated to continuously calibrate the individual input delay elements. The IDELAYCTRL module requires a reference clock, which is assumed to be an input to the example design delivered by CORE Generator. The most efficient way to use the IDELAYCTRL module is to lock the placement of the instance to the clock region of the device where the IDELAY/IODELAY components are placed. An example LOC constraint for the IDELAYCTRL module is shown commented out in the UCF. See the *Virtex-4 User Guide* and code comments for more information.

Virtex-5 Devices

The RGMII design uses IODELAY components on both the receiver and transmitter clock domains for Virtex-5 devices. A fixed tap delay is applied to the `rgmii_txc` output clock to move the rising edge of this clock to the centre of the output data window. For the receiver clock, data and control signals, a fixed tap delay can be applied to either delay the data and control signals or delay the clock so that the data/control are correctly sampled by the `rgmii_rxc` clock at the IOB IDDR registers, meeting RGMII setup and hold timing.

The choice of delaying data/control or clock is dependant upon a number of factors, not least being the required shift. There are trade-offs to be made with either choice: Delaying the clock is clock period specific as we move the clock to line up each edge with data from the following edge. Delaying the data/control introduces more jitter which degrades the overall setup/hold window. The interface timing report in the two cases is also quite

different and for this reason this is discussed in [“Understanding Timing Reports for RGMII Setup/Hold timing”](#).

The following constraint shows an example of setting the delay value for two of these IODELAY components. Data/Control bits can be adjusted individually, if desired, to compensate for any PCB routing skew.

```
INST *delay_rgmii_tx_clk IODELAY_VALUE = 26;

INST *gmii_interface/delay_rgmii_rx_ctl IODELAY_VALUE = 20;
```

The value of IODELAY_VALUE is preconfigured in the example designs to meet the setup and hold constraints for the example RGMII pinout in the particular device. The setup/hold timing which is achieved after place-and-route is reported in the data sheet section of the TRCE report (created by the implement script).

When IDELAY or IODELAY primitives are instantiated with a fixed delay attribute, an IDELAYCTRL component must be also instantiated to continuously calibrate the individual input delay elements. The IDELAYCTRL module requires a reference clock, which is assumed to be an input to the example design delivered by CORE Generator. The most efficient way to use the IDELAYCTRL module is to lock the placement of the instance to the clock region of the device where the IDELAY/IODELAY components are placed. An example LOC constraint for the IDELAYCTRL module is shown commented out in the UCF. See the *Virtex-5 User Guide* and code comments for more information.

RGMII DDR Constraints

If the core is implemented on a device other than Virtex-4 or Virtex-5, the following constraints are required to constrain the RGMII input registers for 1 Gbps operation. The RGMII design requires these clock crossing constraints to ensure timing is met when crossing from rising to falling clock edges and vice versa. A stringent time constraint ensures that timing is met with the worst-case timing allowed in the RGMII specification.

```
INST "rgmii_rxd_reg_4"      TNM="rgmii_falling";
INST "rgmii_rxd_reg_5"      TNM="rgmii_falling";
INST "rgmii_rxd_reg_6"      TNM="rgmii_falling";
INST "rgmii_rxd_reg_7"      TNM="rgmii_falling";
INST "rgmii_rx_ctl_reg"      TNM="rgmii_falling";
INST "gmii_rxd_reg_4"        TNM="rgmii_rising";
INST "gmii_rxd_reg_5"        TNM="rgmii_rising";
INST "gmii_rxd_reg_6"        TNM="rgmii_rising";
INST "gmii_rxd_reg_7"        TNM="rgmii_rising";
INST "gmii_rx_er_reg"        TNM="rgmii_rising";

TIMESPEC "TS_rgmii_falling_to_rising" = FROM "rgmii_falling" TO "rgmii_rising" 3200 ps;
```

Understanding Timing Reports for RGMII Setup/Hold timing

None Virtex-4 or Virtex-5 Devices

Setup and Hold results for the RGMII input bus can be found in the data sheet section of the Timing Report.

The Clock Generation logic includes a BUFGMUX which provides a path for the pre-DCM clock, see [Figure 7-4](#). When this BUFGMUX is present the timing engine will use the non-DCM clock path resulting in false setup and hold numbers. It is therefore necessary to edit the example design HDL to remove this non-DCM path if the setup/hold numbers are to be used for DCM phase adjustment.

After this is completed, the results are self-explanatory and it is easy to see how they relate to Figure 9-3. Here follows an example for the RGMII report from a Virtex-II device. Each Input lists two sets of values - one corresponding to the -ve edge of the clock and one to the +ve edge. The first set listed corresponds to -ve edge which occurs at time 4ns. The implementation requires 0.648ns of setup to the -ve edge and 0.661ns to the +ve edge: this is less than the 1ns required and so there is slack. The implementation requires 0.300 ns of hold to the -ve edge and 0.316ns to the +ve edge: this is less than the 1ns required and so there is slack.

Data Sheet report:

All values displayed in nanoseconds (ns)

Setup/Hold to clock rgmii_rxc

Source	Setup to clk (edge)	Hold to clk (edge)	Internal Clock(s)	Clock Phase
rgmii_rx_ctl	-3.352 (R) 0.661 (R)	4.300 (R) 0.284 (R)	not_rgmii_rx_clk_bufg rgmii_rx_clk_bufg	4.938 0.938
rgmii_rxd<0>	-3.384 (R) 0.629 (R)	4.332 (R) 0.316 (R)	not_rgmii_rx_clk_bufg rgmii_rx_clk_bufg	4.938 0.938
rgmii_rxd<1>	-3.348 (R) 0.665 (R)	4.296 (R) 0.280 (R)	not_rgmii_rx_clk_bufg rgmii_rx_clk_bufg	4.938 0.938
rgmii_rxd<2>	-3.360 (R) 0.653 (R)	4.308 (R) 0.292 (R)	not_rgmii_rx_clk_bufg rgmii_rx_clk_bufg	4.938 0.938
rgmii_rxd<3>	-3.428 (R) 0.585 (R)	4.382 (R) 0.366 (R)	not_rgmii_rx_clk_bufg rgmii_rx_clk_bufg	4.938 0.938

Virtex-4 or Virtex-5 Devices with Delayed Data/Control

Setup and Hold results for the RGMII input bus can be found in the data sheet section of the Timing Report. The results are self-explanatory and it is easy to see how they relate to Figure 9-3. Here follows an example for the RGMII report from a Virtex-5 device. Each Input lists two sets of values - one corresponding to the -ve edge of the clock and one to the +ve edge. The first set listed corresponds to +ve edge which occurs at time 0ns. The implementation requires 0.818ns of setup to the +ve edge and 0.794ns to the -ve edge: this is less than the 1ns required and so there is slack. The implementation requires 0.946 ns of hold to the +ve edge and 0.972ns to the -ve edge: this is less than the 1ns required and so there is slack.

Data Sheet report:

All values displayed in nanoseconds (ns)

Setup/Hold to clock rgmii_rxc

Source	Setup to clk (edge)	Hold to clk (edge)	Internal Clock(s)	Clock Phase
rgmii_rx_ctl	0.810 (R) -3.214 (F)	0.933 (R) 4.959 (F)	rgmii_rx_clk_bufg rgmii_rx_clk_bufg	0.000 4.000
rgmii_rxd<0>	0.811 (R) -3.213 (F)	0.940 (R) 4.966 (F)	rgmii_rx_clk_bufg rgmii_rx_clk_bufg	0.000 4.000
rgmii_rxd<1>	0.801 (R) -3.223 (F)	0.946 (R) 4.972 (F)	rgmii_rx_clk_bufg rgmii_rx_clk_bufg	0.000 4.000

rgmii_rxd<2>	0.818 (R)	0.929 (R)	rgmii_rx_clk_bufg	0.000
	-3.206 (F)	4.955 (F)	rgmii_rx_clk_bufg	4.000
rgmii_rxd<3>	0.809 (R)	0.936 (R)	rgmii_rx_clk_bufg	0.000
	-3.215 (F)	4.962 (F)	rgmii_rx_clk_bufg	4.000

Virtex-4 or Virtex-5 Devices with Delayed Clock

Setup and Hold results for the RGMII input bus can be found in the data sheet section of the Timing Report. However, depending on how the setup/hold requirements have been met the results can initially look strange and it is not immediately obvious how they relate to [Figure 9-3](#). Here follows an example for the RGMII report from a Virtex-4 device where the clock has been delayed to meet the setup/hold requirements.

Data Sheet report:

All values displayed in nanoseconds (ns)

Setup/Hold to clock rgmii_rxc

Source	Setup to clk (edge)	Hold to clk (edge)	Internal Clock(s)	Clock Phase
rgmii_rx_ctl	-7.178 (R)	8.880 (R)	rgmii_rx_clk_bufg	0.000
	-11.178 (F)	12.880 (F)	rgmii_rx_clk_bufg	4.000
rgmii_rxd<0>	-7.192 (R)	8.893 (R)	rgmii_rx_clk_bufg	0.000
	-11.192 (F)	12.893 (F)	rgmii_rx_clk_bufg	4.000
rgmii_rxd<1>	-7.182 (R)	8.884 (R)	rgmii_rx_clk_bufg	0.000
	-11.182 (F)	12.884 (F)	rgmii_rx_clk_bufg	4.000
rgmii_rxd<2>	-7.180 (R)	8.882 (R)	rgmii_rx_clk_bufg	0.000
	-11.180 (F)	12.882 (F)	rgmii_rx_clk_bufg	4.000
rgmii_rxd<3>	-7.179 (R)	8.881 (R)	rgmii_rx_clk_bufg	0.000
	-11.179 (F)	12.881 (F)	rgmii_rx_clk_bufg	4.000

Each Input lists two sets of values - one corresponding to the +ve edge of the clock and one to the -ve edge. The first set listed corresponds to +ve edge which occurs at time 8ns as we have delayed the clock to use the following +ve edge.

The implementation requires -7.179 ns of setup to the +ve edge. [Figure 9-4](#) illustrates that this represents a figure of 0.821ns relative to the following rising edge of the clock (since the IDELAY has acted to delay the clock by an entire period when measured from the input flip-flop). This is less than the 1ns required and so there is slack. Equally for the -ve edge, we have -11.179ns of setup - this edge is at time 12ns and therefore this equates to a setup figure of 0.821ns.

The implementation requires 8.893ns of hold to the +ve edge. [Figure 9-4](#) illustrates that this represents a figure of 0.893 ns relative to the following rising edge of the clock (since the IDELAY has acted to delay the clock by an entire period when measured from the input flip-flop). This is less than the 1ns required and so there is slack. Equally for the -ve edge,

we have 12.893ns of hold - this edge is at time 12ns and therefore this equates to a hold figure of 0.893ns.

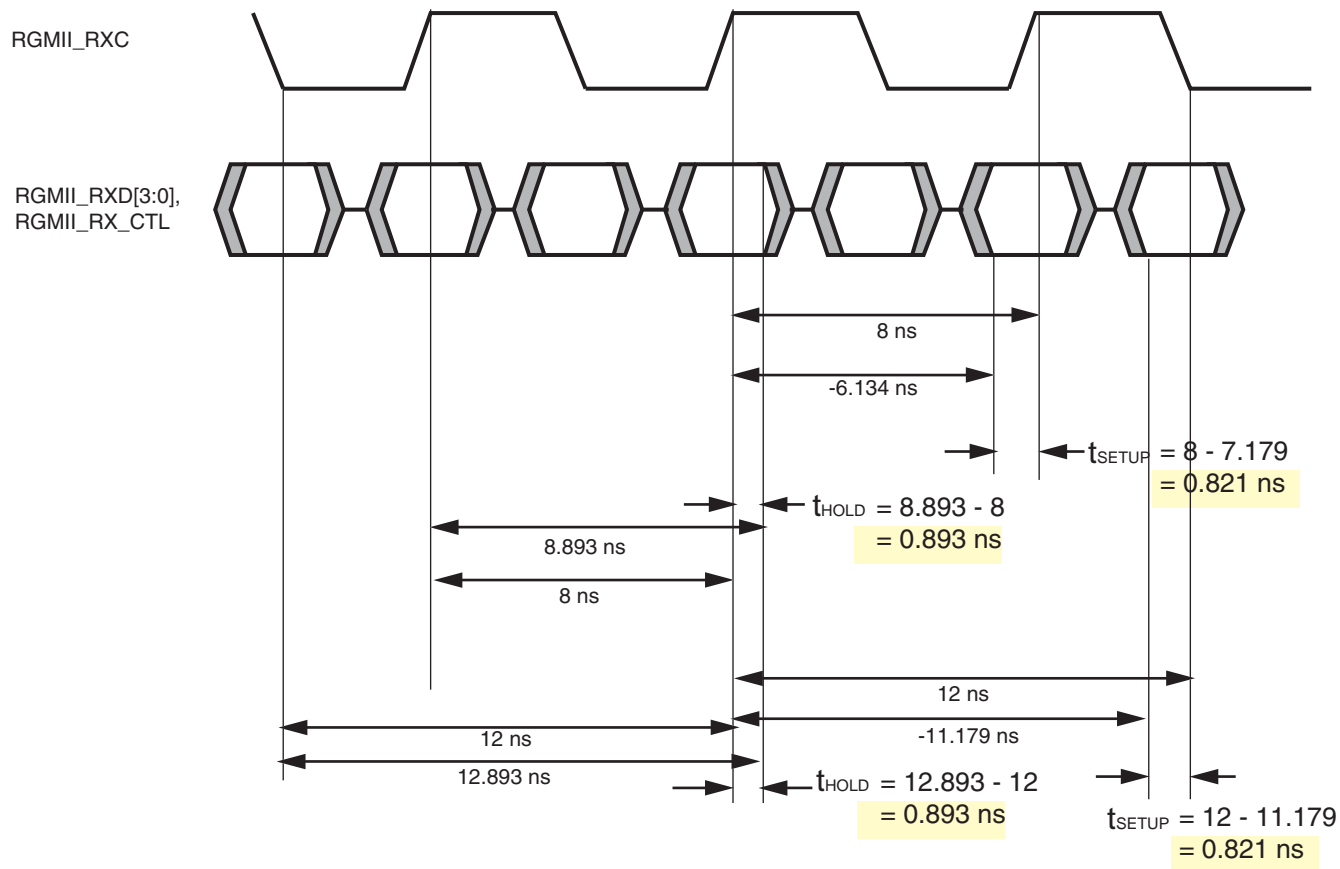


Figure 9-4: Timing Report Setup/Hold

Clocking and Reset

This chapter describes design considerations associated with implementing the TEMAC core.

Clocking

The example design files included with the TEMAC core include clocking circuitry to drive the core at all three speeds.

GMII/MII Transmit Clock Generation

Figure 10-1 illustrates the GMII/MII transmit clocking circuit for all device families.

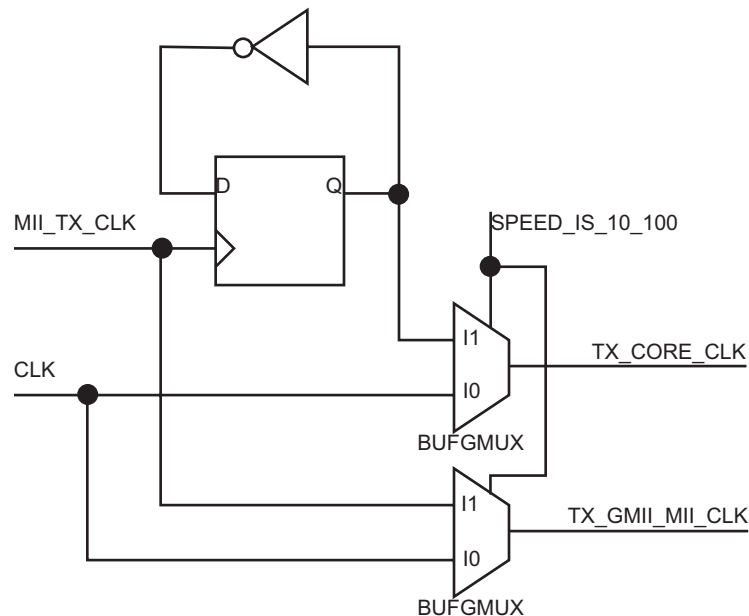


Figure 10-1: GMII/MII Transmit Clock Generator

CLK must be provided to the MAC clock circuitry. This is a high quality 125 MHz clock which satisfies the *IEEE 802.3-2002* requirements. It is expected that this clock will be derived from an external oscillator and connected into the device through an IBUFG. MII_TX_CLK is provided by the PHY chip at speeds of 100 Mbps and below.

If the clock_enables option is set to false, the core requires two transmit clocks. TX_CORE_CLK drives the 8-bit data path in the core, and the client logic.

TX_GMII_MII_CLK drives the GMII/MII logic in the core and the example design. This clock is twice the frequency of the core clock when the device is operating at speeds below 1 Gbps. This is due to the fact that the MII interface implements a 4-bit data path. At these speeds, the 4-bit data is carried on GMII_TXD[3 downto 0]. The upper bits are set to zero. TX_CORE_CLK is connected to the txcoreclk input of the core. TX_GMII_MII_CLK is connected to the txgmiiimiclk input. The clock selection is dependent on the state of the SPEED_IS_10_100 input to the circuit. This is connected to the speedis10100 output of the core. For more information on the GMII/MII transmit interface, see “GMII/MII Transmit Interface,” on page 63.

If the core is not required to operate at above 100 Mbps, the clocking scheme can be simplified to remove the 125 MHz CLK input. Figure 10-2 shows the transmit clocking scheme for a 10/100 Mbps implementation.

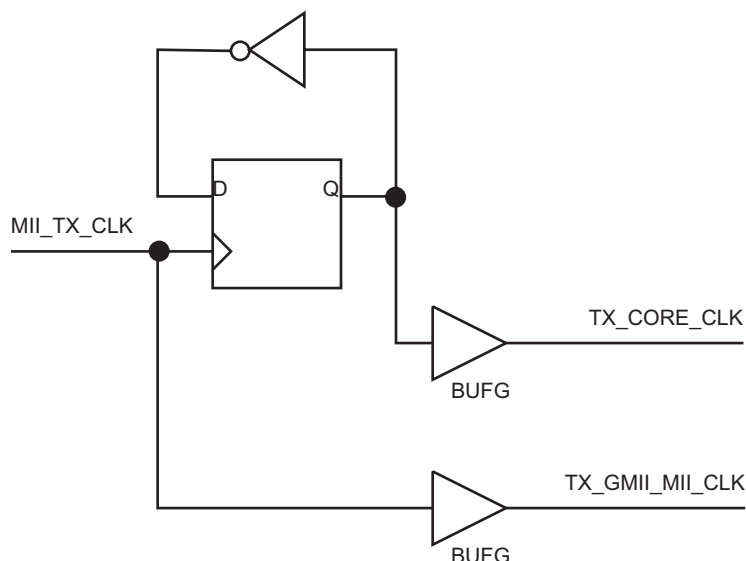


Figure 10-2: 10/100 Mbps MII Transmit Clock Generator

If the core is generated with the clock_enables option set to true, the TX_CORE_CLK generation in Figure 10-1 and Figure 10-2 is omitted. The core clock is the TX_GMII_MII_CLK signal with the user supplying a clock enable to the core and to the remainder of the transmit client circuitry. The transmit clock generation for cores operating at 10/100/1 Gbps in this case is illustrated in Figure 10-3.

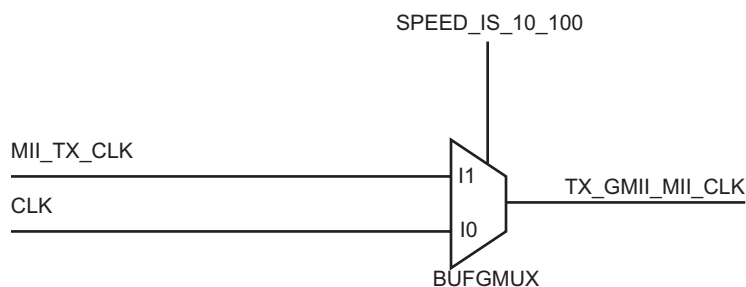


Figure 10-3: GMII/MII Transmit Clock Generator (clock_enables = true)

GMII/MII Receive Clock Generation

Figure 10-4 shows the GMII/MII receiver clock generation circuit for all families apart from Spartan-3, Spartan-3E and Spartan-3A.

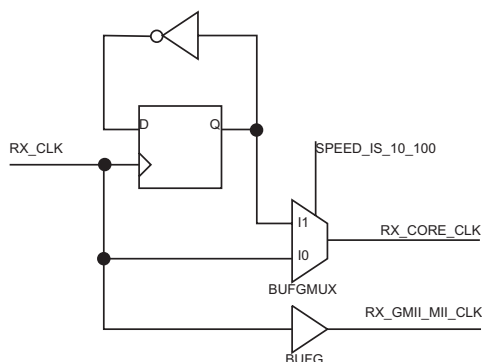


Figure 10-4: GMII/MII Receive Clock Generator

`RX_CLK` is provided by the PHY chip via an `IBUFG`. This clock is output to the `RX_GMII_MII_CLK` port via a `BUFG` where it is used to clock the GMII/MII receiver.

If `SPEED_IS_10_100` (`speedis10100` output from the core) is '0', `RX_CORE_CLK` is generated from a frequency divided by 2 version of `RX_CLK` via a `BUFGMUX`. If `SPEED_IS_10_100` is '1', `RX_CLK` is routed through the `BUFGMUX`. The resulting global clock is used by the core receiver and client side logic. For more information on the GMII/MII receive interface, see [“GMII/MII Receive Interface,”](#) on page 66.

If the core is not required to operate at speeds over 100 Mbps, the clocking scheme can be simplified to remove the `BUFGMUX` as `RX_CORE_CLK` will always be half the frequency of `RX_GMII_MII_CLK`. shows the receiver clock generator for a 10/100 Mbps implementation.

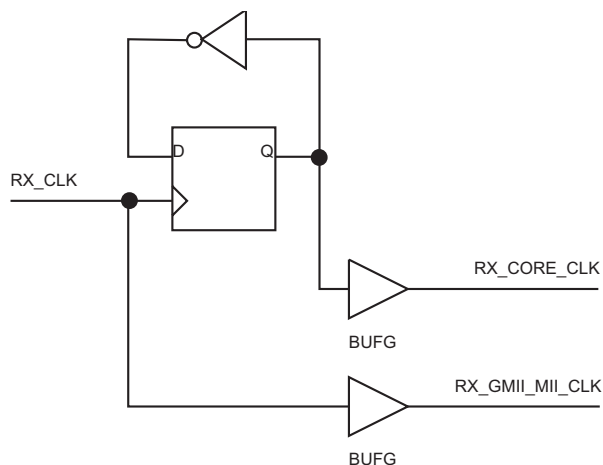


Figure 10-5: 10/100 Mbps MII Receive Clock Generator

If the core is generated with the `clock_enables` option set to true, the `RX_CORE_CLK` generation from Figure 10-4 and Figure 10-5 is omitted. The core clock is the `RX_GMII_MII_CLK` signal with the user supplying a clock enable to the core and to the remainder of the receiver client circuitry. This is illustrated in Figure 10-6.

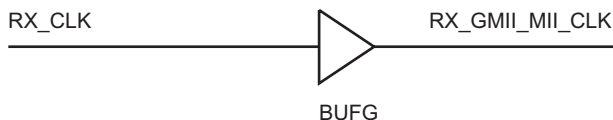


Figure 10-6: GMII/MII Receive Clock Generator (`clock_enables` = true)

RGMII Transmit Clock Generation

Figure 10-7 shows the RGMII transmit clock generator circuit for all device families except Virtex-5.

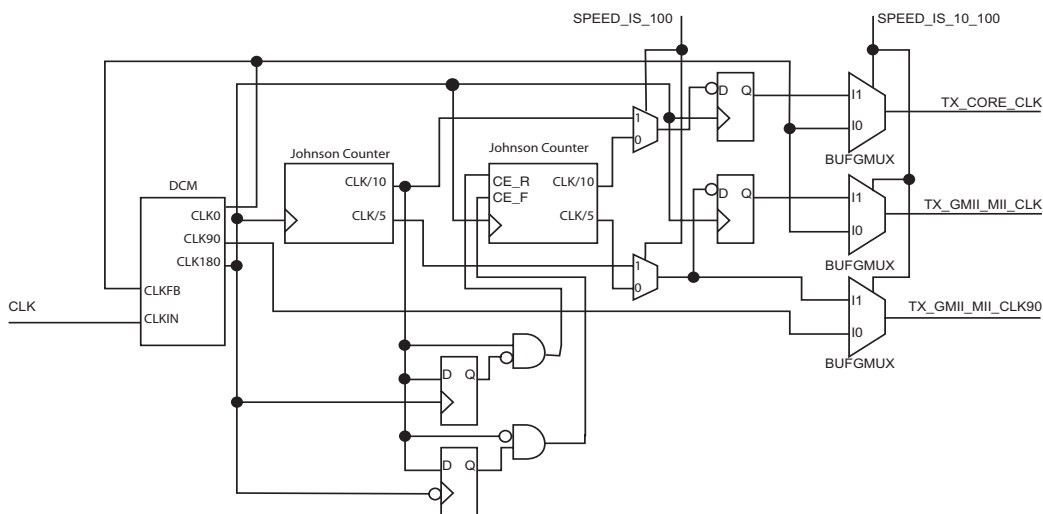


Figure 10-7: RGMII Transmit Clock Generator (`clock_enables` = false)

A high quality 125 MHz clock (CLK) is supplied to the clock circuit. This is then input to a DCM. The CLK0 output from the DCM is used to generate the `TX_CORE_CLK` (if the core is generated with the `clock_enables` option set to false) and `TX_GMII_MII_CLK` outputs.

A Johnson counter is used to divide the CLK input down by 5 and 10 to provide clocks of 25 MHz and 12.5 MHz. The 12.5 MHz clock is then used to generate clock enable inputs to a second Johnson counter to provide clocks of 2.5 MHz and 1.25 MHz. These are then routed to the `TX_CORE_CLK` and `TX_GMII_MII_CLK` outputs depending on the state of the `SPEED_IS_10_100` and `SPEED_IS_100` outputs from the core.

When the core is running at 1 Gbps, the `rgmii_txc` clock output must toggle at the center of the valid data. To do this, the CLK90 output from the DCM is routed to the `TX_GMII_MII_CLK90` output where it is used to generate `rgmii_txc`. See “RGMII Transmit Interface,” on page 70 for more details. At lower speeds, the clock for `rgmii_txc` is generated from the CLK180 output of the DCM via two Johnson counters.

If the core is generated with the optional clock enable circuitry, `TX_CORE_CLK` is unnecessary. In this case, the `TX_GMII_MII_CLK` output is used to clock the entire

transmitter circuit with the user supplying a clock enable to the core and the remainder of the client circuitry, as illustrated in [Figure 10-8](#).

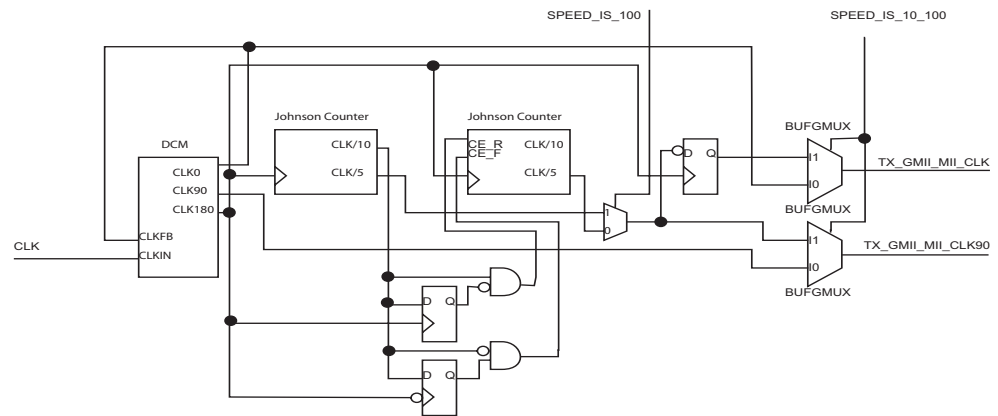


Figure 10-8: RGMII Transmit Clock Generator (clock_enables = true)

For Virtex-5 devices, the RGMII transmit clock generation is simplified, as a 90 degree phase-shifted clock can be generated using an IODELAY component described in “RGMII Transmit Interface” in Chapter 7. Figure 10-9 shows the RGMII transmit clocking circuit for Virtex-5 devices.

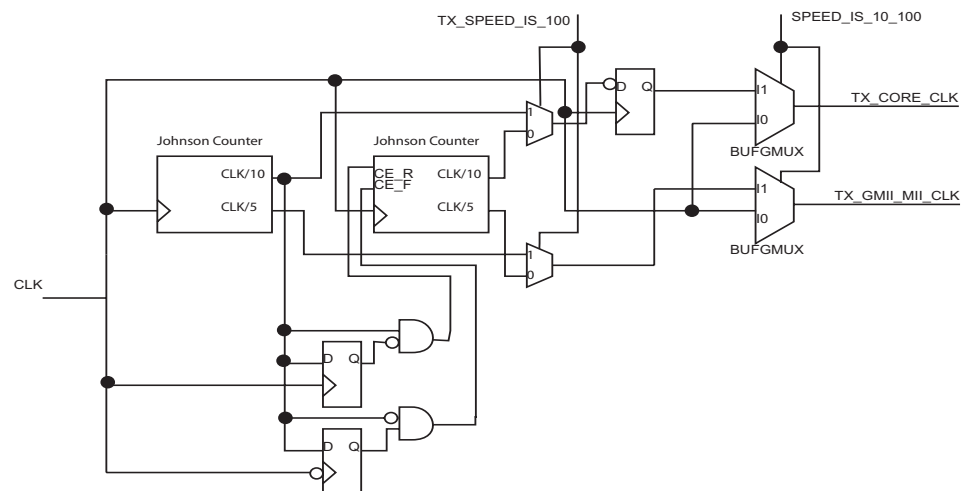


Figure 10-9: RGMII Transmit Clock Generator (clock_enables = false) for Virtex-5

A high quality 125 MHz clock (CLK) is supplied to the clock circuit. For 1 Gbps operation, this clock is routed through separate BUFGMUXs to create the TX_GMII_MII_CLK output and the TX_CORE_CLK output (if the core is generated with the clock_enables option set to false).

A Johnson counter is used to divide the CLK input down by 5 and 10 to provide clocks of 25 MHz and 12.5 MHz. The 12.5 MHz clock is then used to generate clock enable inputs to a second Johnson counter to provide clocks of 2.5 MHz and 1.25 MHz. These are then routed to the TX_CORE_CLK and TX_GMII_MII_CLK outputs depending on the state of the SPEED_IS_10_100 and SPEED_IS_100 outputs from the core.

RGMII Receive Clock Generation

Figure 10-10 shows the clock generation circuitry for the RGMII receiver for all device families except Virtex-4 and Virtex-5. At 1 Gbps, the clock is generated from the 125 MHz `rgmii_rxc` input via a DCM. The DCM is set-up in fixed-phase shift mode. The phase shift value can be varied to provide the correct set-up and hold times at the receiver input. See “Calculating the DCM Phase Shift,” on page 145.

At slower speeds the DCM is bypassed (and held in reset). This is due to the fact that the minimum CLKIN input of the DCM is 24 MHz. This is too low for the 10 Mbps `rgmii_rxc` clock, which runs at a frequency of 2.5 MHz.

At 10/100 Mbps the clock is simply the `rgmii_rxc` input routed through a BUFGMUX.

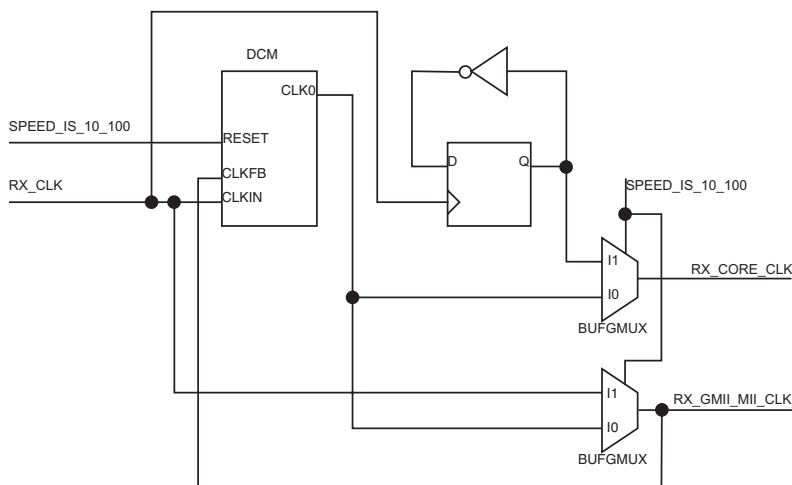


Figure 10-10: RGMII Receive Clock Generator (clock_enables = false)

If the core is generated with the `clock_enables` option set to true, the `RX_CORE_CLK` generation is omitted. The core clock is the `RX_GMII_MII_CLK` signal with the user supplying a clock enable to the core and to the remainder of the receive client circuitry. This is shown in Figure 10-11.

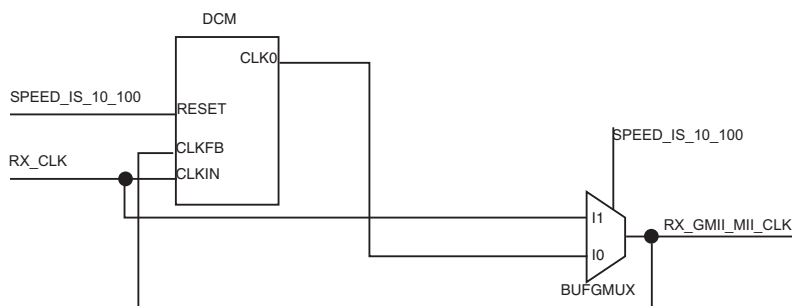


Figure 10-11: RGMII Receive Clock Generator (clock_enables = true)

For Virtex-4 and Virtex-5 devices the receive clock generation is simplified, as illustrated in Figure 10-12. In Virtex-4 and Virtex-5 devices it is possible to meet the RGMII setup and hold requirements by skewing the data and control signals using input delay elements. See

“RGMII Receiver Interface” in Chapter 7 for details on how this is achieved for these families.

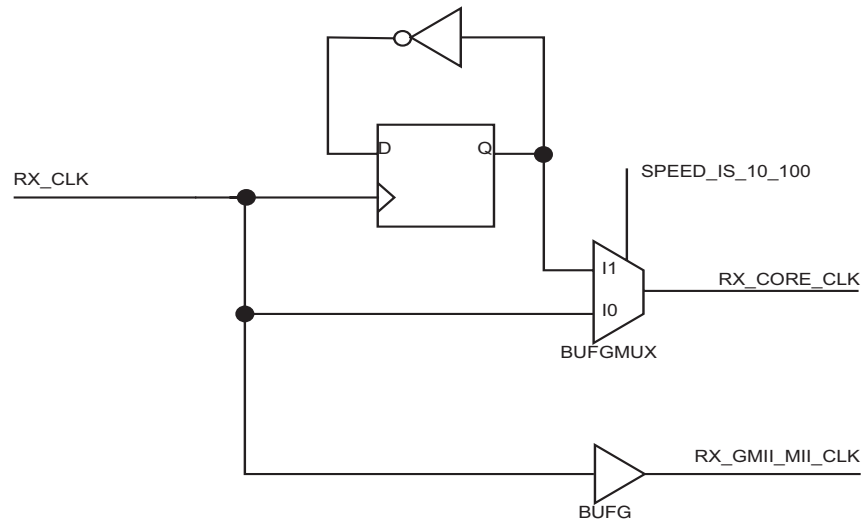


Figure 10-12: RGMII Receive Clock Generator (clock_enables = false) for Virtex-4 and Virtex-5 Devices

Multiple Cores

Clock Sharing

Figure 10-13 illustrates how it is possible to share clock resources across multiple instantiations of the core when using the GMII/MII interface.

A common receiver clock domain is not possible; each core will receive an independent receiver clock from its GMII as illustrated.

At speeds below 1 Gbps, the transmit clock is also an input to the MAC. It is possible to share the clock if the PHY outputs an identical clock for each MAC. If these clocks are different, it is not possible to share the clock circuitry. The shared clock MACs must be operating at the same speed.

Although not illustrated, if the optional Management Interface is used, `hostclk` can also be shared between cores.

If the RGMII interface is used, the `rgmii_txc` clock output can be shared between multiple cores as long as they are all running at the same speed.

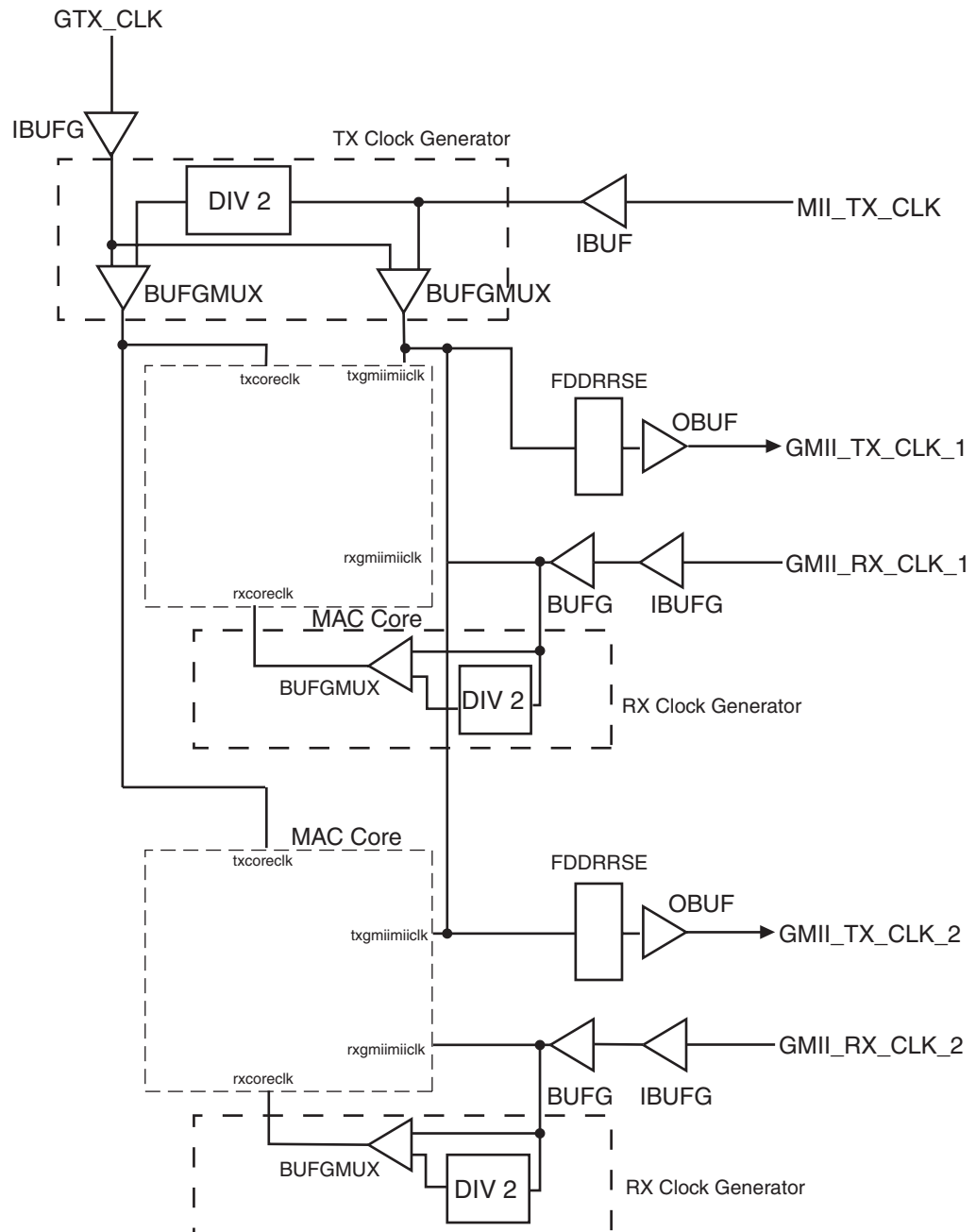


Figure 10-13: Clock Sharing across Two MAC Cores

BUFGMUX Usage

The BUFGMUX components in Virtex-II, Virtex-II Pro, Spartan-3, Spartan-3E and Spartan-3A devices are arranged in pairs. Both the inputs to each pair must be the same. For the BUFGMUXes in the TEMAC this is not the case and so the partner of each

BUFGMUX in the clock circuit cannot be used by another clock. This is illustrated in Figure 10-14 for the generation of txgmiimiiclk.

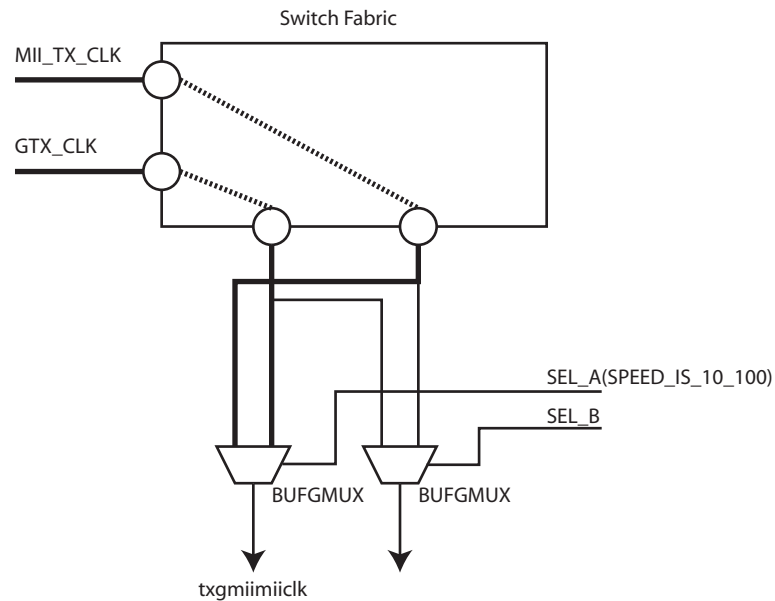


Figure 10-14: Suggested BUFGMUX Scheme

If there are multiple cores on the chip, or if the user logic requires many independent clocks, it is possible to multiplex the two clocks in the FPGA fabric and route the resultant signal through a BUFG component (essentially a BUFGMUX with a constant select line). This will free up the partner BUFGMUX for use by a different clock. This is illustrated in Figure 10-15.

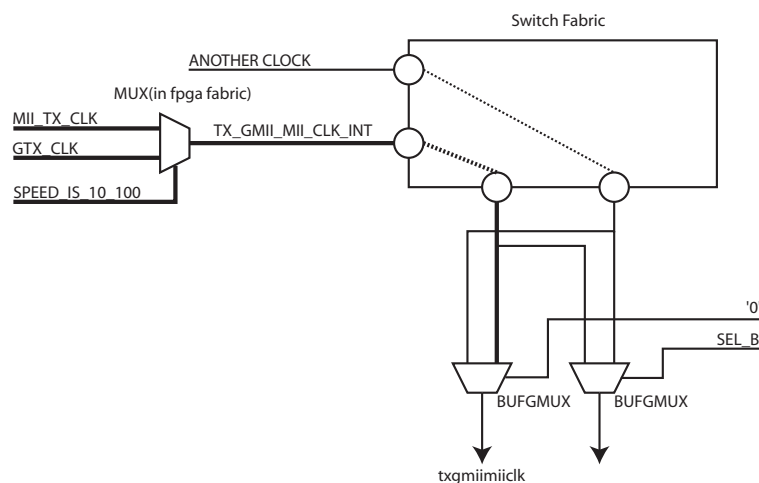


Figure 10-15: Alternative BUFGMUX Scheme

No glitches or short pulses appear on the output of the BUFGMUX components when the select line is toggled. Performing the multiplexing in the FPGA fabric removes this safeguard, and if the user has not implemented a glitch free clock multiplexer circuit, a

reset should be performed after a speed change. The reset has no effect on the speed setting, which is preserved in the configuration registers.

Reset Conditions

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 10-16. This circuit provides controllable skews on the reset nets within the design.

More information on the operation and rationale behind this circuit can be found in Ken Chapman's Xilinx TechXclusive, "Get Smart About Reset" at:

www.xilinx.com/support/techxclusives/global-techX19.htm

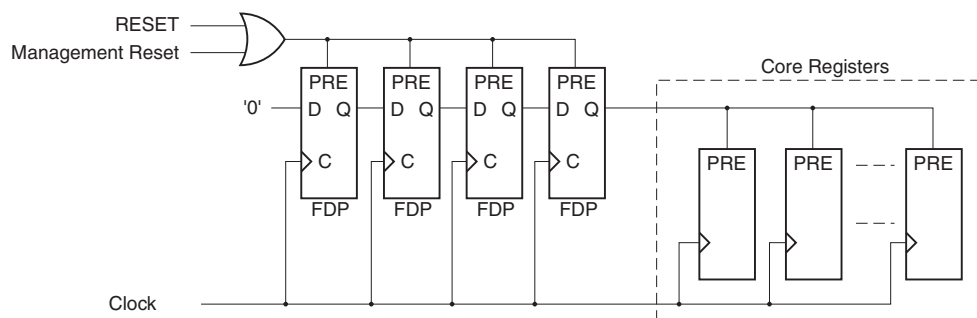


Figure 10-16: Reset Circuit for One Clock/reset Domain

Interfacing to Other Cores

Integrating with the Ethernet 1000BASE-X PCS/PMA or SGMII Core

The TEMAC core can be integrated in a single device with the Ethernet 1000BASE-X PCS/PMA or SGMII core (Virtex II Pro, Virtex-4, and Virtex-5 only) to provide a SGMII interface to an external PHY chip.

A description of the latest available IP Update containing the Ethernet 1000BASE-X PCS/PMA or SGMII core and instructions on obtaining the IP update can be found in the Ethernet 1000BASE-X PCS/PMA or SGMII Product web site at:

www.xilinx.com/systemio/1gbsx_phy/index.htm

A description of the Ethernet 1000BASE-X PCS/PMA or SGMII core is outside the scope of this document.

The Tri-Mode Ethernet MAC should always be configured for full-duplex operation when used with an SGMII. This constraint is due to the increased latency introduced by the SGMII logic; frame collisions and the MACs response will not be detected or made in time.

Integration to Provide SGMII

Virtex-II Pro Devices

Figure 11-1 illustrates the connections and clock management logic required to interface the TEMAC core to the Ethernet 1000BASE-X PCS/PMA or SGMII core in Virtex-II pro devices. This shows that:

- The TEMAC core is generated with optional clock enables.
- The cores are connected together via a SGMII adaptation module. This generates the clock enable needed to run the TEMAC at speeds below 1 Gbps. The clock enable should also be used to enable the client transmit and receive circuitry. These ensure that data is only sampled every 10 clock cycles at 100 Mbps and every 100 clock cycles at 10 Mbps.
- If the TEMAC has been built with the optional management logic (see “Using the Optional Management Interface,” page 81), the MDIO port can be connected up to that of the Ethernet 1000BASE-X PCS/PMA or SGMII core to access its embedded configuration and status registers.

Some simplification to the UCF required for use with the TEMAC is possible. The constraints which cover the clocks (with the exception of the `hostclk`) can be removed as these are covered by the constraints in the Ethernet 1000BASE-X PCS/PMA or SGMII core

UCF. The GMII Transmitter and Receiver Constraints can also be removed as these signals are no longer routed to IOBs.

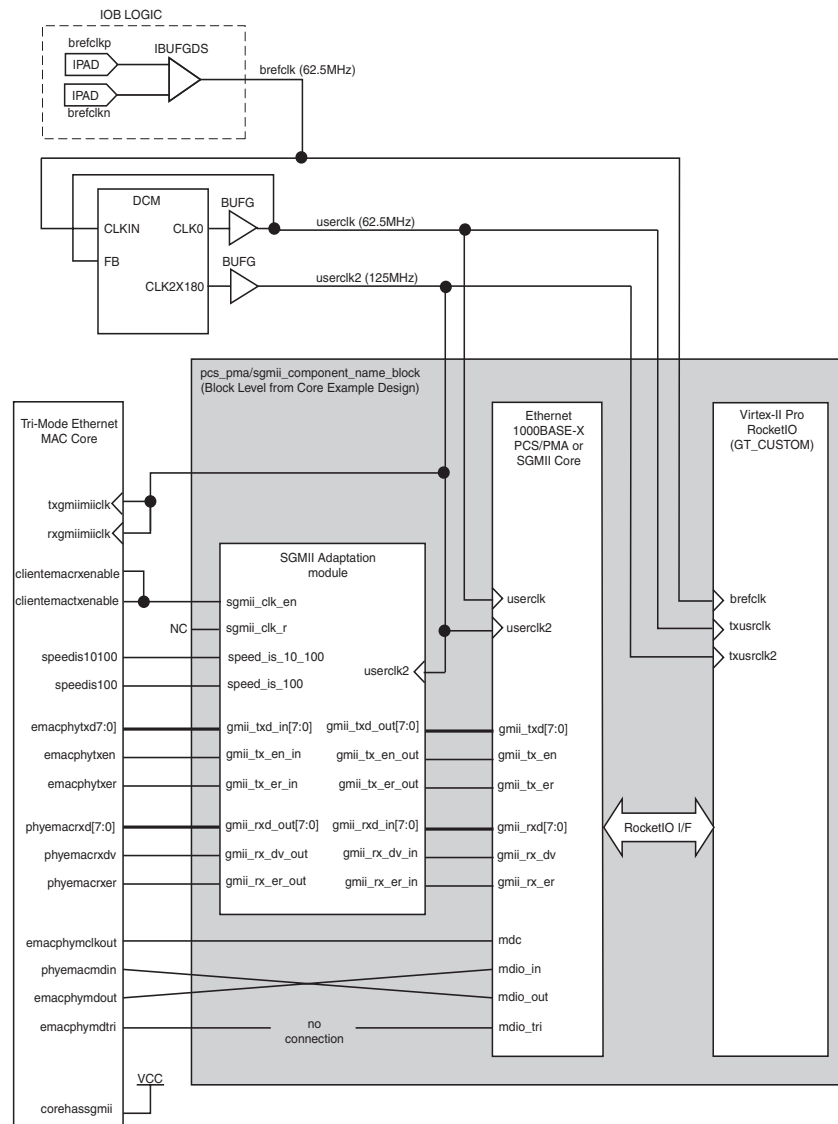


Figure 11-1: Tri-Mode Ethernet MAC Extended to Implement SGMII (Virtex-II Pro)

Virtex-4 Devices

Figure 11-2 illustrates the connections and clock management logic required to interface the TEMAC core to the Ethernet 1000BASE-X PCS/PMA or SGMII core in Virtex-4 devices. As in the Virtex-II Pro architecture, the TEMAC core is generated with optional clock enables and the interface between the two cores is provided by an SGMII Adaptation module.

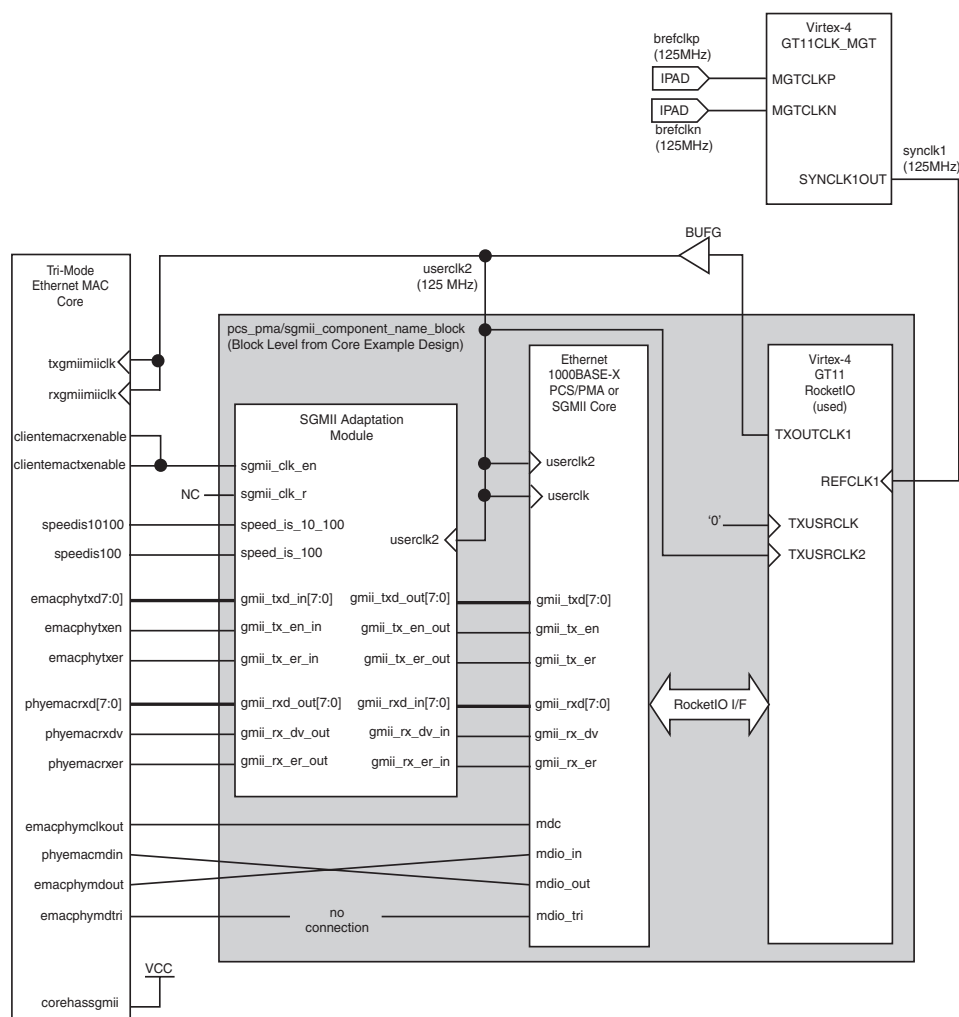


Figure 11-2: Tri-Mode Ethernet MAC Extended to Implement SGMII (Virtex-4)

As in the Virtex-II Pro, some simplification to the UCF required for use with the TEMAC is possible. The constraints which cover the clocks (with the exception of the `hostclk`) can be removed as these are covered by the constraints in the Ethernet 1000BASE-X PCS/PMA or SGMII core UCF. The GMII Transmitter and Receiver Constraints can also be removed as these signals are no longer routed to IOBs.

Virtex-5 Devices

Figure 11-3 illustrates the connections and clock management logic required to interface the TEMAC core to the Ethernet 1000BASE-X PCS/PMA or SGMII core in Virtex-5 devices. As with the Virtex-II Pro architecture, the TEMAC core is generated with optional clock enables and the interface between the two cores is provided by an SGMII Adaptation module.

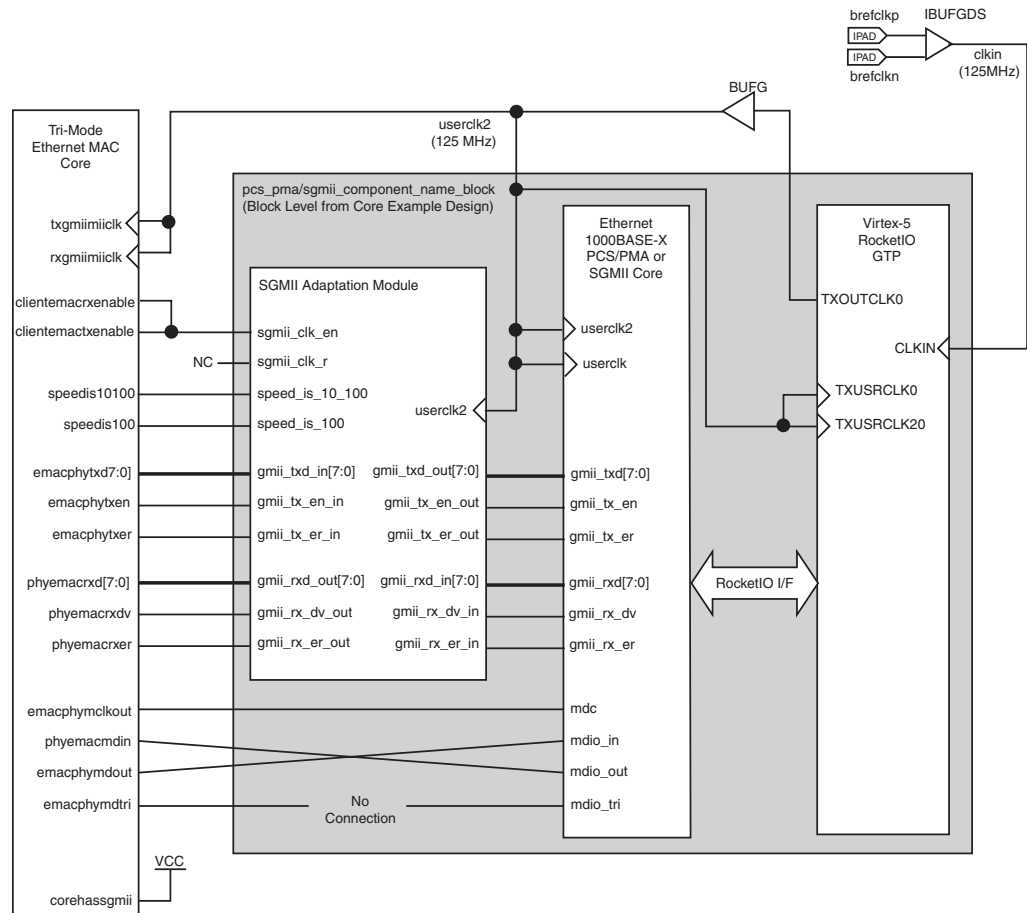


Figure 11-3: Tri-Mode Ethernet MAC Extended to Implement SGMII (Virtex-5)

As in the Virtex-II Pro, some simplification to the UCF required for use with the TEMAC is possible. The constraints which cover the clocks (with the exception of the `hostclk`) can be removed as these are covered by the constraints in the Ethernet 1000BASE-X PCS/PMA or SGMII core UCF. The GMII Transmitter and Receiver Constraints can also be removed as these signals are no longer routed to IOBs.

Integrating with the Ethernet Statistics Core

The TEMAC can be integrated with the Ethernet Statistics core to provide statistical information on the frames that are processed by the MAC. Figure 11-4 illustrates the connections required to interface the two cores when the TEMAC is generated with optional clock enables. If the TEMAC is generated without clock enables, `txcoreclk` and `rxcoreclk` are connected to the `tx_clk` and `rx_clk` inputs of the Ethernet Statistics core. The `tx_enable` and `rx_enable` inputs of the Ethernet Statistics core are tied high.

The Ethernet Statistics core contains two parts:

- The vector decode module decodes the information contained in the statistics vector outputs from the TEMAC core. This block can be modified to allow the user to gather information on the types of frame that are of interest.
- The Statistics 32/64-bit module contains the statistic counters. If the 32-bit module is selected, each counter can count up to $2^{32}-1$ frames. If the 64-bit module is selected, the counters can count $2^{64}-1$ frames. The information from the counters is read back via the Management Interface.

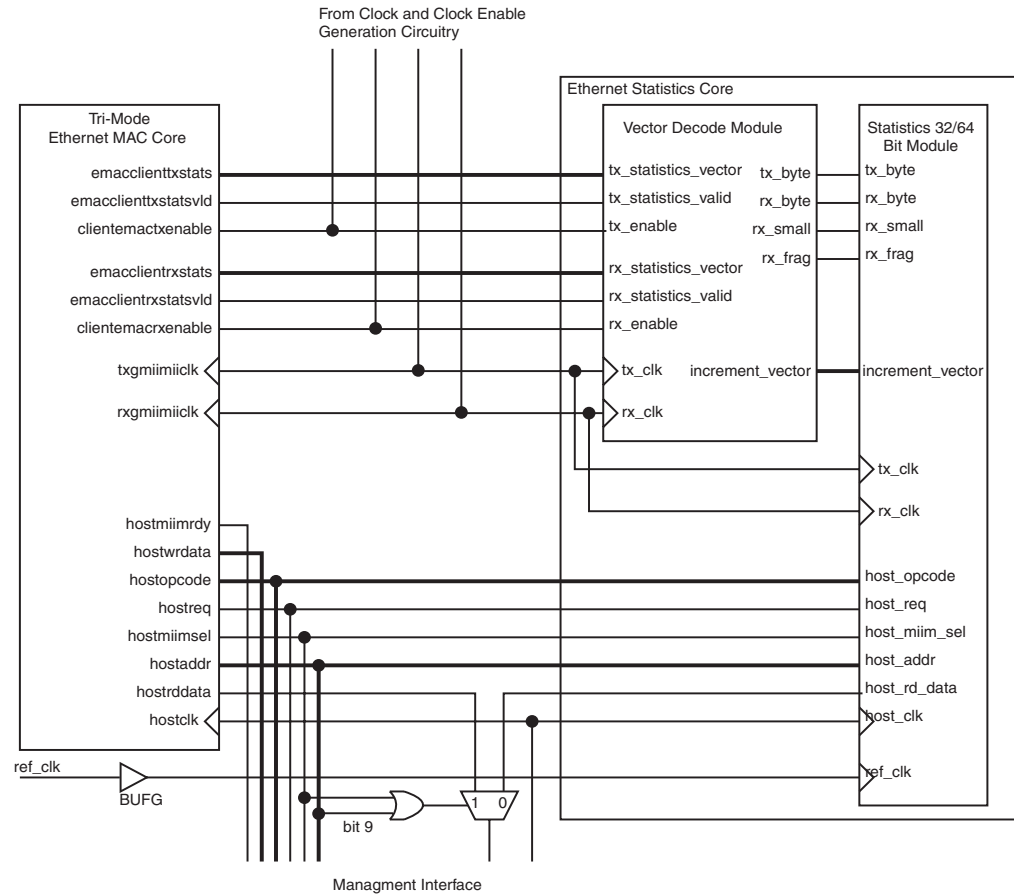


Figure 11-4: Tri-Mode Ethernet MAC with Statistics

For more information on the Ethernet Statistics core, see the *LogiCORE Ethernet Statistics User Guide*.

Implementing Your Design

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

Pre-implementation Simulation

The CORE Generator generates a functional model of the TEMAC core netlist to allow simulation of the block in the design phase of the project.

Using the Simulation Model

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

The model is provided in the CORE Generator project directory.

VHDL

`<component_name>.vhd`

Verilog

`<component_name>.v`

This model can be compiled along with the users code to simulate the overall system.

Synthesis

XST - VHDL

In the CORE Generator project directory, there is a `<component_name>.vho` file that is a component and instantiation template for the core. Use this to help instance the TEMAC core into your VHDL source.

After your entire design is complete, create the following:

- 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 project directory, locate the module declaration for the TEMAC core at:

```
<component_name>/implement/<component_name>.mod.v
```

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

After your entire design is complete, create

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

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

and

```
<component_name>/implement/component_name.mod.v
```

as the first two files in the project list.

- An XST script file *top_level_module_name.scr* containing your required synthesis options

To synthesize the design, run

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

See the *XST User Guide* for more information 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_core_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 top_level_module_name -o top_level_module_name_map.ncd
```

Placing and Routing the Design

To place and route the user design's logic components (mapped physical logic cells) contained within an NCD file in accordance with the layout and timing requirements specified within 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, run the `netgen` command.

VHDL

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

Verilog

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

Using the Model

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

Other Implementation Information

For more information about using the Xilinx implementation tool flow, including command line switches and options, see the Xilinx ISE software manuals.

Using the Client Side FIFO

The example design provided with the TEMAC core contains a LocalLink FIFO used to interface to the client side of the TEMAC core. The source code for the FIFO is provided, and can be used and edited for user applications.

The 10 Mbps/100 Mbps/1 Gbps Ethernet FIFO consists of independent transmit and receive FIFOs embedded in a top-level wrapper. [Figure A-1](#) shows how the FIFO fits into a typical implementation. Each FIFO is built around two Dual Port Block RAMs giving a memory capacity of 4096 bytes in each FIFO. This chapter describes the operation of the FIFO.

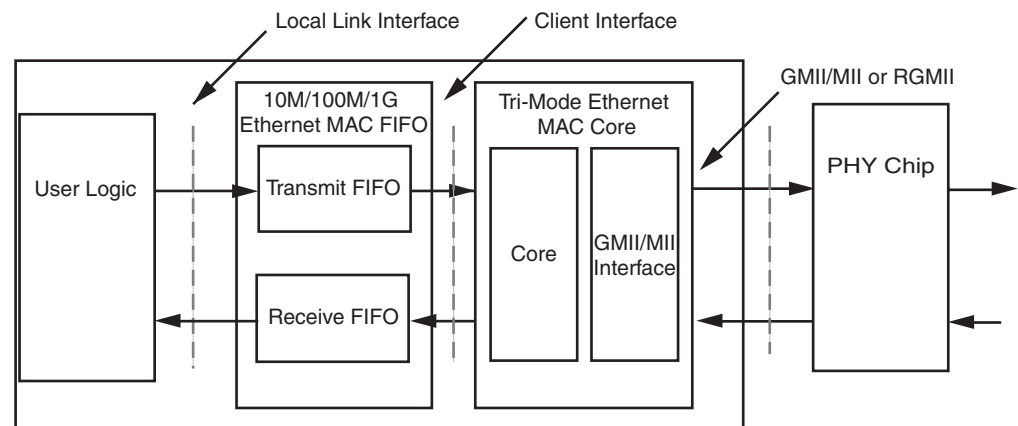


Figure A-1: Typical 10M/100M/1G Ethernet FIFO Implementation

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. For more information on the LocalLink interface, see Xilinx Application Note [XAPP691](#), "Parameterizable LocalLink FIFO." [Figure A-2](#) shows the transfer of an 8-byte frame.

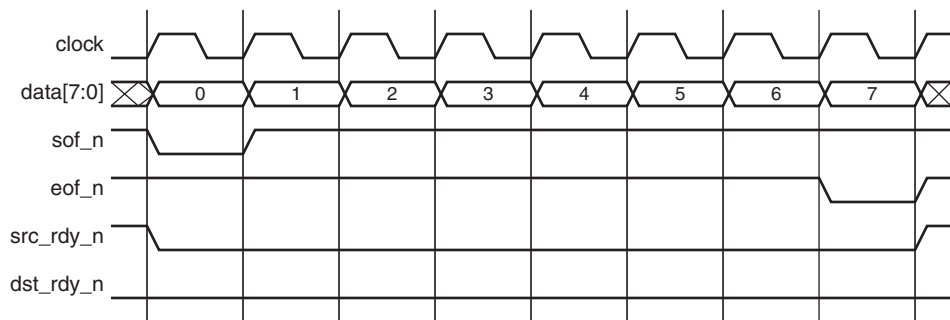


Figure A-2: Frame Transfer across LocalLink Interface

Figure A-3 illustrates frame transfer of a 5-byte frame, where both the `src_rdy_n` and `dst_rdy_n` signals are used to control the flow of data across the interface.

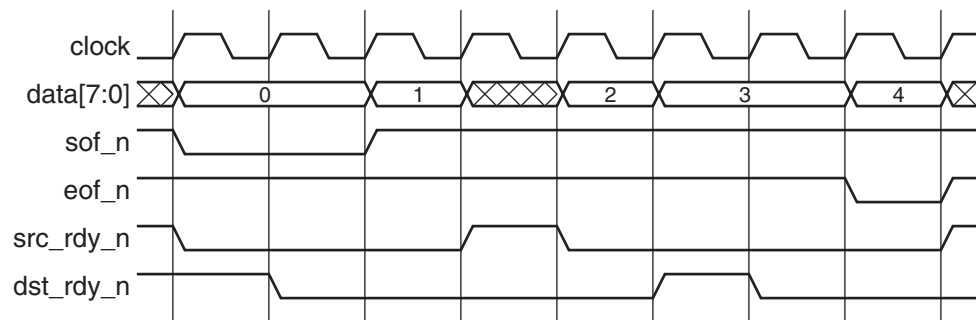


Figure A-3: Frame Transfer with Flow Control

Receive FIFO Operation

The receive FIFO takes data from the client interface of the TEMAC core and converts it into LocalLink format. See “Receiving Inbound Frames,” on page 39 for a description of the TEMAC receive client interface. If the frame is marked as good by the TEMAC, that frame will then be presented on the LocalLink interface for reading by the user. If the frame is marked as bad, that frame will be dropped by the FIFO.

LocalLink Interface

Table A-1 describes the receive FIFO LocalLink interface.

Table A-1: Receive FIFO LocalLink Interface

Signal	Direction	Clock Domain	Description
<code>rx_ll_clock</code>	Input	N/A	Read clock for LocalLink interface
<code>rx_ll_reset</code>	Input	<code>rx_ll_clock</code>	Synchronous reset
<code>rx_ll_data_out[7:0]</code>	Output	<code>rx_ll_clock</code>	Data read from FIFO
<code>rx_ll_sof_out_n</code>	Output	<code>rx_ll_clock</code>	Start of frame indicator

Table A-1: Receive FIFO LocalLink Interface (Continued)

Signal	Direction	Clock Domain	Description
rx_ll_eof_out_n	Output	rx_ll_clock	End of frame indicator
rx_ll_src_rdy_out_n	Output	rx_ll_clock	Source ready indicator
rx_ll_dst_rdy_in_n	Input	rx_ll_clock	Destination ready indicator
rx_fifo_status[3:0]	Output	rx_ll_clock	FIFO memory status

If the receive FIFO memory overflows, the frame currently being received will be dropped, regardless of whether it is a good or bad frame, and the signal `rx_overflow` will be asserted. Frames will continue to be dropped until space is made available in the FIFO by reading data out. The FIFO status signal indicates the occupancy of the FIFO.

Transmit FIFO Operation

The transmit FIFO accepts frames in LocalLink format and stores them in block RAM for transmission via the TEMAC. When a full frame is written into the transmit FIFO, the FIFO will present the data to the TEMAC transmitter client interface. On receiving the `tx_ack` signal from the TEMAC core, the rest of the frame is transmitted. For a description of the TEMAC transmit client interface, see [“Transmitting Outbound Frames,” on page 47](#).

LocalLink Interface

[Table A-2](#) shows the transmit FIFO LocalLink interface signals.

Table A-2: Transmit FIFO LocalLink Interface

Signal	Direction	Clock Domain	Description
tx_ll_clock	Input	N/A	Write clock for LocalLink interface
tx_ll_reset	Input	tx_ll_clock	Synchronous reset
tx_ll_data_in[7:0]	Input	tx_ll_clock	Write data to be sent to transmitter
tx_ll_sof_in_n	Input	tx_ll_clock	Start of frame indicator
tx_ll_eof_in_n	Input	tx_ll_clock	End of frame indicator
tx_ll_src_rdy_in_n	Input	tx_ll_clock	Source ready indicator
tx_ll_dst_rdy_out_n	Output	tx_ll_clock	Destination ready indicator
tx_fifo_status[3:0]	Output	tx_ll_clock	FIFO memory status

In half-duplex operation, if the client interface `emacclienttxcollision` signal is asserted by the TEMAC, the current frame transmission will be terminated. If the `emacclienttxretransmit` signal is also asserted, the FIFO re-queues the frame for transmission.

If the FIFO memory fills up, the `dst_rdy_out_n` signal will be used to halt the LocalLink interface writing in data, until space becomes available in the FIFO. If the FIFO memory fills up but no frames are available for transmission, i.e. if a frame larger than 4000 bytes is

written into the FIFO, the FIFO will assert the `tx_overflow` signal and continue to accept the rest of the frame from the user. The overflow frame will be dropped by the FIFO. This ensures that the LocalLink interface does not lock up.

Clock Requirements

The FIFO has been designed to work with `rxcoreclk` and `txcoreclk` running at speeds in the range of 125 MHz to 1.25 MHz or, with the optional clock enables, `rxgmiimiiclk` and `txgmiimiiclk` in the range of 125 MHz to 2.5 MHz. The `rx_ll_clock` should be no slower than the clock on the receiver client interface. The `tx_ll_clock` should be no slower than the clock on the transmitter client interface divided by 2. It is therefore suggested that the `rx_ll_clock` and `tx_ll_clock` are always 125 MHz or faster.

User Interface Data Width Conversion

Conversion of the user interface 8 bit data path to a 16, 32, 64 or 128 bit data path can be made by connecting the LocalLink interface directly to the Parameterizable LocalLink FIFO ([XAPP691](#)).

Core Verification, Compliance, and Interoperability

The TEMAC has been verified with extensive simulation and hardware verification.

Verification by Simulation

A highly parameterizable transaction-based test bench (not part of the primary core deliverables) was used to test the core. Tests include:

- Register access
- MDIO access
- Frame transmission and error handling
- Frame reception and error handling
- Speed switching
- Address filter operation

Hardware Verification

The core has been tested in a variety of hardware test platforms at Xilinx to cover a variety of parameterizations, including the following:

- Testing with the Ethernet 1000BASE-X PCS/PMA or SGMII cores from Xilinx. A test platform was built around these cores, including a back-end FIFO capable of performing a simple ping function and a test pattern generator. Software running on the embedded PowerPC™ was used to provide access to all configuration, status and statistical counter registers.
- Testing with an external PHY device. The MAC was connected to the external PHY device via the GMII interface.

Core Latency

General

The latency figures given in the following sections may vary by three clock ticks in either direction, due to the crossing of clock domains within the core.

Transmit Path Latency

The transmit path latency is measured by counting the number of clock cycles between a data byte being placed on the client interface (clientemactxd), and it appearing at the GMII/MII output (emacphytxd). At 1 Gbps, this has been measured as 12 clock cycles, at 10/100 Mbps this has been measured as 10 clock cycles.

Receive Path Latency

The receive path latency is measured as the number of clock cycles between a byte being driven onto the GMII/MII receive interface (phyemacrx), and it appearing at the client (emacclientrx). This has been measured as 18 clock cycles at all speeds.

Calculating the DCM Phase Shift

DCM Phase Shifting Requirements

A DCM is used in the receiver clock path to meet the input setup and hold requirements when implementing GMII/MII using the core in Spartan-3, Spartan-3E, and Spartan-3A devices (see [“Implementing External GMII,” on page 63](#)). In RGMII, a DCM is used to maintain the setup and hold times in all devices, except Virtex-5 and Virtex-4 (see [“RGMII Receive Clock Generation,” on page 122](#)).

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 the data is sampled at the optimum time. The ability to shift the internal clock in small increments is critical for sampling high-speed source synchronous signals. 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 GMII 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:

$$\text{System Margin (ps)} = \text{UI(ps)} * (\text{working phase shift range}/128)$$

Finding the Ideal Phase Shift Value

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

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

Perform a complete sweep of phase shift settings during your initial system test. Use only positive (0 to 255) phase shift settings, and use a test range that covers a range of no less than 128, corresponding to a total 180 degrees of clock offset. This does not imply that 128 phase shift values must be tested; increments of 4 (52, 56, 60, and so forth) correspond to

roughly one DCM tap, and consequently provide an appropriate step size. Additionally, it is not necessary to characterize areas outside the working phase shift range.

At the edge of the operating phase shift range, system behavior changes dramatically. In eight phase shift settings or less, the system can transition from no errors to exhibiting errors. Checking the operational edge at a step size of two (on more than one board) refines the typical operational phase shift range. 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.

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