

Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Wrapper v1.8

Getting Started Guide

UG340 March 1, 2011



Xilinx is providing this product documentation, hereinafter "Information," to you "AS IS" with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice.

XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Except as stated herein, none of the Information 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.

© 2006-2011 Xilinx, Inc. Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

	Version	Revision
10/23/06	1.1	Initial Xilinx release.
2/15/07	2.1	Update for version 1.2 of the core; Xilinx tools 9.1i.
5/5/07	3.1	Update for version 1.3 of the core; early access only version.
8/8/07	4.1	Update for full version 1.3 release of the core.
3/24/08	5.1	Update to core version 1.4; Xilinx tools 10.1; Virtex®-5 FXT FPGA support.
9/19/08	6.1	Update to core version 1.5 and Virtex-5 TXT FPGA support.
4/24/09	7.0	Updated to core version 1.6 and Xilinx tools to version 11.1.
4/19/10	7.1	Updated to core version 1.7 and Xilinx tools to version 12.1.
3/1/11	7.2	Updated to core version 1.8 and Xilinx tools to version 13.1

Table of Contents

Schedule of Figures	7
----------------------------------	---

Preface: About This Guide

Guide Contents	9
Conventions	10
Typographical.....	10
Online Document.....	11

Chapter 1: Introduction

System Requirements	13
About the Ethernet MAC Wrapper Core	13
Designs Using RocketIO Transceivers	13
Recommended Design Experience	14
Additional Resources	14
Technical Support	14
Feedback	14
Ethernet MAC Wrapper	14
Document	14

Chapter 2: Licensing the Core

Before you Begin	15
License Options	15
Obtaining Your License Key	15
Installing Your License File	15

Chapter 3: Quick Start Example Design

Overview	17
Generating the Ethernet MAC Wrapper	19
Implementing the Example Design	20
Running the Simulation	20
Functional Simulation	20
Timing Simulation	22
Example Design Detail	22

Chapter 4: Customizing the Core

Ethernet MAC Wrapper Screens	23
Core Configuration Options: Screen 1	24
EMAC Configuration Options: Screen 2	26
EMAC Configuration: Screen 3	28
MDIO/EMAC Configuration: Screen 4	30

Chapter 5: Detailed Example Design

Directory Structure and File Descriptions	33
<project directory>	34
<project directory>/<component name>	34
<component name>/doc	35
<component name>/example_design	35
<component name>/example_design/client	36
<component_name>/example_design/client/fifo	36
<component_name>/example_design/physical	37
<component name>/implement	38
implement/results	38
<component name>/simulation	39
simulation/functional	39
simulation/timing	40
Implementation and Test Scripts	41
Setting up for Simulation	41
Implementation Scripts for Timing Simulation	41
Test Scripts For Timing Simulation	42
Test Scripts For Functional Simulation	43
Example Design	44
HDL Example Design	44
10 Mb/s, 100 Mb/s, 1 Gb/s Ethernet FIFO	45
Address Swap Module	47
Physical Interface	47
Demonstration Test Bench	48
Test Bench Functionality	48
Changing the Test Bench	50

Appendix A: Using the Client Side FIFO

Overview of LocalLink Interface	52
Receive FIFO Operation	53
LocalLink Interface	53
Transmit FIFO Operation	54
LocalLink Interface	54
Clock Requirements	54
User Interface Data Width Conversion	54

Appendix B: Ethernet MAC Clocking

Single-Speed Clocking	55
1000Base-X PCS/PMA: Virtex-5 LXT and SXT Devices	55
1000Base-X PCS/PMA: Virtex-5 FXT and TXT Devices	57
PCS/PMA in Overclocking Mode:	
Virtex-5 LXT, SXT, FXT, and TXT Devices	58
GMII/RGMII at 1000 Mb/s	59
Multi-Speed Clocking	59
SGMII at Multiple Speeds: Virtex-5 LXT and SXT Devices	60
SGMII at Multiple Speeds: Virtex-5 FXT and TXT Devices	61
GMII/MII/RGMII at Multiple Speeds	62
GMII/MII at Multiple Speeds with Clock Enable	65

RGMII at Multiple Speeds with Clock Enable	66
GMII/MII at Multiple Speeds with Byte PHY	66

Appendix C: Constraining the Example Design

Block Level Constraints	69
PCS/PMA/SGMII Clock Constraints	69
GMII/RGMII 1000 Mb/s Clock Constraints	71
GMII/MII/RGMII 10/100/1000 Mb/s Clock Constraints	71
GMII IDELAY_VALUE Constraints	72
RGMII IDELAY_VALUE Constraints	73
LocalLink Level Constraints	74
Example Design Level Constraints	75
GMII/MII Interface	75
RGMII v2.0 Interface	75
Example Placement	75
GMII/RGMII IODELAY Controller Clock Constraint	76
Host Interface Clock Constraint	76
DCR Interface Clock Constraint	76

Appendix D: SGMII Receiver Elastic Buffer

SGMII Capabilities	77
FPGA Fabric Rx Elastic Buffer Requirement	77
The RocketIO Rx Elastic Buffer	79
Jumbo Frame Reception	80

Appendix E: Debugging Designs

Debug Tools	81
Example Design	81
ChipScope Pro Tool	81
Available Reference Boards	82
Link Analyzers	82
Simulation Debug	83
Compiling Simulation Libraries	84
Implementation and Timing Errors	84
Timing Failed for GMII/RGMII/MII OFFSET IN Constraint	84
Hardware Debug	85
General Checks	85
Problems with Transmitting and Receiving Frames	85
Link Bring-up Using 1000BASE-X or SGMII	86
Problems with the MDIO	89
Configuring the Ethernet MAC to the Correct Speed	89

Schedule of Figures

Chapter 1: Introduction

Chapter 2: Licensing the Core

Chapter 3: Quick Start Example Design

Figure 3-1: Default Example Design and Test Bench 18

Figure 3-2: Virtex-5 Embedded Tri-Mode Ethernet MAC Wrapper Main Screen 19

Chapter 4: Customizing the Core

Figure 4-1: Core Configuration Options 24

Figure 4-2: EMAC Configuration Options 26

Figure 4-3: EMAC Configuration Options 28

Figure 4-4: MDIO Configuration 30

Chapter 5: Detailed Example Design

Figure 5-1: HDL Example Design 44

Figure 5-2: Frame Transfer across LocalLink Interface 46

Figure 5-3: Modification of Frame Data by Address Swap Module 47

Figure 5-4: Demonstration Test Bench 48

Appendix A: Using the Client Side FIFO

Figure A-1: Typical 10M/100M/1G Ethernet FIFO Implementation 51

Figure A-2: Frame Transfer across LocalLink Interface 52

Figure A-3: Frame Transfer with Flow Control 52

Appendix B: Ethernet MAC Clocking

Figure B-1: PCS/PMA/SGMII Clocking at 1000 Mb/s: Virtex-5 LXT and SXT 56

Figure B-2: PCS/PMA/SGMII Clocking at 1000 Mb/s: Virtex-5 FXT and TXT Devices . 57

Figure B-3: PCS/PMA Clocking at 2000 Mb/s 58

Figure B-4: GMII/RGMII Clocking at 1000 Mb/s 59

Figure B-5: SGMII Clocking at 10/100/1000 Mb/s: Virtex-5 LXT and SXT Devices 60

Figure B-6: SGMII Clocking at 10/100/1000 Mb/s: Virtex-5 FXT and TXT Devices 61

Figure B-7: RGMII Clocking at 10/100/1000 Mb/s 62

Figure B-8: GMII clocking at 10/100/1000 Mb/s 63

Figure B-9: MII Clocking at 10/100 Mb/s 64

Figure B-10: GMII/MII Clocking at 10/100/1000 Mb/s with Clock Enables 65

Figure B-11: RGMII Clocking at 10/100/1000 Mb/s with Clock Enable 66

Figure B-12: GMII Clocking at 10/100/1000 Mb/s with Byte PHY 67

Figure B-13: MII Clocking at 10/100 Mb/s with Byte PHY 67

Appendix C: Constraining the Example Design

<i>Figure C-1: Input GMII Timing</i>	72
<i>Figure C-2: RGMII Input Timing</i>	73

Appendix D: SGMII Receiver Elastic Buffer

<i>Figure D-1: SGMII Implementation: Separate Clock Sources</i>	78
<i>Figure D-2: SGMII Implementation: Shared Clock Sources</i>	79

Appendix E: Debugging Designs

<i>Figure E-1: Simulation Debug Flow Chart</i>	83
--	----

About This Guide

The *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Wrapper v1.8 Getting Started Guide* provides information about generating an embedded Tri-Mode Ethernet MAC for Virtex®-5 FPGA devices, customizing and simulating the wrapper files utilizing the provided example design, and running the design files through implementation using the Xilinx® tools.

Guide Contents

This guide contains the following chapters:

- [Preface , About This Guide](#) introduces the organization and purpose of this guide and the conventions used in this guide.
- [Chapter 1, Introduction](#) describes the Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC wrapper and related information, including recommended design experience, additional resources, technical support and submitting feedback to Xilinx.
- [Chapter 3, Quick Start Example Design](#) describes how to quickly generate the example design using the CORE Generator™ Graphical User Interface (GUI) software.
- [Chapter 4, Customizing the Core](#) describes the CORE Generator software customization options.
- [Chapter 5, Detailed Example Design](#) provides detailed information about the example design and demonstration test bench.
- [Appendix A, Using the Client Side FIFO](#) describes the operation of the example design client side FIFO.
- [Appendix B, Ethernet MAC Clocking](#) describes the provided clocking scheme for each interface.
- [Appendix C, Constraining the Example Design](#) describes the timing and placement constraints included with the example design.
- [Appendix D, SGMII Receiver Elastic Buffer](#) defines the SGMII capabilities for the core.

Conventions

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

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
Courier font	Messages, prompts, and program files that the system displays. Signal names also.	speed grade: - 100
Courier bold	Literal commands that you enter in a syntactical statement	ngdbuild <i>design_name</i>
Helvetica bold	Commands that you select from a menu	File → Open
	Keyboard shortcuts	Ctrl+C
<i>Italic font</i>	Variables in a syntax statement for which you must supply values	ngdbuild <i>design_name</i>
	References to other manuals	See the <i>User Guide</i> for more information.
	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 [<i>option_name</i>] <i>design_name</i>
Braces { }	A list of items from which you must choose one or more	lowpwr = { on off }
Vertical bar	Separates items in a list of choices	lowpwr = { on off }
Angle brackets < >	User-defined variable or in code samples	<directory name>
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Repetitive material that has been omitted	allow block <i>block_name</i> <i>loc1</i> <i>loc2 ... locn</i> ;

Convention	Meaning or Use	Example
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	usr_teof_n is active low.

Online Document

The following conventions are used in this document:

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

Introduction

This chapter introduces the Virtex®-5 FPGA Embedded Tri-Mode Ethernet MAC wrapper and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx. The Ethernet MAC wrapper supports Verilog HDL and VHDL.

System Requirements

Windows

- Windows XP Professional 32-bit/64-bit
- Windows Vista Business 32-bit/64-bit

Linux

- Red Hat Enterprise WS 4.0 32-bit/64-bit
- Red Hat Enterprise Desktop 5.0 32-bit/64-bit (with Workstation option)
- SUSE Linux Enterprise (SLE) v10.1 32-bit/64-bit

Software

- Xilinx® ISE® 13.1 software

About the Ethernet MAC Wrapper Core

The Ethernet MAC wrapper is included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, visit the Ethernet MAC wrapper [product page](#). The Ethernet MAC wrapper is provided to all licensed Xilinx ISE software customers free of charge and is generated using the Xilinx CORE Generator™ v13.1 software.

Designs Using RocketIO Transceivers

RocketIO™ transceivers are defined by device family in the following way:

- For Virtex-5 LXT and SXT devices, RocketIO GTP transceivers
- For Virtex-5 FXT and TXT devices, RocketIO GTX transceivers

Throughout this guide, the term *RocketIO transceiver* is used to represent any or all of the RocketIO transceivers; select the RocketIO transceiver specific to the desired target device.

Recommended Design Experience

Although the Ethernet MAC wrapper is fully verified, 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 Resources

For additional details and updates, see the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide*, available from www.xilinx.com/support/documentation/virtex-5_user_guides.htm.

Technical Support

The fastest method for obtaining specific technical support for the Ethernet MAC wrapper is through the www.xilinx.com/support website. Questions are routed to a technical support team with specific expertise using the Ethernet MAC wrapper.

Xilinx provides technical support for use of this product as described in the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Data Sheet*, *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Getting Started Guide*, and the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide*. Xilinx does not guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the Ethernet MAC wrapper and the supplied documentation.

Ethernet MAC Wrapper

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

- Product name
- Version number
- Explanation of your comments

Document

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

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

Licensing the Core

In ISE® v13.1 software, a license key is not required for full access to the Virtex®-5 Ethernet MAC Wrapper. However, if you are using ISE 11.4 software or older, follow the instructions below for obtaining a license key before you use the core in your design. The Ethernet wrapper core is provided under the terms of the [Xilinx End User Agreement](#), which conforms to the terms of the SignOnce IP License standard defined by the Common License Consortium.

Before you Begin

This chapter assumes that you have installed the core using either the CORE Generator™ IP Software Update installer, or by performing a manual installation after downloading the core from the web.

For information about installing the core, see the [Ethernet Wrapper Product page](#).

License Options

After installing the required Xilinx ISE software and IP Service Packs, see [Obtaining Your License Key](#) for instructions on how to obtain a full license key.

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

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

Obtaining Your License Key

To obtain a Full license key for ISE v11.4 software or older, follow these instructions. In ISE 13.1 software, the license key requirement has been removed.

1. Navigate to the product page for this core:
www.xilinx.com/products/ipcenter/V5_Embedded_TEMAC_Wrapper.htm
2. Click the “Access Core” link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

After submitting your license key request, you will be sent an email with a full license key, along with instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

Quick Start Example Design

This chapter provides instructions for generating the Ethernet MAC wrapper using the CORE Generator™ GUI.

Overview

The Ethernet MAC wrapper consists of the following:

- A wrapper file that assigns the attributes of each Ethernet MAC to the values selected in the CORE Generator GUI. In addition, unused inputs are tied low and unused outputs are disconnected.
- An example design with a three-level hierarchy:
 - The block-level wrapper instantiates the Ethernet MAC wrapper and the interface logic for each of the selected physical interfaces.
 - The LocalLink wrapper connects the transmit and receive client interfaces of each selected Ethernet MAC to a LocalLink FIFO.
 - The example design wrapper connects the FIFOs so that data received at the client looped back to the transmitter. A small address-swap module is also instantiated to swap the source and destination addresses of the incoming frame. Clock management logic including DCMs and Global Clock Buffer instances, where required, is also included.
- A demonstration test bench to exercise the wrappers and the example design. This injects frames into the physical interface receiver of each selected Ethernet MAC and monitors the data that is output at the transmitter.

Figure 3-1 displays the example design and test bench provided with the Ethernet MAC wrapper. The example design has been tested with Xilinx® ISE® 13.1 software, Cadence Incisive Enterprise Simulator (IES) 10.2, Mentor Graphics ModelSim 6.6d, and Synopsys VCS and VCS MX 2010.06

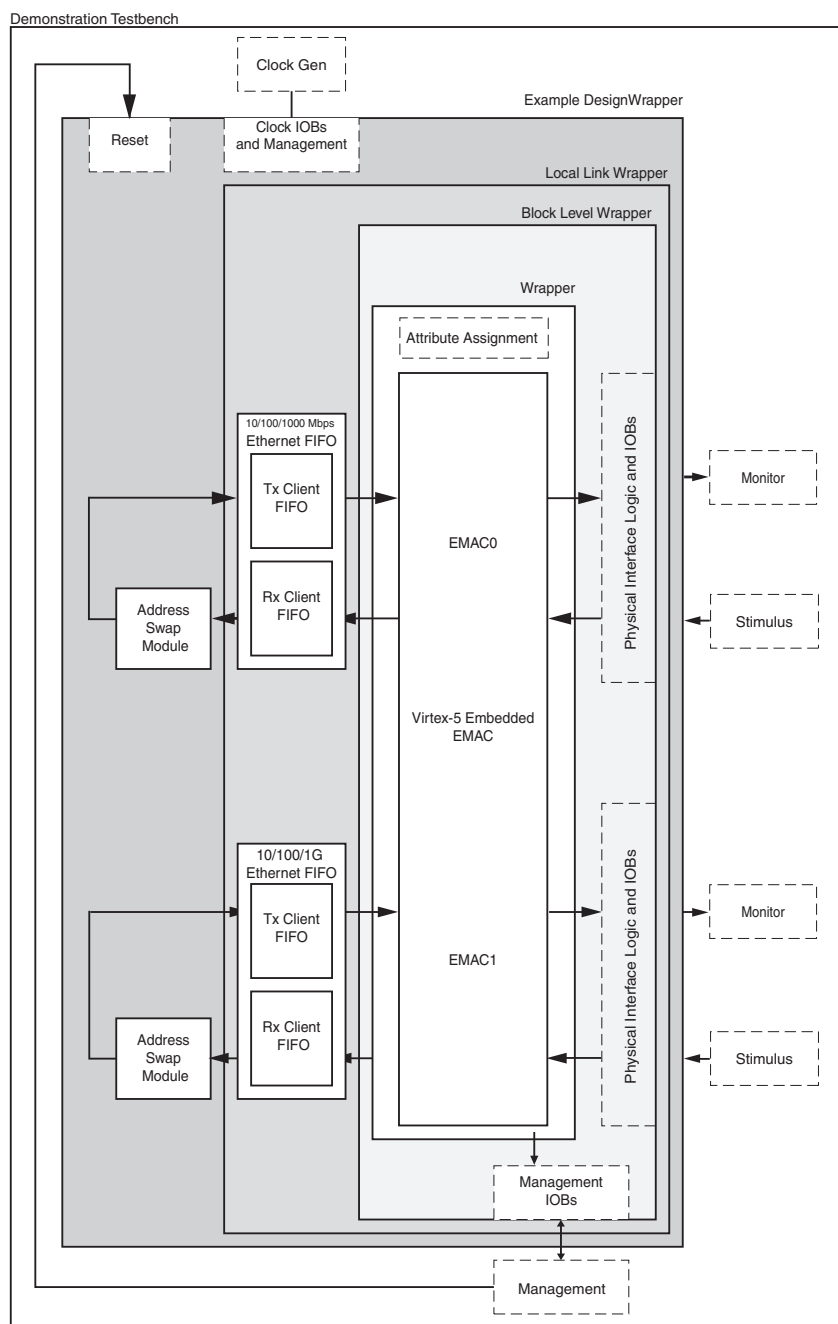


Figure 3-1: Default Example Design and Test Bench

Generating the Ethernet MAC Wrapper

To generate the Ethernet MAC wrapper and example design, do the following:

1. Start the CORE Generator software.
2. Choose File > New Project.
3. Set the following project options:
 - From Target Architecture, select Virtex-5.

Note: If an unsupported silicon family or part is selected, the Ethernet MAC wrapper is not displayed in the taxonomy tree.

 - For Design Entry, select either VHDL or Verilog; for Vendor, select Other.
4. After creating the project, locate the directory containing the Ethernet MAC wrapper in the taxonomy tree. The project appears under one of the following:
 - Communications & Networking /Ethernet
 - Communications & Networking /Networking
 - Communications & Networking /Telecommunications
5. Double-click Virtex[®]-5 Embedded Tri-Mode Ethernet MAC Wrapper. The initial customization screen appears.

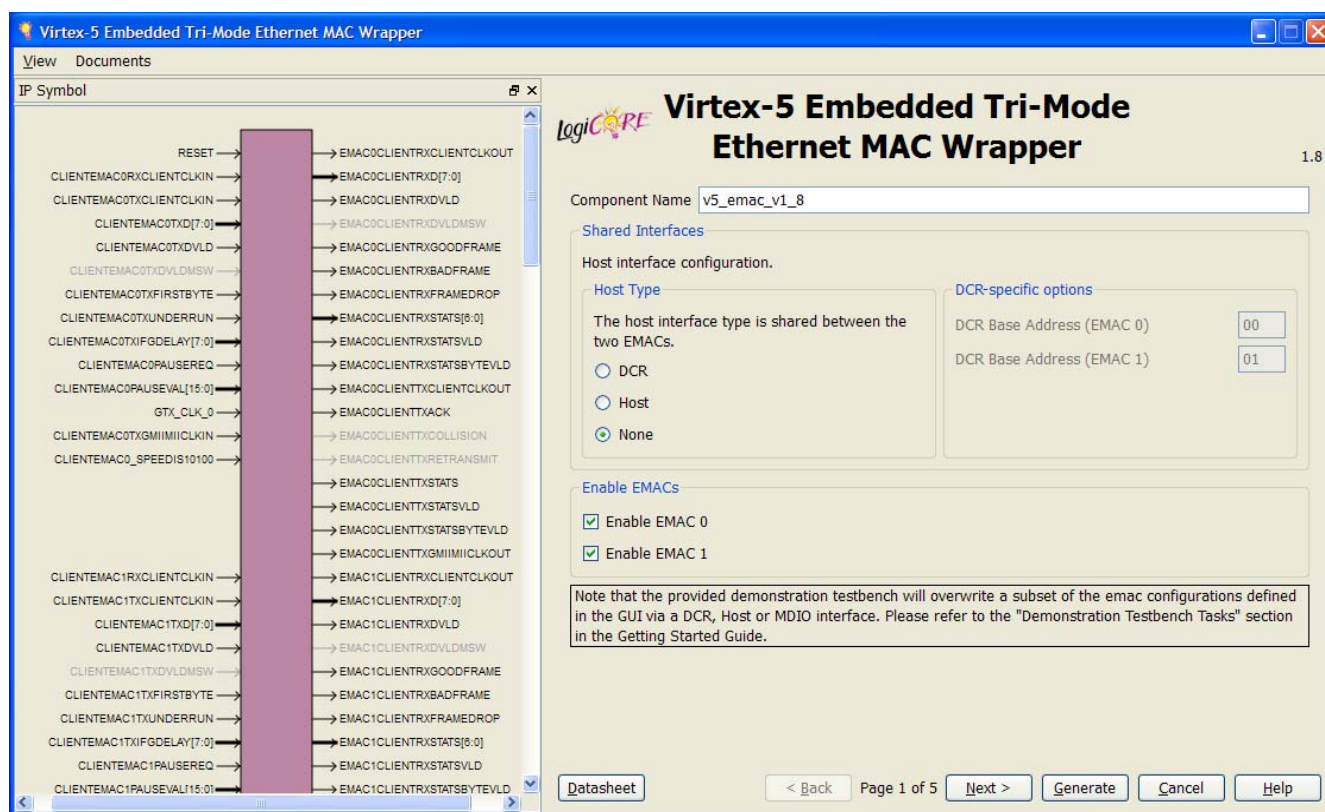


Figure 3-2: Virtex-5 Embedded Tri-Mode Ethernet MAC Wrapper Main Screen

6. In the Component Name field, enter a name for the core instance, and then click Finish to generate the example design using the default values.

The wrapper and its supporting files, including the example design, are generated in your project directory. For a detailed description of the design example files and directories, see [Chapter 5, Detailed Example Design](#).

A functional simulation directory is created that contains scripts to simulate the example design using the structural hdl models. For more information see [Functional Simulation](#), page 20.

Implementing the Example Design

The HDL example design can be processed using the Xilinx implementation toolset. The generated output files include several scripts to assist you in running the Xilinx software.

In the examples that follow, *<project_dir>* is the CORE Generator software project directory and *<component_name>* is the name entered in the Component Name field.

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

For Linux

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

For Windows

```
ms-dos> cd <component_name>\implement
ms-dos> implement.bat
```

These commands execute a script that synthesizes, builds, maps, and place-and-routes the example design. The resulting files are placed in the results directory.

These commands start a script that synthesizes the HDL example design and builds the design. The script also maps and place-and-routes the example design. It then creates gate-level netlist HDL files in both VHDL and Verilog, along with associated timing information (SDF) files.

Running the Simulation

Functional Simulation

To run the functional simulation you must have the Xilinx Simulation Libraries compiled for your system. For more information, see [Compiling Xilinx Simulation Libraries \(COMPXLIB\)](#) in the *Xilinx ISE Synthesis and Verification Design Guide*, which can be obtained from www.xilinx.com/support/software_manuals.htm. In addition, use the following guidelines to determine the simulator required for your design:

Virtex-5 Devices

Virtex-5 device designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator.

Verilog LRM-IEEE 1364-2005 encryption-compliant simulators are:

- ModelSim v6.6d
- Cadence Incisive Enterprise Simulator (IES) 10.2
- Synopsys VCS and VCS MX 2010.06

When running VHDL simulations, a mixed HDL license is required.

In the simulation examples that follow, *<project_dir>* is the CORE Generator software project directory, and *<component_name>* is the component name as entered in the core customization dialog box.

VHDL Simulation

To run a VHDL functional simulation:

- Launch the simulator and set the current directory to
<project_dir>/<component_name>/simulation/functional
- For ModelSim, map the UniSim and SecureIP library:

```
ModelSim> vmap unisim <path to compiled libraries>/unisim
ModelSim> vmap secureip <path to compiled libraries>/secureip
```
- Launch the simulation script:

```
ModelSim> do simulate_mti.do
IES> ./simulate_ncsim.sh
VCS> ./simulate_vcs.sh
```

The scripts compile the example design files and the demonstration test bench, add some relevant signals to a wave window, then run the simulation to completion. At this point, you can review the simulation transcript and waveform to observe the operation of the Ethernet MACs.

Verilog Simulation

To run a Verilog functional simulation:

- Launch the simulator and set the current directory to
<project_dir>/<component_name>/simulation/functional
- For ModelSim, map the UniSim and SecureIP library:

```
ModelSim> vmap unisim_ver <path to compiled libraries>/unisim_ver
ModelSim> vmap secureip <path to compiled libraries>/secureip
```
- Launch the simulation script:

```
ModelSim> do simulate_mti.do
IES> ./simulate_ncsim.sh
VCS> ./simulate_vcs.sh
```

The scripts compile the example design files and the demonstration test bench, add some relevant signals to a wave window, then run the simulation to completion. At this point, you can review the simulation transcript and waveform to observe the operation of the Ethernet MACs.

Timing Simulation

To run the gate-level simulation you must have the Xilinx Simulation Libraries compiled for your system. For more information, see *Compiling Xilinx Simulation Libraries (COMPXLIB)* in the *Xilinx ISE Synthesis and Verification Design Guide*, which can be obtained from www.xilinx.com/support/software_manuals.htm.

In the simulation examples that follow, *<project_dir>* is the CORE Generator software project directory; *<component_name>* is the component name as entered in the core customization dialog box.

VHDL Simulation

To run a VHDL timing simulation:

- Launch the simulator and set the current directory to `<project_dir>/<component_name>/simulation/timing`
- For ModelSim, map the SimPrim and SecureIP library:


```
ModelSim> vmap simprim <path to compiled libraries>/simprim
ModelSim> vmap secureip <path to compiled libraries>/secureip
```
- Launch the simulation script:


```
ModelSim> do simulate_mti.do
IES> ./simulate_ncsim.sh
```

The scripts compile the gate-level model and the demonstration test bench, add some relevant signals to a wave window, then run the simulation to completion. At this point, you can review the simulation transcript and waveform to observe the operation of the Ethernet MACs.

Verilog Simulation

To run a Verilog timing simulation:

- Launch the ModelSim simulator and set the current directory to `<project_dir>/<component_name>/simulation/timing`
- For ModelSim, map the SimPrim and SecureIP library:


```
ModelSim> vmap simprims_ver <path to compiled libraries>/simprims_ver
ModelSim> vmap secureip <path to compiled libraries>/secureip
```
- Launch the simulation script:


```
ModelSim> do simulate_mti.do
IES> ./simulate_ncsim.sh
```

The scripts compile the gate-level model and the demonstration test bench, add some relevant signals to a wave window, then run the simulation to completion. At this point, you can review the simulation transcript and waveform to observe the operation of the Ethernet MACs.

Example Design Detail

For detailed information about the example design, including guidelines for modifying the design and extending the test bench, see [Chapter 5, Detailed Example Design](#).

Customizing the Core

This chapter describes Virtex®-5 FPGA Embedded Tri-Mode Ethernet MAC Wrapper GUI to customize the functions of the core.

Ethernet MAC Wrapper Screens

The Ethernet MAC Wrapper GUI consists of several screens. The first screen is used to set core parameters and enable one or both Ethernet MACs. Subsequent screens are used to configure all enabled EMACs. If both EMACs are enabled, the subsequent screens are displayed twice - once each for each enabled EMAC.

- **Core Configuration Options: Screen 1.** Used to name the core, select the desired interface, and enable the number of EMACs.
- **EMAC Configuration Options: Screen 2.** Used to select the PHY interface, speed, data width, global buffer usage, management data (MDIO) bus enable, and flow control configuration for the specified EMAC. If both EMACs are enabled, this screen is displayed twice; once for each enabled EMAC.
- **EMAC Configuration: Screen 3.** Used to set transmitter, receiver, and address filter configuration. If both EMACs are enabled, this screen is displayed twice; once for each enabled EMAC.
- **MDIO/EMAC Configuration: Screen 4.** This screen is only displayed if the Enable Management Data (MDIO) option is selected on the first screen.

Core Configuration Options: Screen 1

Use the initial configuration screen to define the core name, select options for shared interfaces and host type, and enable one or both EMACs.

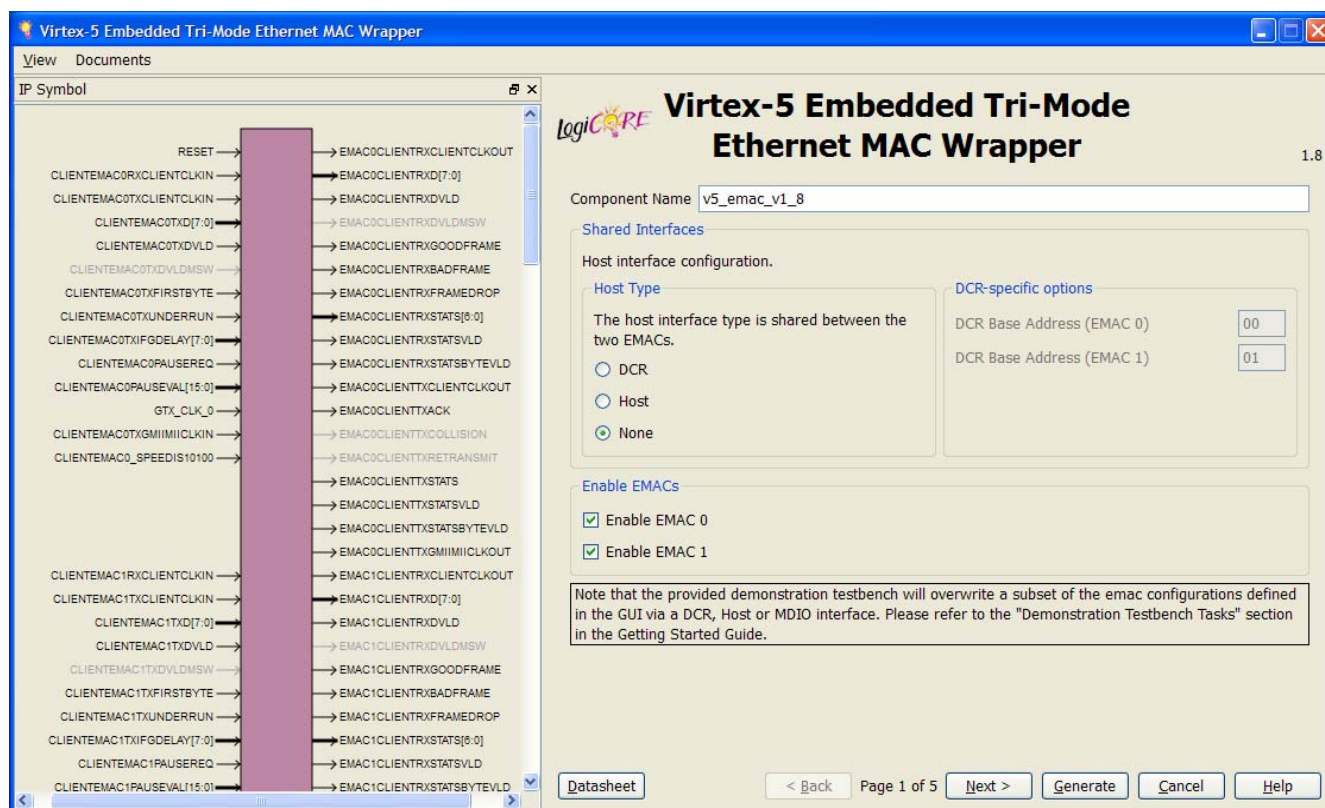


Figure 4-1: Core Configuration Options

Component Name

Enter the base name of the output files generated for the core. The name must begin with a letter and be composed of the following characters: a to z, 0 to 9, and “_.”

Host Type

Select the core host bus interface in one of the following ways:

- **Device Control Registers (DCR).** Accesses the configuration registers through DCR interface. When the DCR bus is used to access the internal registers of the Ethernet MAC, the DCR bus bridge in the host interface translates commands carried over the DCR bus into Ethernet MAC host bus signals. The resulting signals are input into one of the Ethernet MACs.
- **Host.** Accesses the Host Interface through the fabric. When the generic host bus is used, the HOSTEMAC1SEL signal selects either the host access of EMAC0 or EMAC1. When HOSTEMAC1SEL is asserted, the host accesses EMAC1. HOSTEMAC1SEL acts as the host address bit 10. If only one Ethernet MAC is used, this signal can be tied off to use either one of the Ethernet MACs during the power-up FPGA configuration.
- **None.** The Ethernet MACs are configured using attributes set depending on the configuration options selected in the GUI, and are loaded into the Ethernet MACs on power-up or when reset is asserted. If None is selected, the transmit and receive engines must be enabled to ensure proper operation of the Ethernet MAC.

Enable EMACs

Select one or both to enable one or both EMACs; at least one EMAC must be enabled to generate a core. In this chapter, the EMAC configuration screens (screens 2, 3, and 4) define options for EMAC 0 only. If EMAC 1 is also enabled, an additional set of configuration screens appear for EMAC 1 after configuration of EMAC 0 is complete.

DCR-specific Options

EMAC 0 and **EMAC 1.** Enter a unique address for each enabled EMAC in the DCR Base Address field.

EMAC Configuration Options: Screen 2

The EMAC configuration screen allows you to determine the Physical (Phy) interface, speed, data width, global buffer usage, and management data (MDIO) bus enable for the specified EMAC. Some options on this screen are dimmed depending on the Phy Interface selected; not all options are available with all Phy interface types.

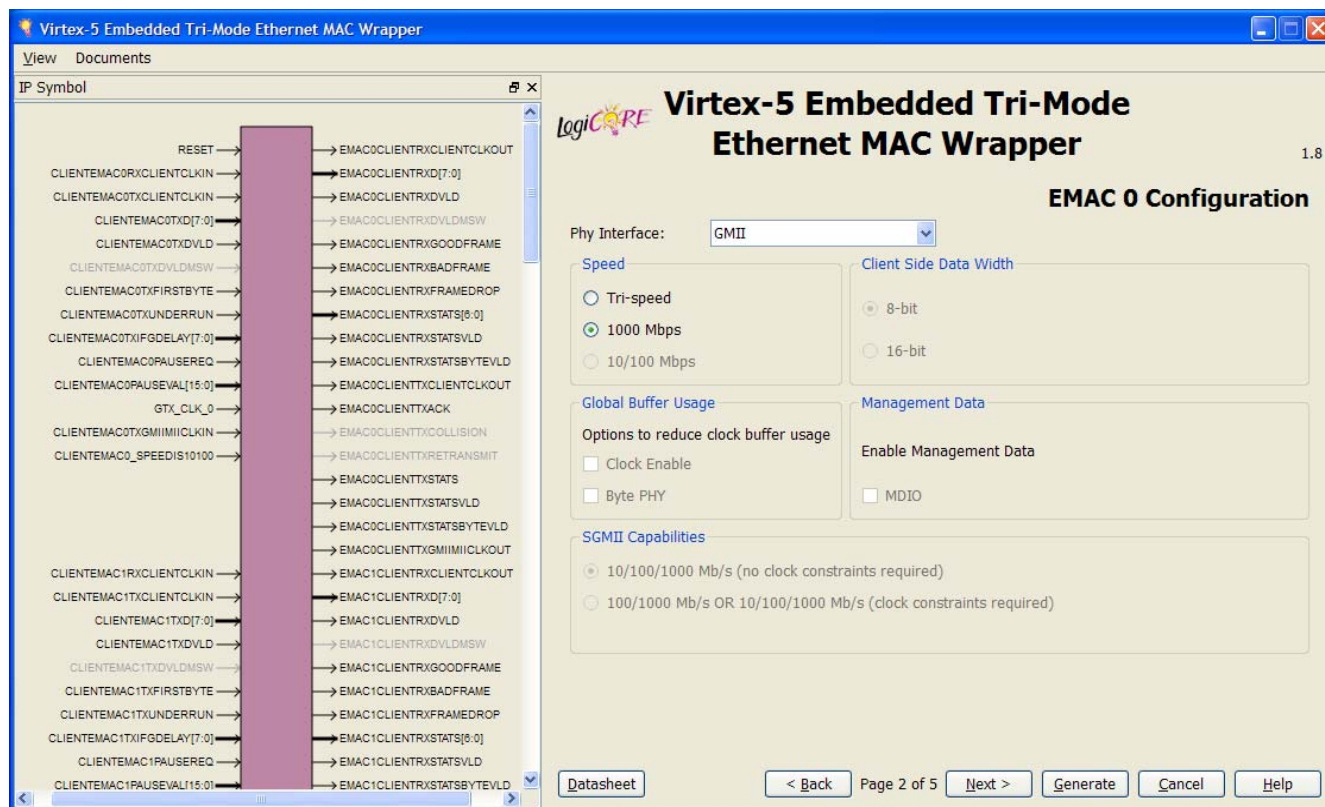


Figure 4-2: EMAC Configuration Options

PHY Interface

Select the Phy interface type from the drop-down list:

- MII
- GMII
- RGMII v1.3
- RGMII v2.0
- SGMII
- 1000BASE-X PCS/PMA

Speed

Configures the core to run at a single or tri-speed rate.

- **Tri-speed.** Configures the core to run at a tri-speed rate.
- **1000 Mb/s.** Configures the core to run at a single rate.
- **10/100 Mb/s.** Configures the core to run at 10 or 100 Mb/s.

Client Side Data Width

- **8-bit.** An 8-bit data width is available for all interface types.
- **16-bit.** A 16-bit client interface is available for the 100BASE-X PCS/PMA interface, which enables the EMAC to operate at 250 MHz, while the logic in the FPGA fabric is clocked at 125 MHz. The 16-bit option yields a 2.5 Gb/s line rate.

Global Buffer Usage

- **Clock Enable.** Selecting Clock Enable reduces the number of BUFs by requiring the user logic to use a separate clock-enable signal. See the *Virtex-5 FPGA Tri-Mode Ethernet MAC User Guide* for information about determining the clock-enable signal setup. This option is available for 10 or 100 Mb/s operation using the MII interface as well as for Tri-speed operation in GMII and RGMII modes.
- **Byte PHY.** In Tri-Speed GMII mode, selecting Byte PHY reduces the number of BUFs by adding the Byte PHY to the physical side logic. See the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide* for more information about Byte PHY mode.

Management Data

MDIO. When selected, the MDIO option enables the MDIO ports on the core to access the registers in the internal and external PHY. When the MDIO option is selected for one or both EMACs, an MDIO configuration screen appears (for each EMAC) before generating the core. When unselected, the MDIO configuration screen is not displayed.

SGMII Capabilities

- **10/100/1000 Mb/s (no clock constraints required).** Default setting; provides the implementation using the Receiver Elastic Buffer in FPGA fabric. This alternative Receiver Elastic Buffer utilizes a single block RAM to create a buffer twice as large as the one present in the RocketIO™ transceiver, subsequently consuming extra logic resources. However, this default mode provides reliable SGMII operation under all conditions.
- **10/100/1000 Mb/s OR 100/1000 Mb/s (clock constraints required).** Uses the receiver elastic buffer present in the RocketIO transceivers. This is half the size and can potentially under- or overflow during SGMII frame reception at 10 Mb/s operation. However, there are logical implementations where this can be proven reliable: if so it is favored because of its lower logic utilization.

For detailed information about SGMII capabilities, see [Appendix D, SGMII Receiver Elastic Buffer](#).

EMAC Configuration: Screen 3

The next EMAC configuration screen defines the configuration of each EMAC. For each enabled EMAC, a separate screen is provided, with the selected EMAC displayed at the top of the screen.

Some of these configurations are overwritten when running simulation using the demonstration test bench. See [Demonstration Test Bench Tasks](#), page 49 for more information.

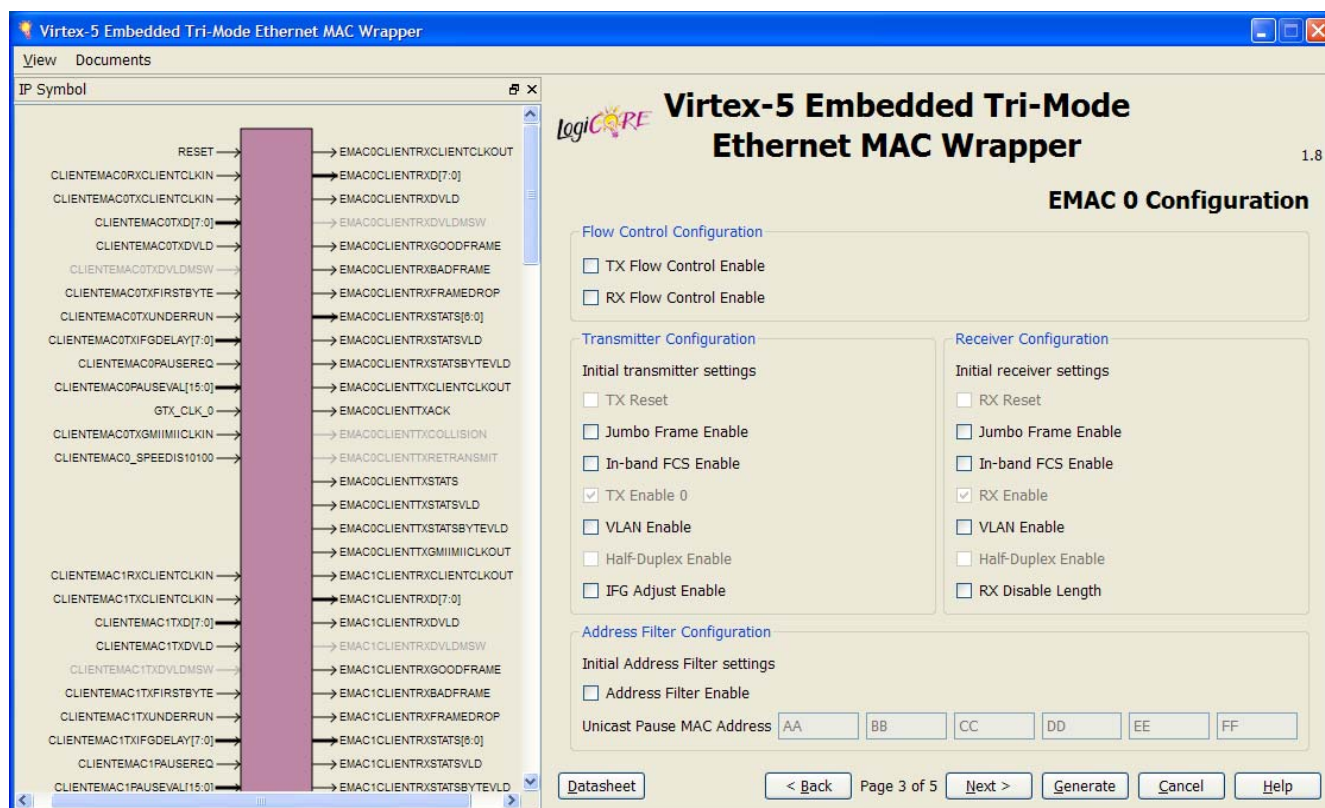


Figure 4-3: EMAC Configuration Options

Flow Control Configuration

Allows both the receive and transmit flow control to be enabled or disabled. Flow control is disabled by default.

- **Tx Flow Control Enable.** Enable transmit flow control.
- **Rx Flow Control Enable.** Enable receive flow control.

Transmitter Configuration

Transmitter configuration refers to the Ethernet MAC configuration registers located at 0x280. Initial values for several bits of this register can be set using the GUI. Changes to the register bits can be written using one of the host interfaces, if enabled. For more information, see *Configuration Registers*, in the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

- **TX Reset.** When Host type is set to None, the Initial value of this bit cannot be changed.
- **Jumbo Frame Enable.** When selected, the transmitter sends frames greater than the maximum length specified in the *IEEE Std 802.3-2002*. When unselected, the transmitter sends only frames less than the specified maximum length.
- **In-band FCS Enable.** When selected, this bit sets the Ethernet MAC transmitter to be ready for the FCS field from the client.
- **TX Enable 0.** When Host type is set to None, the Initial value of this bit cannot be changed.
- **VLAN Enable.** When selected, the VLAN transmitter allows transmission of the VLAN-tagged frames.
- **Half-Duplex Enable.** When selected, the transmitter operates in half-duplex mode (applicable only for 10 and 100 Mb/s). When unselected, the transmitter operates in full-duplex mode.
- **IFG Adjust Enable.** When selected, the transmitter reads the value of `CLIENTEMAC#TXIFGDELAY` at the start of frame transmission and adjusts the IFG.

Receiver Configuration

Receiver configuration refers to the Ethernet MAC configuration registers located at 0x240. Initial values for several bits of this register can be set using the GUI. Changes to the register bits can be written using one of the host interfaces, if enabled. For more information, see *Configuration Registers*, in the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

- **RX Reset.** When Host type is set to None, the Initial value of this bit cannot be changed.
- **Jumbo Frame Enable.** When selected, the Ethernet MAC receiver accepts frames over the maximum length specified in the *IEEE Std 802.3-2002* specification. When unselected, the receiver accepts only frames up to the specified maximum.
- **In-band FCS Enable.** When selected, the receiver passes the FCS field up to the client. When unselected, the FCS field is not passed to the client. In either case, the FCS is verified on the frame.
- **RX Enable.** When Host type is set to None, the Initial value of this bit cannot be changed.
- **VLAN Enable.** When selected, the receiver accepts VLAN tagged frames. The maximum payload length increases by four bytes.
- **Half-Duplex Enable.** When selected, the receiver operates in half-duplex mode. When unselected, the receiver operates in full-duplex mode.
- **RX Disable Length.** When selected, disables the Length/Type field check on the frame.

Address Filter Configuration

The Unicast Pause MAC Address (entered by the user) is used by the EMAC to compare the destination address of any incoming flow control frames, and as the source address for any outbound flow control frames.

The address is ordered for the least significant byte in the register to have the first byte transmitted or received, for example, an EMAC address of AA-BB-CC-DD-EE-FF is entered as FF-EE-DD-CC-BB-AA.

MDIO/EMAC Configuration: Screen 4

The MDIO Configuration screen is only displayed if the 1000BASE-X PCS/PMA or SGMII PHY interface is selected and the Enable Management Data (MDIO) option is selected in the [Management Data](#) section of the first EMAC configuration screen.

Some of these configurations are overwritten when running simulation using the demonstration test bench. See [Demonstration Test Bench Tasks](#), page 49 for more information.

If both EMACs are enabled identically, the screen appears twice; if only one EMAC uses the 1000BASE-X PCS/PMA or SGMII PHY interface and MDIO option, the screen appears only once for the enabled EMAC.

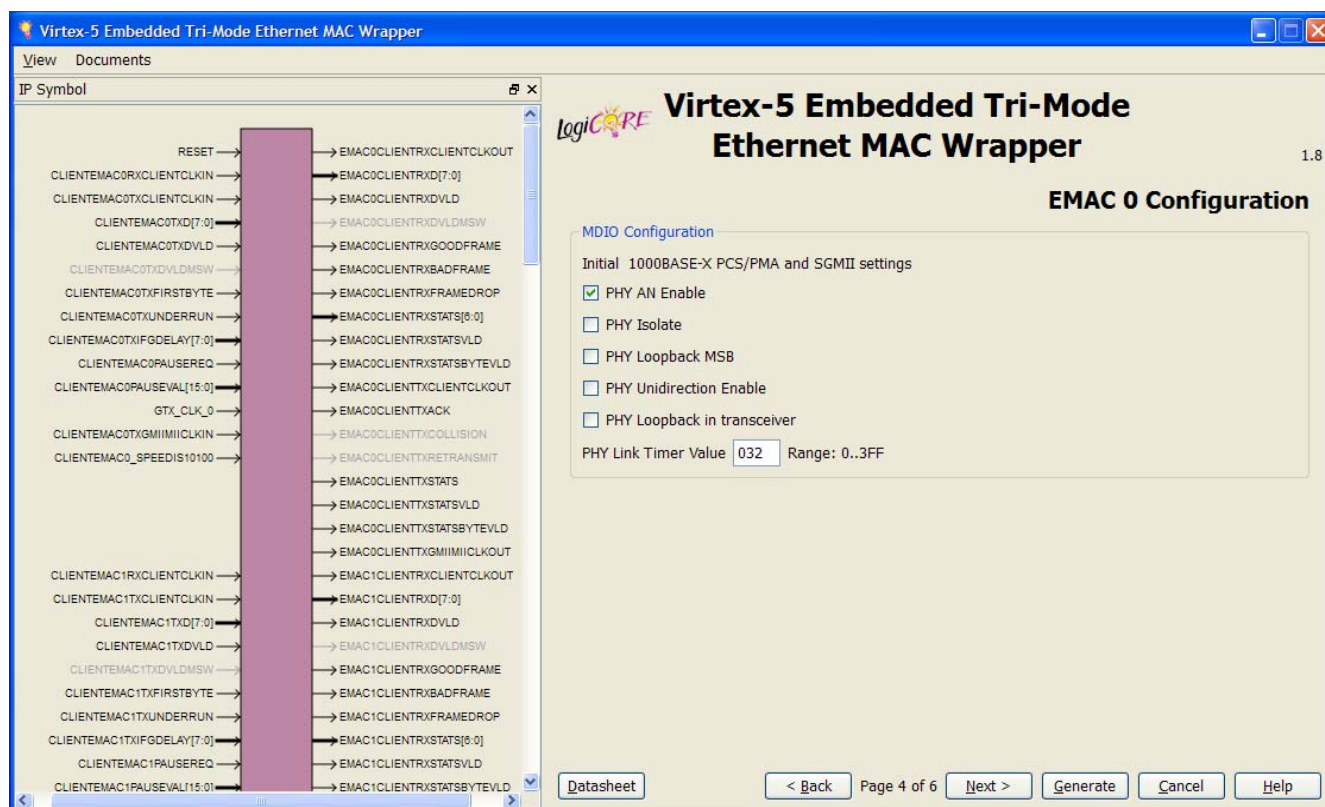


Figure 4-4: MDIO Configuration

MDIO Configuration

- **PHY AN Enable.** If selected, auto-negotiation is enabled.
- **PHY Isolate.** If selected, the PHY is electrically isolated.
- **PHY Loopback MSB.** If selected, the PHY loopback is enabled.
- **PHY Unidirection Enable.** If selected, the PHY is capable of transmitting data regardless of whether a valid link has been established.
- **PHY Loopback in Transceiver.** If selected, loopback occurs in the RocketIO transceiver. Otherwise loopback occurs in the Ethernet MAC.
- **PHY Link Timer Value.** Programmable auto negotiation link timer value.













Several default MDIO configurations, for example PHY Reset and PHY Powerdown, needs to be set manually in the EMAC wrapper and cannot be configured via the GUI.

Detailed Example Design

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

Directory Structure and File Descriptions

The Virtex®-5 FPGA Embedded Tri-Mode Ethernet MAC core directories and their associated files are defined in the sections that follow. To go to a specific directory, click one of the following links.

-  [<project directory>](#)
Top-level project directory; user-defined name
 -  [<project directory>/<component name>](#)
Core release notes file
 -  [<component name>/doc](#)
Product documentation
 -  [<component name>/example_design](#)
Verilog and VHDL (or whichever, if it is only one) design files
 -  [<component name>/example_design/client](#)
Support files for the example client loopback logic
 -  [<component_name>/example_design/client/fifo](#)
Files for the FIFO instances in the LocalLink client
 -  [<component_name>/example_design/physical](#)
Files that describe the physical interfaces of the Ethernet MAC
 -  [<component name>/implement](#)
Implementation script files
 -  [implement/results](#)
Results directory, created after implementation scripts are run, and contains implement script results
 -  [<component name>/simulation](#)
Test bench HDL (Verilog or VHDL)
 -  [simulation/functional](#)
Functional simulation scripts
 -  [simulation/timing](#)
Timing simulation scripts

<project directory>

The <project directory> contains all the CORE Generator software project files.

Table 5-1: Project Directory

Name	Description
<project_dir>	
<component_name>.xco	As an output file, the XCO file is a log file which records the settings used to generate a particular instance of the Ethernet MAC wrapper. An XCO file is generated by the CORE Generator System for each core that it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator software.
<component_name>_flist.txt	A text file listing all the output files produced when the wrapper and example design files were generated in the CORE Generator software.

[Directory Structure and File Descriptions](#)

<project directory>/<component name>

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

Table 5-2: Component Name Directory

Name	Description
<project_dir>/<component_name>	
v5_emac_readme.txt	Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Wrapper release notes text file.

[Directory Structure and File Descriptions](#)

<component name>/doc

The doc directory contains Ethernet MAC documentation. For detailed information about the Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC, see the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide*, available from www.xilinx.com/bvdocs/userguides/ug194.pdf.

Table 5-3: Doc Directory

Name	Description
<project_dir>/<component_name>/doc	
v5_emac_ds550.pdf	Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Wrapper Data Sheet
v5_emac_gsg340.pdf	Virtex-5 Embedded Tri-Mode Ethernet MAC Wrapper Getting Started Guide

[Directory Structure and File Descriptions](#)

<component name>/example_design

The example design directory and its subdirectories contain the support files necessary for a Verilog or VHDL implementation of the example design. See [Example Design, page 44](#) for more information. The main Embedded Ethernet MAC Wrapper file and the top-level file for the example design are contained in this directory.

Table 5-4: Example Design Directory

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>.v[hd]	Ethernet MAC wrapper file.
<component_name>_block.v[hd]	Block-level Ethernet MAC wrapper with instantiation of physical interface circuitry.
<component_name>_locallink.v[hd]	Top-level example design with a LocalLink client interface provided by the instantiation of the receive and transmit FIFOs.
<component_name>_example_design.v[hd]	Top-level example design providing a simple loopback function and clock buffer instantiation.
<component_name>_example_design.ucf	UCF for the design. See Appendix C, Constraining the Example Design for more information.

[Directory Structure and File Descriptions](#)

<component_name>/example_design/client

This directory contains the support files necessary for the example client loopback logic, which is connected to the Ethernet MAC client interfaces. The 8-bit versions of the following files are only present when an 8-bit client interface is selected. Similarly, the 16-bit versions are only present when a 16-bit client interface is selected.

Table 5-5: Example Design Directory

Name	Description
<project_dir>/<component_name>/example_design/client	
address_swap_module_[8 16].v[hd]	The client loopback instances this to swap the source and destination addresses of the incoming frames.

[Directory Structure and File Descriptions](#)

<component_name>/example_design/client/fifo

This directory contains the files for the FIFO instantiated in the LocalLink client wrapper. For more information about the FIFO see [10 Mb/s, 100 Mb/s, 1 Gb/s Ethernet FIFO, page 45](#).

Table 5-6: Example Design Directory

Name	Description
<project_dir>/<component_name>/example_design/client/fifo	
eth_fifo_[8 16].v[hd]	The FIFO top level, which instantiates the transmit and receive client FIFOs.
tx_client_fifo_[8 16].v[hd]	The transmit client FIFO. Takes data from the client in LocalLink format, stores it, and sends it to the MAC.
rx_client_fifo_[8 16].v[hd]	The receive client FIFO. Reads in and stores data from the MAC before outputting it to the client in LocalLink format.

[Directory Structure and File Descriptions](#)

<component_name>/example_design/physical

This directory contains the files that describe the physical interfaces of the Ethernet MAC. Appropriate files are delivered by the CORE Generator software depending on the options selected.

Table 5-7: Example Design Directory

Name	Description
<project_dir>/<component_name>/example_design/physical	
fcs_blk_mii.v[hd]	Generated if the MII or tri-speed GMII physical interface is selected for the Ethernet MAC. Assures correct FCS transmission.
fcs_blk_rgmii.v[hd]	Generated if the 10/100 Mb/s or tri-speed RGMII physical interface is selected for the Ethernet MAC. Assures correct FCS transmission.
gmii_if.v[hd]	If GMII is selected on one or both Ethernet MACs without the Advanced Clocking option (Byte PHY).
gmii_byte_phy_if.v[hd]	If GMII is selected on one or both Ethernet MACs with the Byte PHY Advanced Clocking option.
mii_if.v[hd]	If MII is selected on one or both of the Ethernet MACs.
mii_byte_phy_if.v[hd]	If MII is selected on one or both Ethernet MACs with the Byte PHY Advanced Clocking option.
rgmii_if.v[hd]	If RGMII version 1.3 is selected on one or both of the Ethernet MACs.
rgmii_v2_0_if.v[hd]	If RGMII version 2.0 is selected on one or both of the Ethernet MACs.
gtp_dual_1000X.v[hd]	If a Virtex-5 LXT or SXT device is targeted and a SGMII or 1000Base-X PCS/PMA interface is selected on one or both Ethernet MACs, these files collectively connect the RocketIO™ GTP transceivers to the physical interface.
gtx_dual_1000X.v[hd]	If a Virtex-5 FXT or TXT device is targeted and an SGMII or 1000Base-X PCS/PMA interface is selected on one or both Ethernet MACs, these files collectively connect the RocketIO GTX transceivers to the physical interface.

Table 5-7: Example Design Directory (Cont'd)

Name	Description
<code>rx_elastic_buffer.v[hd]</code>	If the Tri-speed SGMII interface and SGMII Capabilities 10/100/1000 Mb/s (no clock constraints required) options are selected (Screen 2 of the GUI), the clock correction has to be implemented in the fabric to prevent buffer errors from occurring in long frames at 10 Mb/s. This file implements a clock correction buffer using a RAMB18.

[Directory Structure and File Descriptions](#)

<component name>/implement

The implement directory contains the core implementation script files.

Table 5-8: Implement Directory

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

[Directory Structure and File Descriptions](#)

implement/results

The results directory is created by the implement scripts and is used to run the example design files and the Ethernet MAC wrapper file through the Xilinx implementation tools. After these scripts are run, timing simulation files appear in the directory.

Table 5-9: Results Directory

Name	Description
<code><project_dir>/<component_name>/implement/results</code>	
<code>routed.v[hd]</code>	The back-annotated SimPrim based Verilog or VHDL design. Used for timing simulation.
<code>routed.sdf</code>	The timing information for simulation is contained in this file.

[Directory Structure and File Descriptions](#)

<component name>/simulation

The simulation directory and its subdirectories provide the files necessary to test a Verilog or VHDL implementation of the example design.

Table 5-10: Simulation Directory

Name	Description
<project_dir>/<component_name>/simulation	
demo_tb.v[hd]	The Verilog or VHDL demonstration test bench for the Ethernet MAC wrapper.
configuration_tb.v[hd]	The configuration test bench is instantiated in demo_tb.vhd. It provides stimuli to configure the Ethernet MACs via the selected management interface.
emac0_phy_tb.v[hd]	The physical interface test bench for EMAC0. This stimulates the receiver ports and monitors the transmitter ports of the EMAC0 physical interface. This is instantiated in demo_tb.vhd and is only present when EMAC0 is selected.
emac1_phy_tb.v[hd]	The physical interface test bench for EMAC1. This stimulates the receiver ports and monitors the transmitter ports of the EMAC1 physical interface. This is instantiated in demo_tb.vhd and is only present when EMAC1 is selected.

[Directory Structure and File Descriptions](#)

simulation/functional

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

Table 5-11: Functional Directory

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_mti.do	A ModelSim macro file that compiles the example design sources and the structural simulation model then runs the functional simulation to completion.
wave_mti.do	A ModelSim macro file that opens a wave windows and adds interesting signals to it. It is called used by the simulate_mti.do macro file.
simulate_ncsim.sh	An IES script file that compiles the example design sources and the structural simulation model and then runs the functional simulation to completion.

Table 5-11: Functional Directory (Cont'd)

Name	Description
wave_ncsim.sv	An IES macro file that opens a wave window and adds interesting signals to it.
simulate_vcs.sh	VCS script file that compiles the Verilog sources and runs the simulation to completion.
ucli_commands.key	The file sourced by VCS at the start of simulation; it configures the simulator.
vcs_session.tcl	VCS macro file that opens a wave window and adds signals of interest. It is called by the simulate_vcs.sh script file.

[Directory Structure and File Descriptions](#)

simulation/timing

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

Table 5-12: Timing Directory

Name	Description
<project_dir>/<component_name>/simulation/timing	
simulate_mti.do	A ModelSim macro file that compiles the Verilog or VHDL timing model and demo test bench then runs the timing simulation to completion.
wave_mti.do	A ModelSim macro file that opens a wave windows and adds interesting signals to it. It is called used by the simulate_mti.do macro file.
simulate_ncsim.sh	An IES script file that compiles the Verilog or VHDL timing model and demo test bench and then runs the timing simulation to completion.
wave_ncsim.sv	An IES macro file that opens a wave window and adds interesting signals to it.
simulate_vcs.sh	VCS script file that compiles the Verilog timing model and runs the simulation to completion.
ucli_commands.key	The file sourced by VCS at the start of simulation; it configures the simulator.
vcs_session.tcl	VCS macro file that opens a wave window and adds signals of interest. It is called by the simulate_vcs.sh script file.

[Directory Structure and File Descriptions](#)

Implementation and Test Scripts

Setting up for Simulation

The Xilinx® UniSim and SecureIP libraries must be mapped into the simulator. If the UniSim and SecureIP libraries are not set up for your environment, go to [Answer Record 15338](#) for assistance compiling Xilinx simulation models and for setting up the simulator environment.

Virtex-5 Device Requirements

Virtex-5 device designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator.

Verilog LRM-IEEE 1364-2005 encryption-compliant simulators are:

- ModelSim v6.6d
- Cadence Incisive Enterprise Simulator (IES) v10.2
- Synopsys VCS and VCS MX 2010.06

When running VHDL simulations, a mixed HDL license is required.

Implementation Scripts for Timing Simulation

The implementation script, generated in the implement directory, is either a shell script or batch file that processes the example design through the Xilinx tool flow.

```
<project_dir>/<component_name>/implement
```

[Figure 5-1](#) shows a block diagram of the design.

Linux

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

Windows

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

The implement script performs the following steps:

1. The HDL example design is synthesised using XST.
2. NGDbuild is run to produce an NGD file containing the entire design. A constraints file is also used at this stage to constrain the clocks to operate at the correct speed for Ethernet implementations. This file also contains constraints to control any clock domain crossings present in the design and example pin placements where appropriate.
For detailed information about the constraints files, see [Appendix C, Constraining the Example Design](#).
3. The design is placed-and-routed on the target device.
4. Static timing analysis is performed on the routed design using **trce**.
5. A bitstream is generated.
6. Netgen runs on the routed design to generate Verilog and VHDL netlists and timing information in the form of SDF files.

The Xilinx tool flow generates several output and report files. These files are saved in the following directory which is created by the implement script:

```
<project_dir>/<component_name>/implement/results
```

Test Scripts For Timing Simulation

The test script macro that automates the simulation of the test bench. The test scripts do the following:

- Compile the gate-level netlist
- Compile the demonstration test bench
- Start a simulation of the test bench
- Open a Wave window and adds some signals of interest (**wave_mti.do**, **wave_ncsim.sv**, **vcs_session.tcl**)
- Run the simulation to completion

For ModelSim

Verilog

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

VHDL

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

For IES

Verilog

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

VHDL

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

For VCS

Verilog

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

Test Scripts For Functional Simulation

The test script that automates the functional simulation of the test bench. The test scripts do the following:

- Compile the Ethernet MAC wrapper
- Compile the example design files
- Compile the demonstration test bench
- Start a simulation of the test bench with no timing information
- Open a Wave window and adds some signals of interest (**wave_mti.do**, **wave_ncsim.sv**, **vcs_session.tcl**)
- Run the simulation to completion

For ModelSim

Verilog

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

VHDL

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

For IES

Verilog

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

VHDL

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

For VCS

Verilog

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

Example Design

HDL Example Design

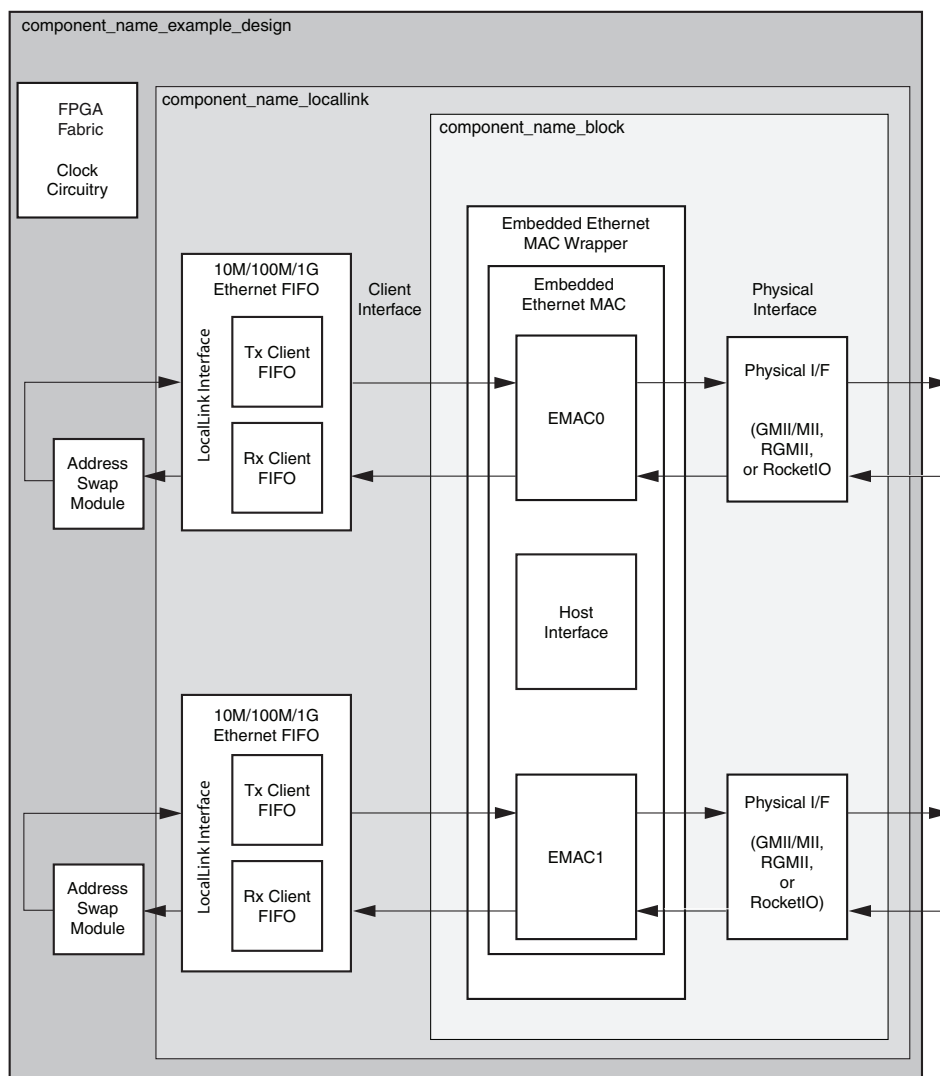


Figure 5-1: HDL Example Design

The top-level example design for the Ethernet MAC wrapper is defined in the following files:

Verilog

```
<project_dir>/<component_name>/example_design/  
<component_name>_example_design.v
```

VHDL

```
<project_dir>/<component_name>/example_design/  
<component_name>_example_design.vhd
```

The HDL example design contains the following:

- An instance of the Ethernet MAC wrapper
- An instance of the block level EMAC wrapper containing GMII/MII, RGMII, SGMII or 1000Base-X PCS/PMA interface logic
- An instance of the LocalLink wrapper containing transmit and receive LocalLink FIFOs
- An instance of the top-level example design containing an address swap module, which loops the received data back to the transmitter. Clock management logic including DCMs and Global Clock Buffer instances where required, is also instantiated. This allows the functionality of the core to be demonstrated either using a simulation package, as discussed in this guide, or in hardware, if placed on a suitable board

10 Mb/s, 100 Mb/s, 1 Gb/s Ethernet FIFO

The 10 Mb/s, 100 Mb/s, 1 Gb/s Ethernet FIFO is defined in the following files:

Verilog

```
<project_dir>/<component_name>/example_design/client/fifo/  
eth_fifo_[8/16/8,16].v  
<project_dir>/<component_name>/example_design/client/fifo/  
tx_client_fifo_[8/16/8,16].v  
<project_dir>/<component_name>/example_design/client/fifo/  
rx_client_fifo_[8/16/8,16].v
```

VHDL

```
<project_dir>/<component_name>/example_design/client/fifo/  
eth_fifo_[8/16/8,16].vhd  
<project_dir>/<component_name>/example_design/client/fifo/  
tx_client_fifo_[8/16/8,16].vhd  
<project_dir>/<component_name>/example_design/client/fifo/  
rx_client_fifo_[8/16/8,16].vhd
```

The 10 Mb/s, 100 Mb/s, 1 Gb/s Ethernet FIFO contains an instance of `tx_client_fifo` to connect to the Ethernet MAC client side transmitter interface, and an instance of the `rx_client_fifo` to connect to the Ethernet MAC client receiver interface. Both transmit and receive FIFO components implement a LocalLink user interface, through which the frame data can be read and written.

Figure 5-2 illustrates a simple frame transfer across the LocalLink. For more information about the FIFO, see [Appendix A, Using the Client Side FIFO](#).

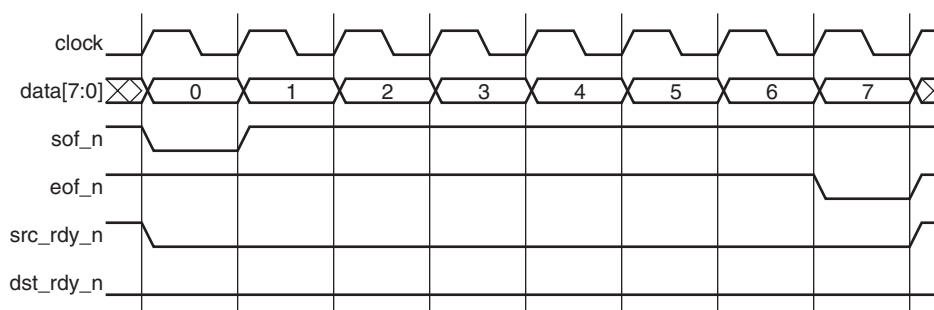


Figure 5-2: Frame Transfer across LocalLink Interface

rx_client_fifo

The `rx_client_fifo` is built around 2 Dual Port block RAMs, providing a total memory capacity of 4096 bytes of frame data. The receive FIFO writes in data received through the Ethernet MAC. If the frame is marked as good, that frame is presented on the LocalLink interface for reading by the user, (in this case the `tx_client_fifo` module). If the frame is marked as bad, that frame is dropped by the receive FIFO.

If the receive FIFO memory overflows, the frame currently being received is dropped, regardless of whether it is a good or bad frame, and the signal `rx_overflow` is asserted. Situations in which the memory can overflow are:

- The FIFO can overflow if the receiver clock is running at a faster rate than the transmitter clock or if the inter-packet gap between the received frames is smaller than the inter-packet gap between the transmitted frames. If this is the case, the Tx fifo cannot read data from the rx fifo as fast as it is being received.
- The FIFO size of 4096 bytes limits the size of the frames that it can store without error. If a frame is larger than 4000 bytes, the FIFO can overflow and data lost. For this reason, it is recommended that the example design not be used with the Ethernet MAC in jumbo frame mode for frames larger than 4000 bytes.

tx_client_fifo

The `tx_client_fifo` is built around 2 Dual Port block RAMs, providing a total memory capacity of 4096 bytes of frame data. When a full frame has been written into the transmit FIFO, the FIFO presents data to the MAC transmitter. On receiving the acknowledge signal from the Ethernet MAC, the rest of the frame is transmitted providing there is no retransmit request output by the Ethernet MAC. If a retransmission request is received, the frame is queued for retransmission.

If the FIFO memory fills to capacity, the `dst_rdy_out_n` signal is used to halt the LocalLink interface writing data until space becomes available in the FIFO. If the FIFO memory fills but no full frames are available for transmission, that is, if a frame larger than 4000 bytes is written into the FIFO, the FIFO asserts `tx_overflow` and continues to accept the rest of the frame from the user. The overflow frame is dropped by the FIFO to ensure that the LocalLink interface does not lock up.

Address Swap Module

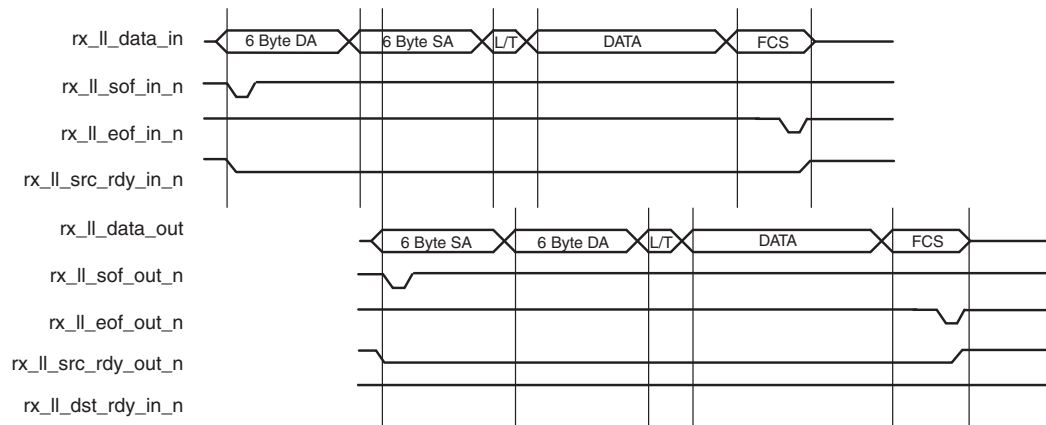


Figure 5-3: Modification of Frame Data by Address Swap Module

The address swap module is described in the following files:

Verilog

```
<project_dir>/<component_name>/example_design/client/  
address_swap_module_[8|16|8,16].v
```

VHDL

```
<project_dir>/<component_name>/example_design/client/  
address_swap_module_[8|16|8,16].vhd
```

The address swap module takes frame data from the Ethernet MAC LocalLink client interface. The module swaps the destination and source addresses of each frame (as shown in Figure 5-3) to ensure that the outgoing frame destination address matches the source address of the link partner. The module transmits the frame control signals with an equal latency to the frame data.

Physical Interface

An appropriate Physical Interface is provided for each selected EMAC0/EMAC1. This connects the physical interface of the Ethernet MAC block to the I/O of the FPGA, and as required, contains the following components:

- For GMII/MII, this component contains Input/Output block (IOB) buffers and IOB flip-flops.
- For RGMII, this component contains IOB buffers and IOB Double-Data Rate flip-flops. IODELAYs are also instantiated on the receiver data input. These are configured in *FIXED* mode and align the received data with the clock. If RGMII v2.0 is selected, an IODELAY is used to delay the transmitter clock output by the 2ns required by the specification.
- For 1000BASE-X PCS/PMA or SGMII, this component instantiates and connects the RocketIO transceivers.

Demonstration Test Bench

Test Bench Functionality

The demonstration test bench, illustrated in Figure 5-4, is a simple VHDL or Verilog program to exercise the example design and the core itself.

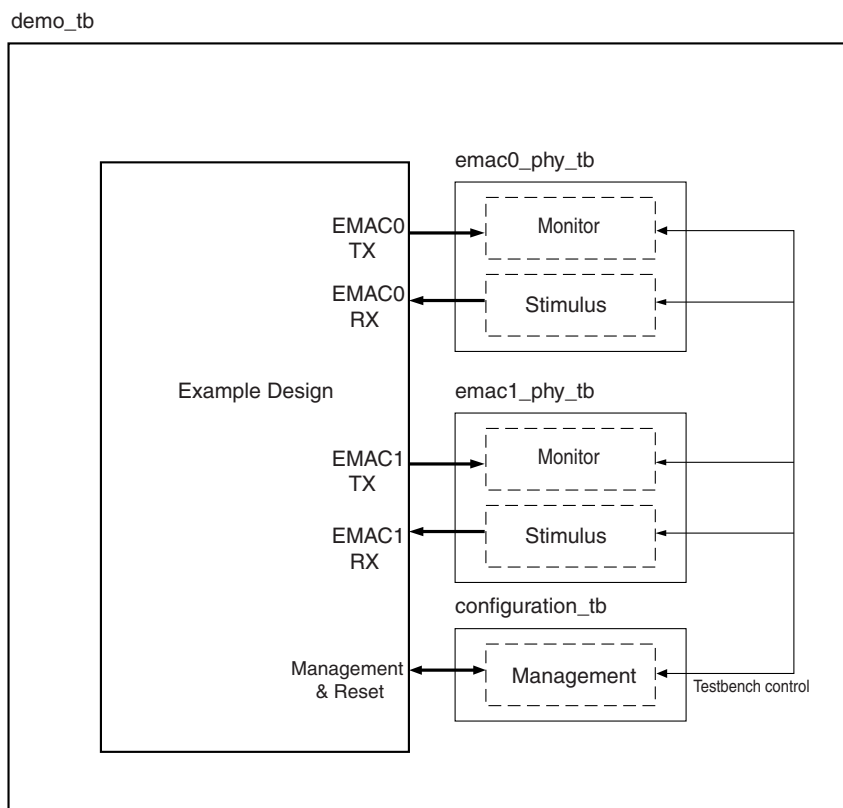


Figure 5-4: Demonstration Test Bench

The demonstration test bench is defined in the following files:

Verilog

```
<project_dir>/<component_name>/simulation/demo_tb.v
<project_dir>/<component_name>/simulation/configuration_tb.v
<project_dir>/<component_name>/simulation/emac0_phy_tb.v
<project_dir>/<component_name>/simulation/emac1_phy_tb.v
```

VHDL

```
<project_dir>/<component_name>/simulation/demo_tb.vhd
<project_dir>/<component_name>/simulation/configuration_tb.vhd
<project_dir>/<component_name>/simulation/emac0_phy_tb.vhd
<project_dir>/<component_name>/simulation/emac1_phy_tb.vhd
```


The top-level test bench (`demo_tb.vhd`, `demo_tb.v`) consists of the following:

- Clock generators
- A control mechanism to manage the interaction of management, stimulus, and monitor blocks.

The configuration test bench (`configuration_tb.vhd`, `configuration_tb.v`) consists of the following:

- A management block to exercise the host or DCR interfaces (if selected) or to configure the Ethernet MACs through the configuration vector
- Semaphores to indicate configuration status to the top level test bench

The physical layer test benches (`emac0/1_phy_tb.vhd`, `emac0/1_phy_tb.v`) *each* consist of the following:

- A stimulus block, which connects to the physical receiver interface of the example design
- A monitor block to check data returned through the physical transmitter interface

Demonstration Test Bench Tasks

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The selected Ethernet MACs are configured through the management or configuration interface. The settings are:
 - Define the MDC clock frequency
 - Disable auto-negotiation in SGMII and 1000Base-X PCS/PMA modes (overwriting GUI configurations)
 - Disable flow control (overwriting GUI configurations)
 - Disable receiver and transmit reset configuration registers (overwriting GUI configurations)
 - Disable receiver and transmit half duplex mode (overwriting GUI configurations)
 - Disable Phy isolate feature for SGMII and 1000Base-X PCS/PMA modes (overwriting GUI configurations)
 - Disable the transceiver and EMAC loopback feature for SGMII and 1000Base-X PCS/PMA modes (overwriting GUI configurations)
 - Disable the unidirectional enable bit in PCS/PMA management register for SGMII and 1000Base-X PCS/PMA modes (overwriting GUI configurations)
 - Enable transmitter and receiver
- The configuration test bench then sets the speed of the selected Ethernet MACs.
- If EMAC0 is selected to run at 1000 Mb/s or in Tri-Speed mode, the following four frames are pushed into the EMAC0 receiver interface at 1 Gb/s:
 - The first frame is a minimum length frame
 - The second frame is a type frame
 - The third frame is an errored frame
 - The fourth frame is a padded frame

- If EMAC1 is selected to run at 1000 Mb/s or in Tri-Speed mode, the same four frames are applied to the EMAC1 receiver interface simultaneously.
- The frames received at the transmitter of each Ethernet MAC interface are checked against the stimulus frames to ensure data is the same. The monitor process takes into account the source/destination address field and FCS modifications resulting from the address swap module.
- If applicable, the selected Ethernet MACs are configured through the management interface to run at 100 Mb/s. The same four frames are then sent to the receiver interface and checked against the stimulus frames.
- If applicable, the selected Ethernet MACs are then configured through the management interface to run at 10 Mb/s. The same four frames are then sent to the receiver interface and checked against the stimulus frames.

Changing the Test Bench

Changing Frame Data

The contents of the frame data passed into the Ethernet MAC receivers can be modified by changing the DATA fields for each frame defined in the test bench. More frames can be added by defining a new frame of data.

Changing Frame Error Status

Errors can be inserted into any of the pre-defined frames by changing the ERROR field to '1' in any column of that frame. When an error is introduced into a frame, the BAD_FRAME field for that frame must be set in order to disable the monitor checking for that frame. The error currently written into the third frame can be removed by setting all ERROR fields for the frame to '0' and unsetting the BAD_FRAME field.

Changing the Tri-Mode Ethernet MAC Configuration

The configuration of the Ethernet MACs used in the demonstration test bench can be altered.

Caution: Certain Ethernet MAC configurations cause the test bench either to result in failure or cause processes to run indefinitely. Be sure to determine which configurations can be used safely with the test bench.

The Ethernet MACs can be reconfigured by adding more steps in the test bench management process to write new configurations to the Ethernet MAC.

Using the Client Side FIFO

The example design provided with the Ethernet MAC wrapper contains a LocalLink FIFO used to interface to the client side of the Ethernet MAC. The source code for the FIFO is provided and can be used and adjusted for user applications.

The 10 Mb/s, 100 Mb/s, 1 Gb/s 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 2 Dual Port block RAMs providing a memory capacity of 4096 bytes in each FIFO.

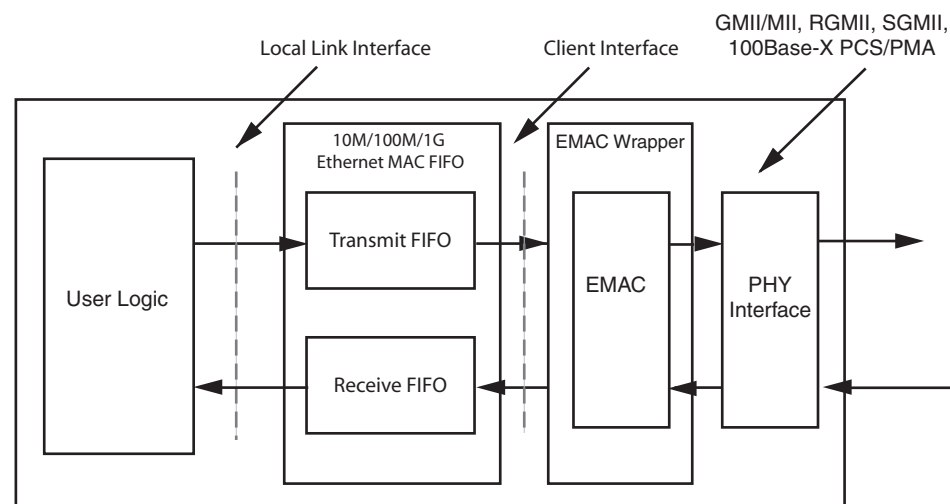


Figure A-1: Typical 10M/100M/1G Ethernet FIFO Implementation

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](#). [Figure A-2](#) shows the transfer of an 8-byte frame.

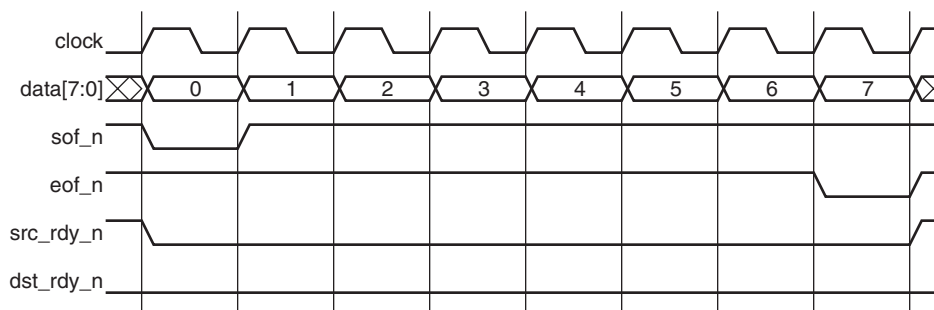


Figure A-2: Frame Transfer across LocalLink Interface

[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.

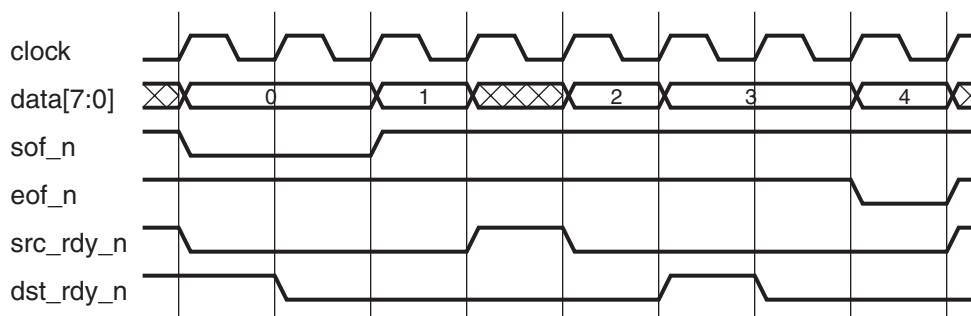


Figure A-3: Frame Transfer with Flow Control

Receive FIFO Operation

The receive FIFO takes data from the client interface of the Ethernet MAC core and converts it into LocalLink format. See the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide* for a description of the Ethernet MAC receive client interface. If the frame is marked as good by the Ethernet MAC, that frame is presented on the LocalLink interface for reading by the user. If the frame is marked as bad, that frame is 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
rx_ll_clock	Input	N/A	Read clock for LocalLink interface
rx_ll_reset	Input	rx_ll_clock	Synchronous reset
rx_ll_data_out[7:0]	Output	rx_ll_clock	Data read from FIFO
rx_ll_sof_out_n	Output	rx_ll_clock	Start of frame indicator
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 is dropped, regardless of its status (good or bad), and the `rx_overflow` is asserted. Frames 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 through the EMAC. When a full frame is written into the transmit FIFO, the FIFO presents the data to the Ethernet MAC transmitter client interface. On receiving the acknowledge signal from the Ethernet MAC, the rest of the frame is transmitted. For a description of the Ethernet MAC transmit client interface, see the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

LocalLink Interface

Table A-2 defines 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 collision signal is asserted by the EMAC, the current frame transmission is terminated. If the retransmit signal is also asserted, the FIFO re-queues the frame for transmission.

If the FIFO memory fills to capacity, the `dst_rdy_out_n` signal is used to halt the LocalLink interface writing in data until space becomes available in the FIFO. If the FIFO memory fills to capacity but no frames are available for transmission, that is, if a frame larger than 4000 bytes is written into the FIFO, the FIFO asserts the `tx_overflow` signal and continues to accept the rest of the frame from the user. The overflow frame is dropped by the FIFO to ensure that the LocalLink interface does not lock up.

Clock Requirements

The FIFO is designed to work with the client clocks running at speeds in the range of 125 MHz to 1.25 MHz. The `rx_ll_clock` should be no slower than the clock on the receiver client interface and the `tx_ll_clock` should be no slower than the clock on the transmitter client interface. For this reason, it is 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 datapath to a 16, 32, 64, or 128 bit datapath can be made by connecting the LocalLink interface directly to the Parameterizable LocalLink FIFO ([XAPP691](#)).

Ethernet MAC Clocking

The Ethernet MAC example design provides clocking schemes for each supported interface. This chapter provides details about the supplied clocking schemes. Clocking is implemented in the `<component_name>_example_design.v/vhd` file. For more information about Ethernet MAC clock management, see the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide*. In the following examples, # refers to the Ethernet MAC number (EMAC0 or EMAC1).

Single-Speed Clocking

1000Base-X PCS/PMA: Virtex-5 LXT and SXT Devices

In PCS/PMA and SGMII modes ([Figure B-1](#)) the user supplies a high quality differential clock to the RocketIO™ GTP transceiver. This is input to the wrappers on the CLK_DS port. This clock can be shared between multiple instantiations of the wrappers.

A 125 MHz clock is used to drive the transmit and receive sections of the wrappers. This is supplied via a BUFG and input on the CLK125 port. The REFCLKOUT output from the transceiver is made available to the user and can be used to drive CLK125. This clock can be shared between multiple instantiations of the wrappers.



1000Base-X PCS/PMA: Virtex-5 FXT and TXT Devices

The RocketIO transceiver in Virtex®-5 FXT and TXT devices requires an additional 62.5 MHz clock input. This drives the transceiver 2-byte internal datapath. This clock should be generated from REFCLKOUT via a DCM as illustrated in [Figure B-2](#).

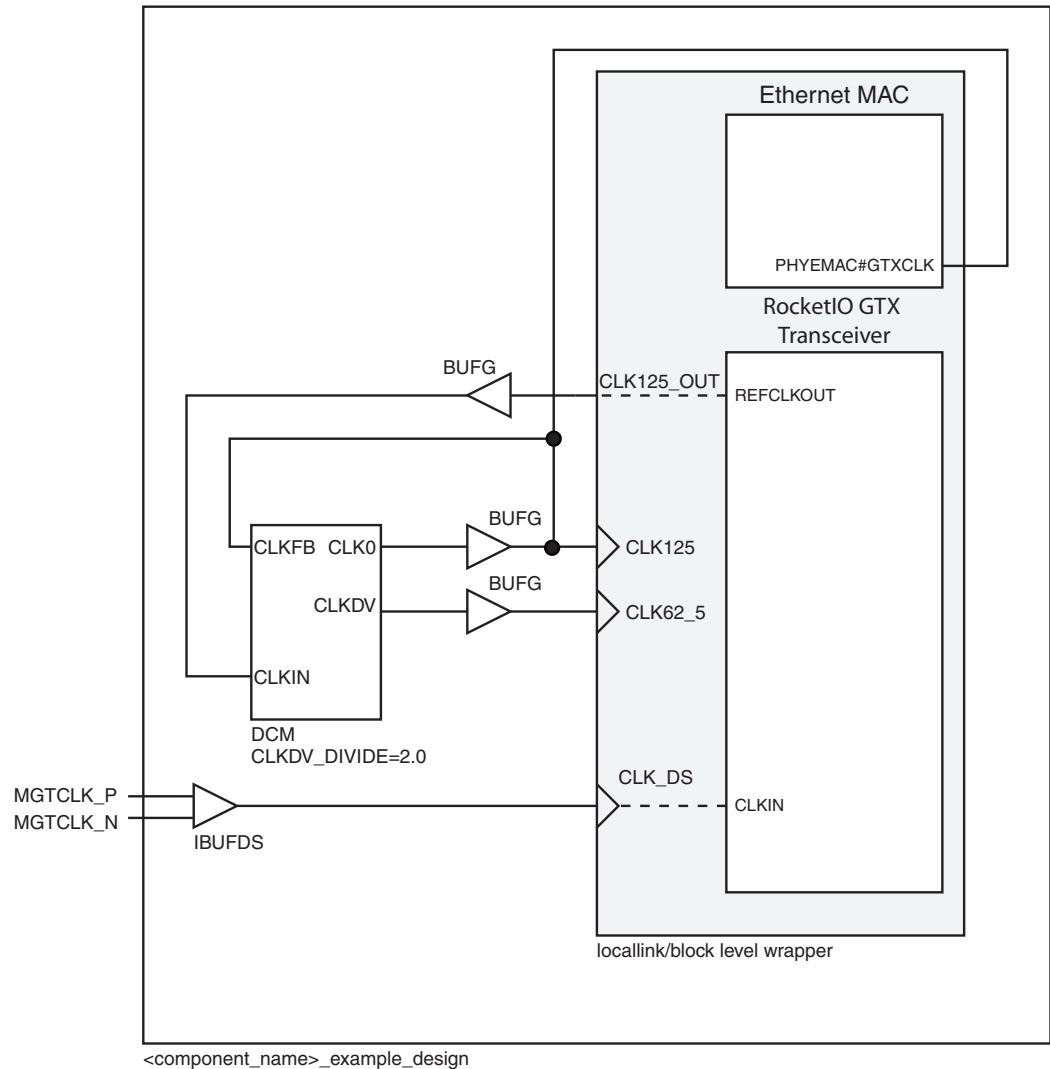


Figure B-2: PCS/PMA/SGMII Clocking at 1000 Mb/s: Virtex-5 FXT and TXT Devices

GMII/RGMII at 1000 Mb/s

Figure B-4 shows the clocking for a parallel interface (GMII or RGMII) operating at 1000 Mb/s. TX_CLK_# clocks the transmitter circuitry and is driven by a high quality 125 MHz clock. This can be shared between multiple instantiations of the wrappers.

The receiver is driven by the input clock from the PHY chip via an IDELAY element. The IDELAY is used to align the clock to the data inputs as they enter the FPGA. The delayed clock is routed through a BUFG (or BUFR) and cannot be shared between multiple Ethernet MACs.

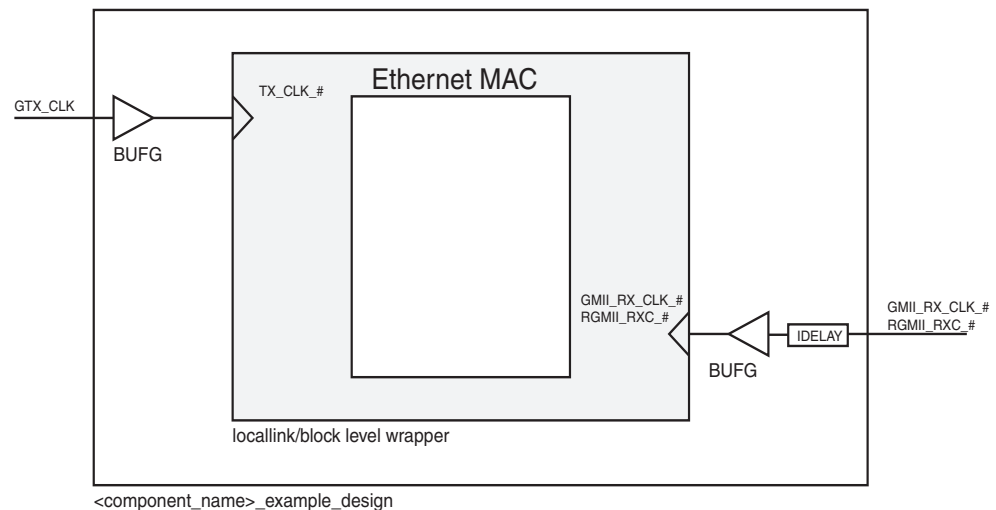


Figure B-4: GMII/RGMII Clocking at 1000 Mb/s

Multi-Speed Clocking

This section illustrates the clocking for the Ethernet MAC wrapper when operating at multiple speeds. For most interfaces, the Ethernet MAC supplies a clock output that can be used to drive the client logic at all speeds. The frequency of the clock output is dependent on the setting of the speed selection bits in the Ethernet MAC Mode Configuration Register.

For the implementation of some interfaces the Ethernet MAC EMAC#SPEEDIS10100 output is exposed at the wrapper interface. This is asserted when the Ethernet MAC is operating at 10 or 100 Mb/s. This signal can be used to select between different clock inputs depending on the current speed of operation. For information about the Ethernet MAC signals and register definitions see the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

SGMII at Multiple Speeds: Virtex-5 LXT and SXT Devices

For SGMII operation at multiple speeds the user supplies the high quality differential clock (CLK_DS) and 125 MHz reference clock (CLK125) described in [Figure B-1](#). These are used to clock the RocketIO transceiver and the physical side of the Ethernet MAC and can be shared between multiple instantiations of the wrappers.

In addition a 1.25/12.5/125 MHz client clock (CLIENT_CLK_#) must be supplied. This is used to drive the client logic at all 3 speeds. The EMAC#CLIENTTXCLIENTCLKOUT output from the EMAC is made available to the user on the CLIENT_CLK_OUT_# port and this is used to drive CLIENT_CLK_#. This clock cannot be shared between multiple Ethernet MACs unless they all operate at the same speed. [Figure B-5](#) shows the supplied clocking scheme.

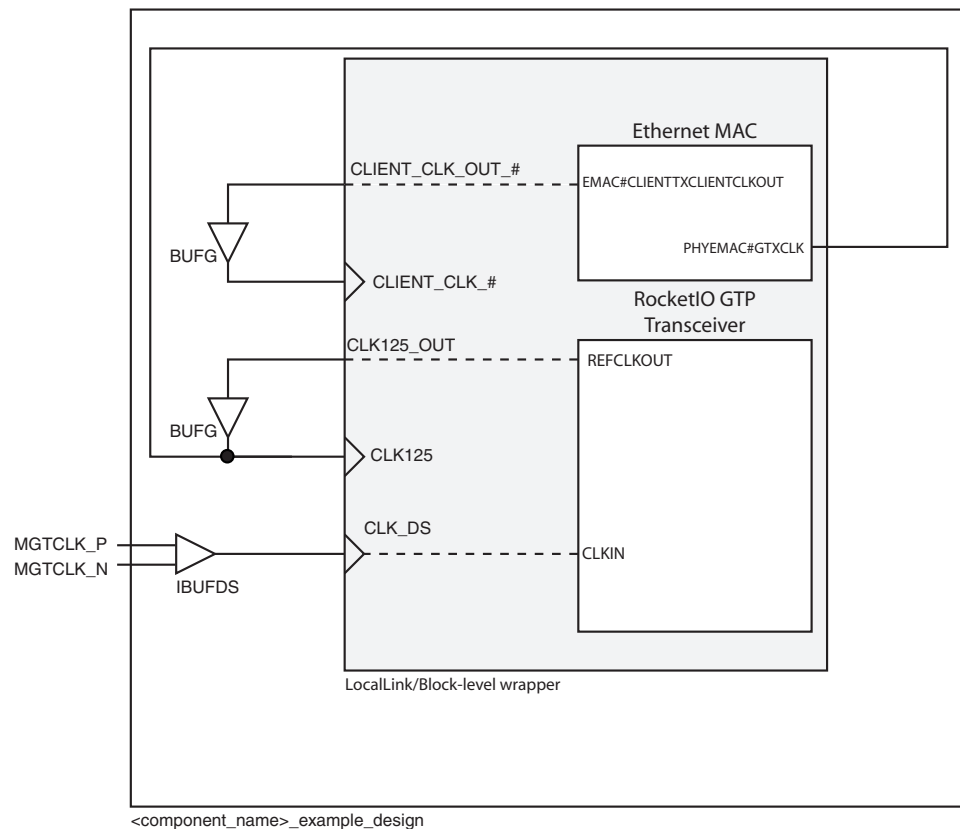


Figure B-5: SGMII Clocking at 10/100/1000 Mb/s: Virtex-5 LXT and SXT Devices

SGMII at Multiple Speeds: Virtex-5 FXT and TXT Devices

For SGMII implementation in Virtex-5 FXT and TXT devices the user supplies the high-quality differential clock (CLK_DS) and 125 MHz reference clock (CLK125). These are used to clock the RocketIO GTX transceiver and the physical side of the Ethernet MAC and can be shared between multiple instantiations of the wrappers.

The RocketIO GTX transceiver also requires a 62.5 MHz clock input. This drives the transceiver 2-byte internal datapath. This clock should be generated from REFCLKOUT via a DCM. In addition a 1.25/12.5/125 MHz client clock (CLIENT_CLK_#) must be supplied. This is used to drive the client logic at all 3 speeds. The EMAC#CLIENTTXCLIENTCLKOUT output from the EMAC is made available to the user on the CLIENT_CLK_OUT_# port and this is used to drive CLIENT_CLK_#. This clock cannot be shared between multiple Ethernet MACs unless they all operate at the same speed. Figure B-6 shows the supplied clocking scheme.

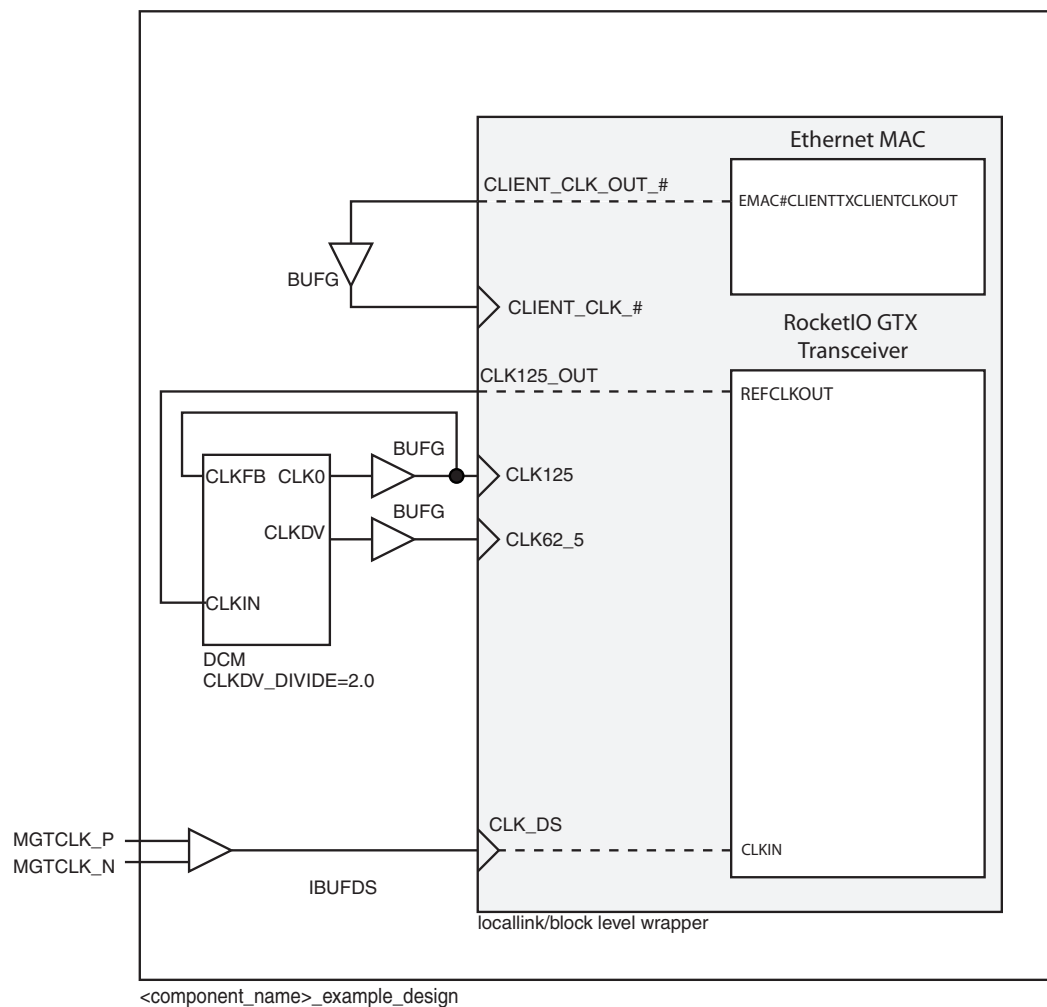


Figure B-6: SGMII Clocking at 10/100/1000 Mb/s: Virtex-5 FXT and TXT Devices

GMII/MII/RGMII at Multiple Speeds

There are a variety of clocking schemes available when using a parallel PHY interface at multiple speeds. Figure B-7, Figure B-8 and Figure B-9 show the default method, where the user supplies 1.25/12.5/125 MHz clocks for the transmit and receive client interfaces and 2.5/25/125 MHz clocks for the physical interface. The signals are generated from the clock outputs of the Ethernet MAC.

Because this method uses a large amount of clocking resources, Xilinx recommends that you use the clock enable or Byte PHY methods shown in Figure B-10, Figure B-11, and Figure B-12.

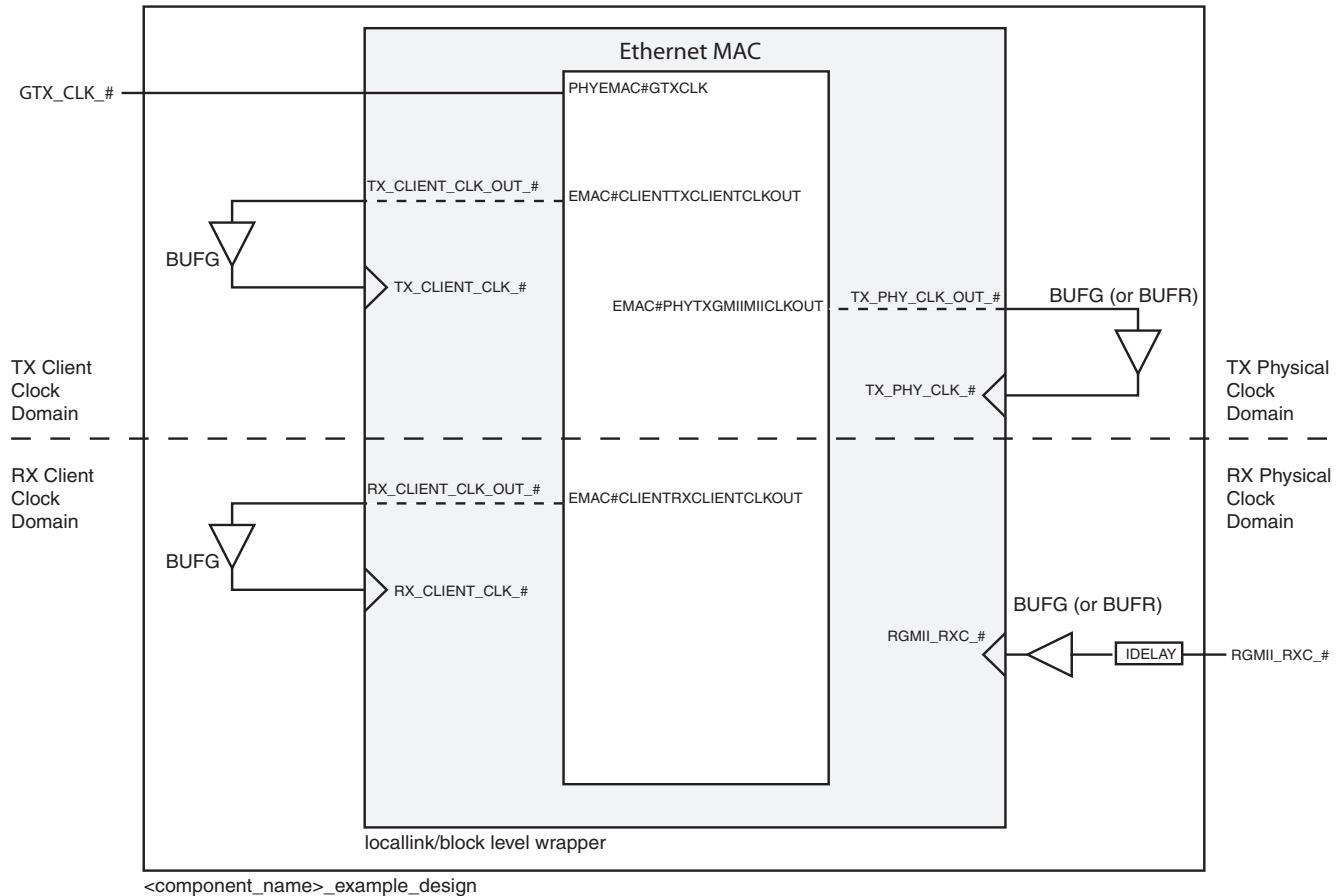


Figure B-7: RGMII Clocking at 10/100/1000 Mb/s

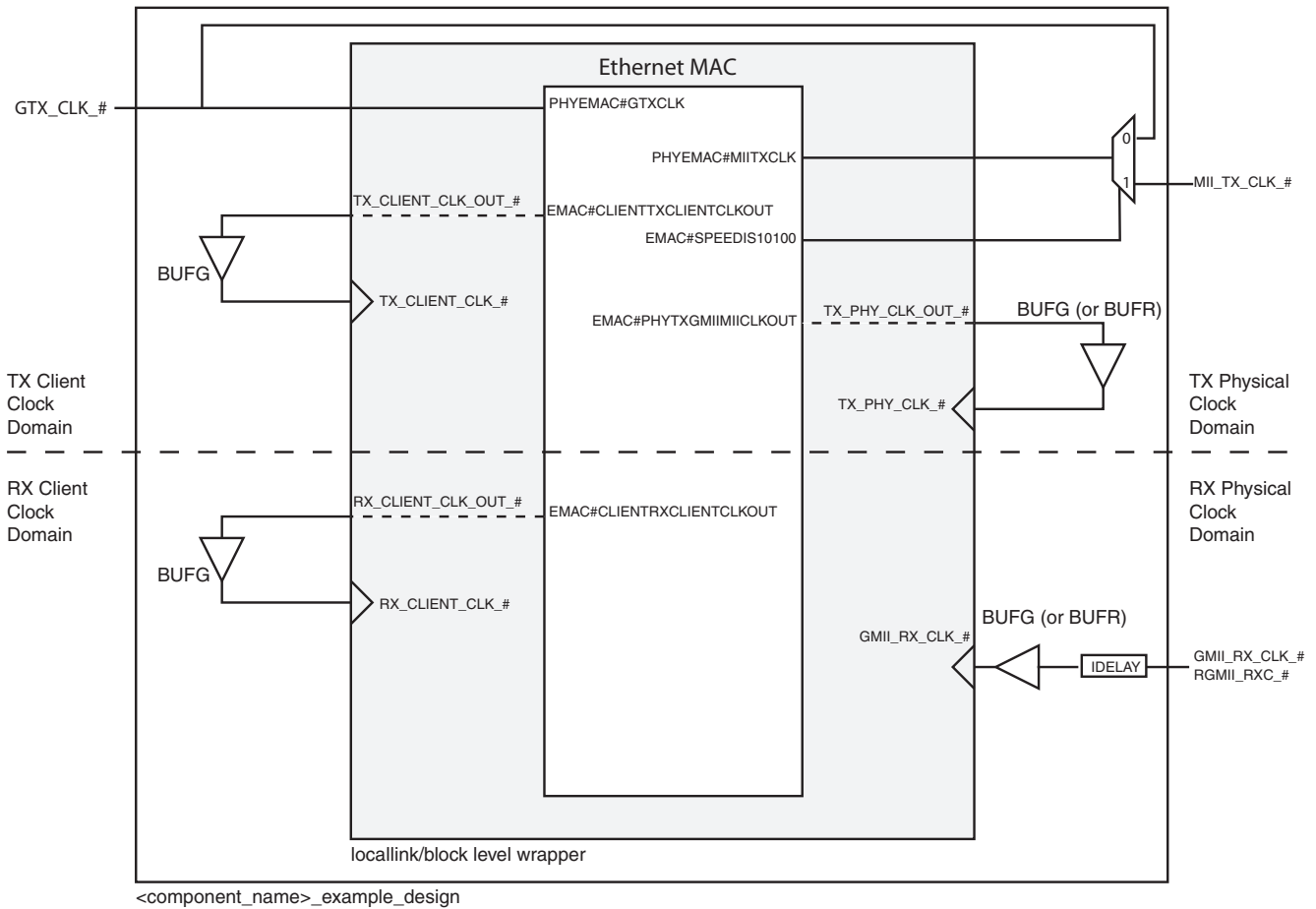


Figure B-8: GMII clocking at 10/100/1000 Mb/s

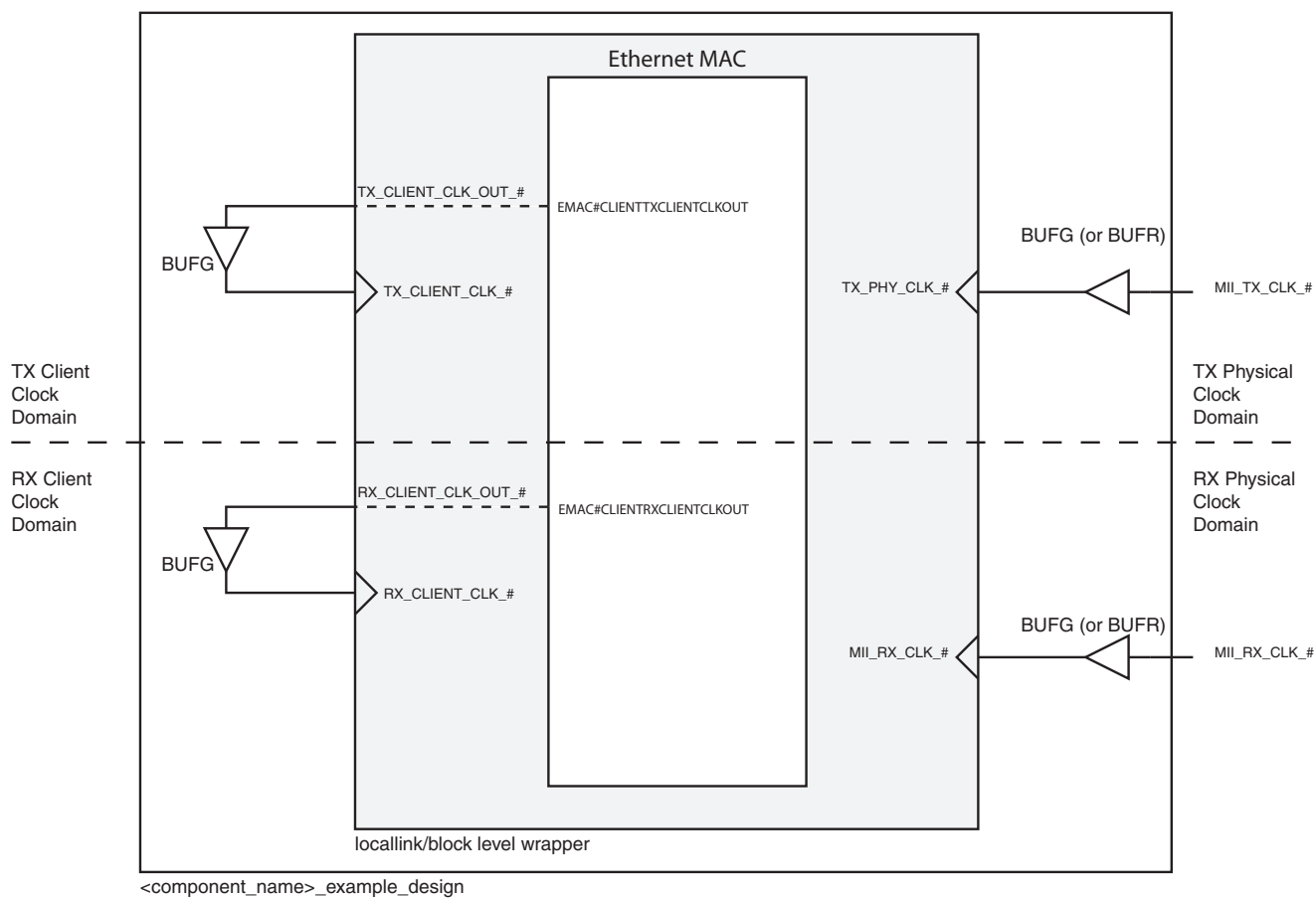


Figure B-9: MII Clocking at 10/100 Mb/s

GMII/MII at Multiple Speeds with Clock Enable

Clock enable mode is used to reduce the amount of clocking resources that are used when running at multiple speeds with a parallel interface (Figure B-10). In this mode the transmitter clock is supplied from a high quality 125 MHz reference clock (GTX_CLK_#) at 1000 Mb/s and from the 2.5/25 MHz TX input clock from the PHY (MII_TX_CLK_#) at 10/100 Mb/s. The receiver is clocked by the 2.5/25/125 MHz receiver clock from the PHY (GMII/MII_RX_CLK_#). In GMII mode an IDELAY element is also used to align the clock to the data inputs as they enter the FPGA.

The clock enable outputs (tx_enable_#_i and rx_enable_#_i) are used by the 8-bit client logic to maintain the correct data rate through the system. At 1000 Mb/s the clock enables are held high. At slower speeds the clock enables are high on each alternate clock cycle. This gives a data throughput of 100 Mb/s on the 25 MHz clock and 10 Mb/s on the 2.5 MHz clock.

If the MII interface is used (10/100 Mb/s only) the BUFGMUX is replaced by a BUFG with MII_TX_CLK_# as its input. It should be noted that, together with the receiver clock, the transmitter clock must still be constrained to run at 125 MHz even if the design does not operate at 1000 Mb/s.

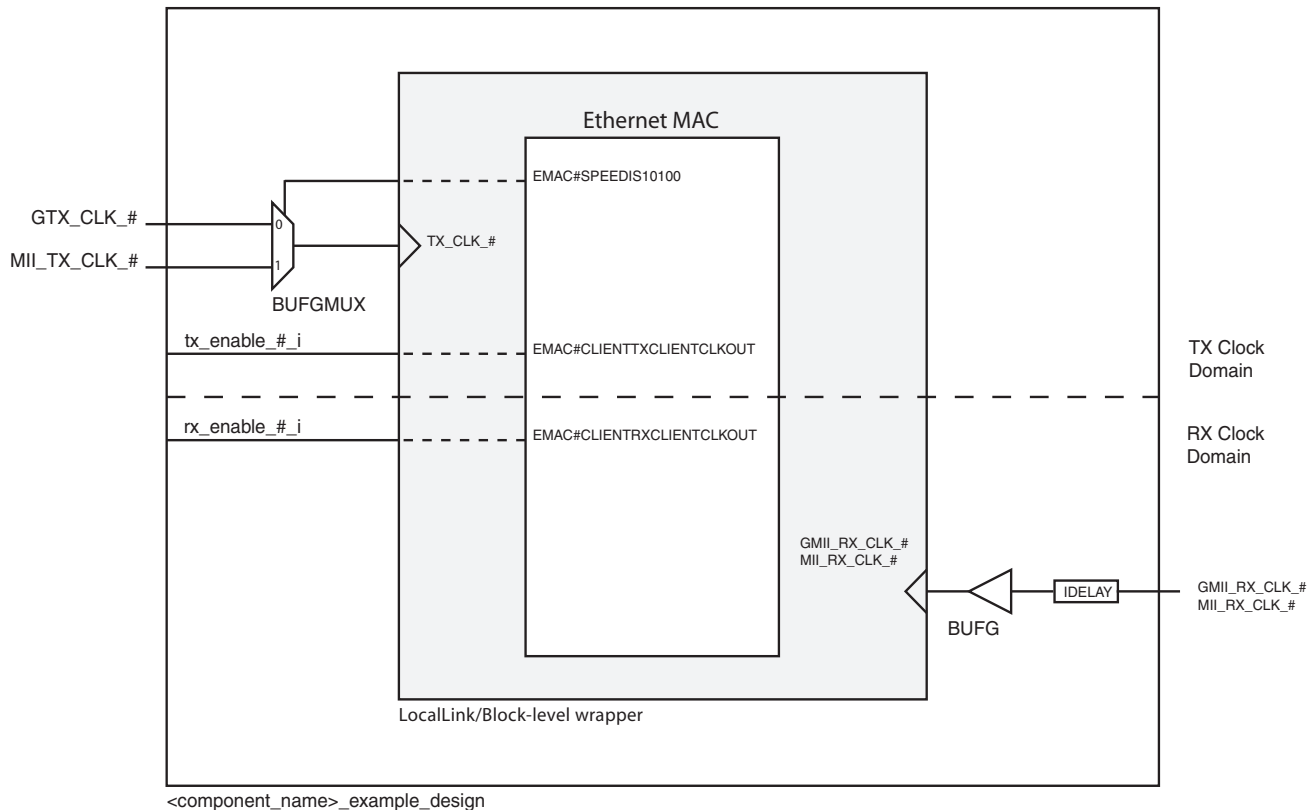


Figure B-10: GMII/MII Clocking at 10/100/1000 Mb/s with Clock Enables

RGMII at Multiple Speeds with Clock Enable

Figure B-11 shows the clocking scheme used when running at multiple speeds with the RGMII interface. In this case, the transmit clock runs at 2.5, 25, or 125 MHz depending on the speed of operation. The TX_CLK_OUT_# output carries the EMAC#CLIENTTXCLIENTCLKOUT output from the Ethernet MAC and is used to drive the transmitter input clock (TX_CLK_#).

The receiver is clocked by the input clock from the PHY through an IDELAY and a BUFG. The IDELAY element is used to align the clock to the data inputs as they enter the FPGA. Clock enables are used to control the data throughput at the client interface.

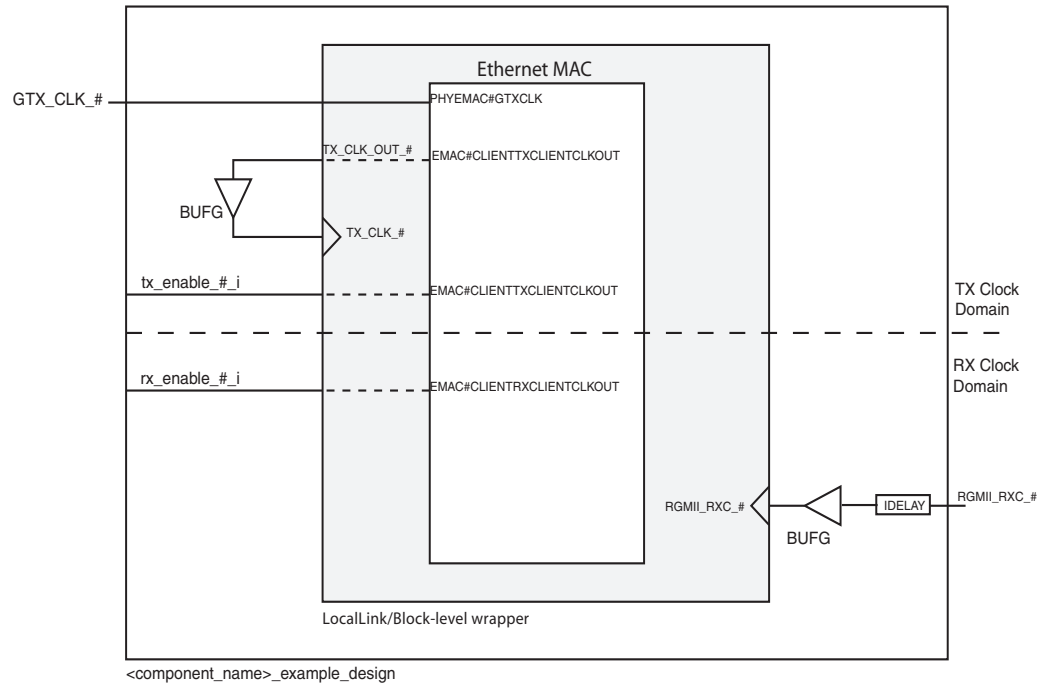


Figure B-11: RGMII Clocking at 10/100/1000 Mb/s with Clock Enable

GMII/MII at Multiple Speeds with Byte PHY

An alternative to clock enable is to use Byte PHY mode, illustrated in Figure B-12. This has advantages for MII operation at 10/100 Mb/s as the transmit and receive clocks do not need to be constrained to run at 125 MHz. In addition the client circuitry does not need to be clock enabled.

When running at all three speeds, the transmitter circuitry is driven by the 125 MHz reference clock at 1000 Mb/s and by the MII_TX_CLK_# input from the PHY at slower speeds. However, at the slower speeds the frequency of MII_TX_CLK_# is divided by 2. A similar setup is used for the receiver circuitry. If the design does not run at 1000 Mb/s, the BUFG MUXs are replaced by BUFGs that take the divided MII_TX_CLK_# and MII_RX_CLK signals as inputs, as illustrated in Figure B-13.

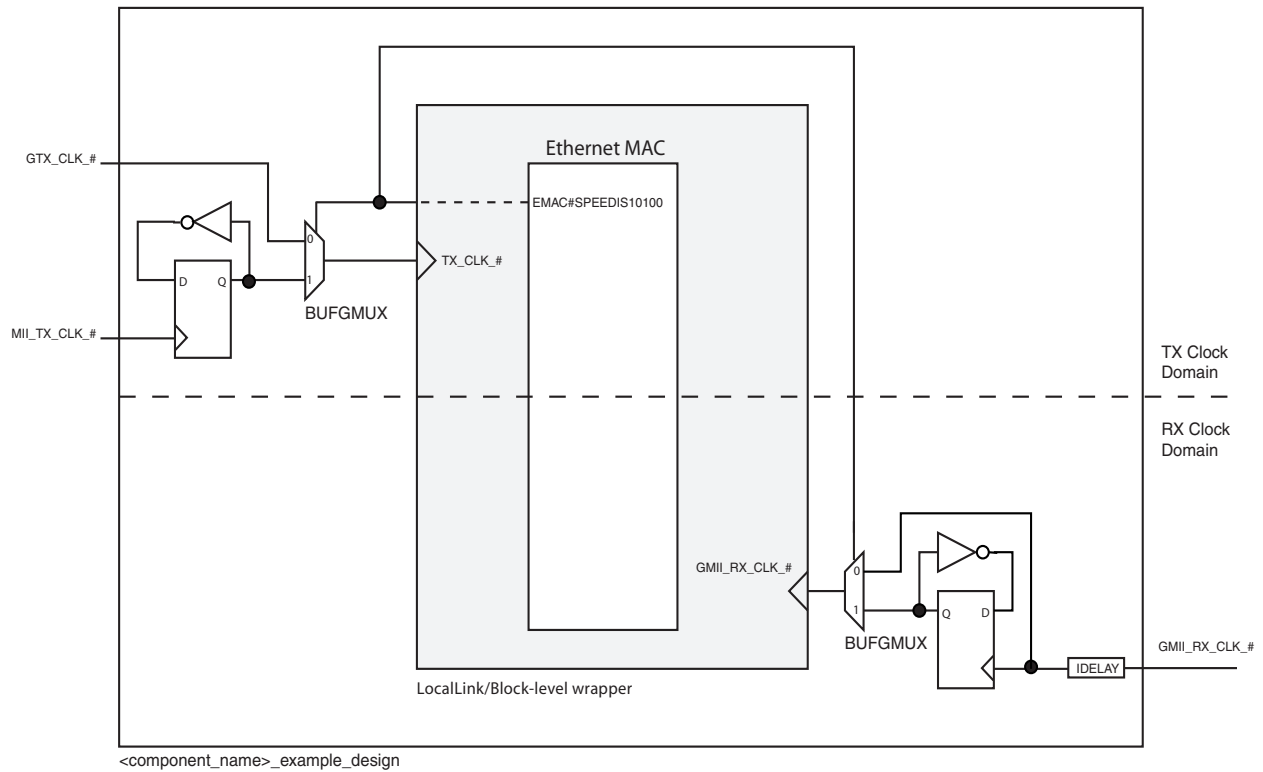


Figure B-12: GMII Clocking at 10/100/1000 Mb/s with Byte PHY

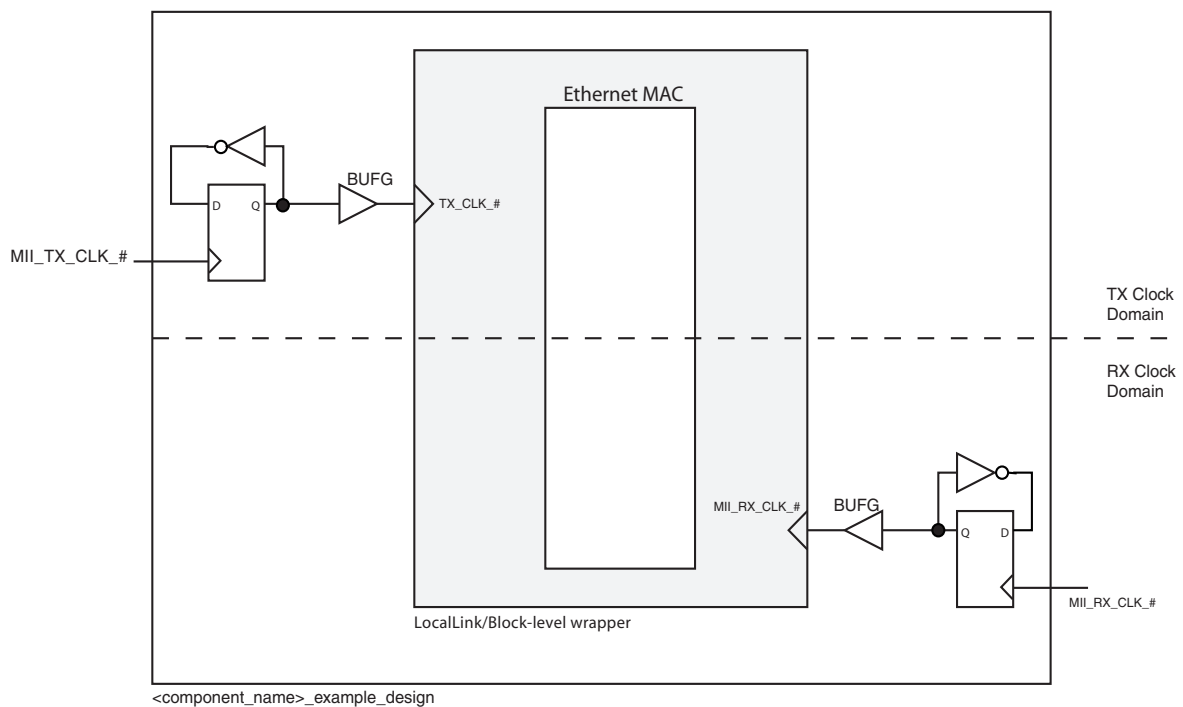


Figure B-13: MII Clocking at 10/100 Mb/s with Byte PHY

Constraining the Example Design

An example UCF, separated into three sections, is provided with the HDL example design, and provides examples of constraint requirements for the block, LocalLink, and example_design levels. In all the examples, (#) represents the Ethernet MAC number (EMAC0 or EMAC1).

Block Level Constraints

The block level UCF (<component_name>_block.ucf) contains the constraints for the clocks in the design. For more information on the clocking schemes used for the various physical interfaces see [Appendix B, Ethernet MAC Clocking](#).

PCS/PMA/SGMII Clock Constraints

The following constraints must be used with a RocketIO™ transceiver. This constrains the Ethernet MAC input clock to run at 125 MHz.

```
# 125 MHz clock input from BUFG
NET "CLK125" TNM_NET = "clk_gtp";
TIMEGRP "<component_name>_gtp_clk" = "clk_gtp";
TIMESPEC "TS_<component_name>_gtp_clk" = PERIOD
"<c_component_name>_gtp_clk" 8 ns HIGH 50 %;
```

If SGMII is selected and the device is to operate at 10/100/1000 Mb/s, the CLIENT_CLK_# input must be constrained to run at 125 MHz.

```
# EMAC# Tri-speed clock input from BUFG
NET "CLIENT_CLK_#" TNM_NET = "clk_client#";
TIMEGRP "<component_name>_gtp_clk_client#" = "clk_client#";
TIMESPEC "TS_<component_name>_gtp_clk_client#" = PERIOD
"<component_name>_gtp_clk_client#" 8 ns HIGH 50 %;
```

When the Ethernet MAC is used in overclocking mode (Client Interface Width = 16-bit) then the CLK250 input should be constrained to run at 250 MHz.

```
# 250 MHz clock input from DCM
NET "CLK250" TNM_NET = "clk_2x";
TIMEGRP "<component_name>_gtp_clk_2x" = "clk_2x";
TIMESPEC "TS_<component_name>_gtp_clk_2x" = PERIOD
"<component_name>_gtp_clk_2x" 4 ns HIGH 50 %;
```

If a Virtex®-5 LXT or SXT device is targeted and the Ethernet MAC is configured in No Clock SGMII mode, the following constraints should be applied to constrain the recovered clock and remove meta-stability in the fabric buffer.

```
#-----
# EMAC# Fabric Rx Elastic Buffer Timing Constraints: -
#-----
NET "GTP_DUAL_1000X_inst?RXRECCLK_#_BUFR" TNM_NET = "clk_rec_clk#";
TIMEGRP "<component_name>_client_rec_clk#" = "clk_rec_clk#";
TIMESPEC "TS_<component_name>_rec_clk#" = PERIOD
"<component_name>_client_rec_clk#" 8ns HIGH 50 %;

# Control Gray Code delay and skew
NET "GTP_DUAL_1000X_inst?rx_elastic_buffer_inst_#?wr_addr_gray<?>"
MAXDELAY = 6 ns;

# Reduce clock period to allow 3 ns for metastability settling time
INST "GTP_DUAL_1000X_inst?rx_elastic_buffer_inst_#?rd_wr_addr_gray*"
TNM = "rx_graycode_#";
INST "GTP_DUAL_1000X_inst?rx_elastic_buffer_inst_#?rd_occupancy*"
TNM = "rx_binary_#";
TIMESPEC "ts_rx_buf_meta_protect_#" = FROM "rx_graycode_#" TO
"rx_binary_#" 5 ns;
```

If a Virtex-5 FXT or TXT device is targeted and the Ethernet MAC is configured in *No Clock SGMII* mode, the following constraints should be applied to constrain the recovered clock and remove meta-stability in the fabric buffer.

```
#-----
# EMAC# Fabric Rx Elastic Buffer Timing Constraints: -
#-----
NET "GTX_DUAL_1000X_inst?RXRECCLK_#_BUFR" TNM_NET = "clk_rec_clk#";
TIMEGRP "<component_name>_client_rec_clk#" = "clk_rec_clk#";
TIMESPEC "TS_<component_name>_rec_clk#" = PERIOD
"<component_name>_client_rec_clk#" 8 ns HIGH 50 %;
# Control Gray Code delay and skew
NET "GTX_DUAL_1000X_inst?rx_elastic_buffer_inst_#?wr_addr_gray<?>"
MAXDELAY = 6 ns;
# Reduce clock period to allow 3 ns for metastability settling time
INST "GTX_DUAL_1000X_inst?rx_elastic_buffer_inst_#?rd_wr_addr_gray*"
TNM = "rx_graycode_#";
INST "GTX_DUAL_1000X_inst?rx_elastic_buffer_inst_#?rd_occupancy*"
TNM = "rx_binary_#";
TIMESPEC "ts_rx_buf_meta_protect_#" = FROM "rx_graycode_#" TO
"rx_binary_#" 5 ns;
```

GMII/RGMII 1000 Mb/s Clock Constraints

If GMII or RGMII are selected and the speed is set to 1000 Mb/s then the following constraints should be applied to the design.

```
# EMAC# TX Clock input from BUFG
NET "TX_CLK_#" TNM_NET = "clk_tx#";
TIMEGRP "<component_name>_gtx_clk#" = "clk_tx#";
TIMESPEC "TS_<component_name>_gtx_clk#" = PERIOD
"<component_name>_gtx_clk#" 8 ns HIGH 50 %;

# EMAC# RX PHY Clock
NET "GMII_RX_CLK_#" TNM_NET = "phy_clk_rx#";
TIMEGRP "<component_name>_gclk_phy_rx#" = "phy_clk_rx#";
TIMESPEC "TS_<component_name>_gclk_phy_rx#" = PERIOD
"<component_name>_gclk_phy_rx#" 8 ns HIGH 50 %;
```

If the Ethernet MAC is configured in RGMII mode RGMII_RXC_# replaces GMII_RX_CLK_#.

GMII/MII/RGMII 10/100/1000 Mb/s Clock Constraints

If the design is used in multi-speed mode the constraints are dependant on whether clock enable or byte PHY modes are being used.

If the clock enable or Byte PHY options are selected then the following constraints should be applied to the design.

```
# EMAC0 TX Clock input from BUFG
NET "TX_CLK_#" TNM_NET = "clk_tx#";
TIMEGRP "<component_name>_tx_clk#" = "clk_tx#";
TIMESPEC "TS_<component_name>_tx_clk#" = PERIOD
"<component_name>_tx_clk#" 8 ns HIGH 50 %;

# EMAC0 RX PHY Clock
NET "GMII_RX_CLK_#" TNM_NET = "phy_clk_rx#";
TIMEGRP "<component_name>_clk_phy_rx#" = "phy_clk_rx#";
TIMESPEC "TS_<component_name>_clk_phy_rx#" = PERIOD
"<component_name>_clk_phy_rx#" 8 ns HIGH 50 %;
```

If MII is selected, these constraints can be relaxed in Byte PHY mode.

If no advanced clocking options are selected then the following constraints are included in the <component_name>_block.ucf file.

```
# EMAC# TX Client Clock input from BUFG
NET "TX_CLIENT_CLK_#" TNM_NET = "clk_client_tx#";
TIMEGRP "<component_name>_client_clk_tx#" = "clk_client_tx#";
TIMESPEC "TS_<component_name>_client_clk_tx#" = PERIOD
"<component_name>_client_clk_tx#" 8 ns HIGH 50 %;

# EMAC# RX Client Clock input from BUFG
NET "RX_CLIENT_CLK_#" TNM_NET = "clk_client_rx#";
TIMEGRP "<component_name>_client_clk_rx#" = "clk_client_rx#";
TIMESPEC "TS_<component_name>_client_clk_rx#" = PERIOD
"<component_name>_client_clk_rx#" 8 ns HIGH 50 %;
```

```
# EMAC# TX PHY Clock input from BUFG
NET "TX_PHY_CLK_#" TNM_NET = "clk_phy_tx#";
TIMEGRP "<component_name>_phy_clk_tx#" = "clk_phy_tx#";
TIMESPEC "TS_<component_name>_phy_clk_tx#" = PERIOD
"<component_name>_phy_clk_tx#" 8 ns HIGH 50 %;

# EMAC# RX PHY Clock
NET "GMII_RX_CLK_#" TNM_NET = "phy_clk_rx#";
TIMEGRP "<component_name>_clk_phy_rx#" = "phy_clk_rx#";
TIMESPEC "TS_<component_name>_clk_phy_rx#" = PERIOD
"<component_name>_clk_phy_rx#" 8 ns HIGH 50 %;
```

If the Ethernet MAC is configured in RGMII mode RGMII_RXC_# replaces GMII_RX_CLK_#. In MII mode MII_RX_CLK_# replaces GMII_RX_CLK_#. If MII is selected, these constraints can be relaxed.

GMII IDELAY_VALUE Constraints

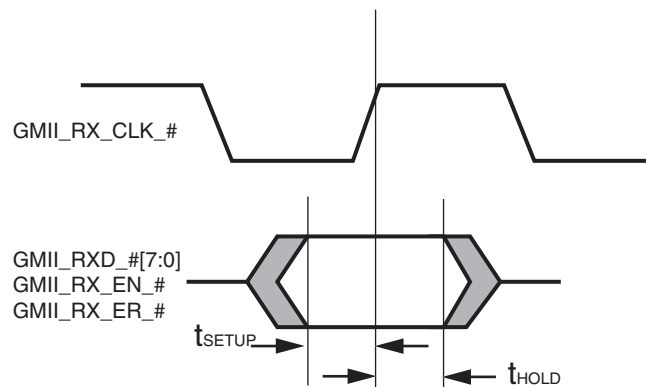


Figure C-1: Input GMII Timing

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

Table C-1: Input GMII Timing

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

There is, in total, a 2 ns data valid window of guaranteed data that is presented across the GMII input bus. This must be correctly sampled by the FPGA.

To do this IDELAY elements are placed on the GMII_RX_CLK_#, GMII_RXD_#[7:0], GMII_RX_EN_# and GMII_RX_ER_# inputs. The IDELAY_VALUE parameters of these elements is set in the UCF so that the data is sampled correctly. The following constraints are for the example placement given.


```
# Data and control inputs
INST "*gmii#?ideldv" IDELAY_VALUE = 40;
INST "*gmii#?ideld0" IDELAY_VALUE = 40;
INST "*gmii#?ideld1" IDELAY_VALUE = 40;
INST "*gmii#?ideld2" IDELAY_VALUE = 40;
INST "*gmii#?ideld3" IDELAY_VALUE = 40;
INST "*gmii#?ideld4" IDELAY_VALUE = 40;
INST "*gmii#?ideld5" IDELAY_VALUE = 40;
INST "*gmii#?ideld6" IDELAY_VALUE = 40;
INST "*gmii#?ideld7" IDELAY_VALUE = 40;
INST "*gmii#?ideler" IDELAY_VALUE = 40;

# Clock input
INST "*gmii_rxc#_delay" IDELAY_VALUE = 0;
```

Setup and hold information is included in the timing report when the trce command is invoked with the -u option. This gives information that can be used to set the IDELAY_VALUE parameters correctly.

RGMII IDELAY_VALUE Constraints

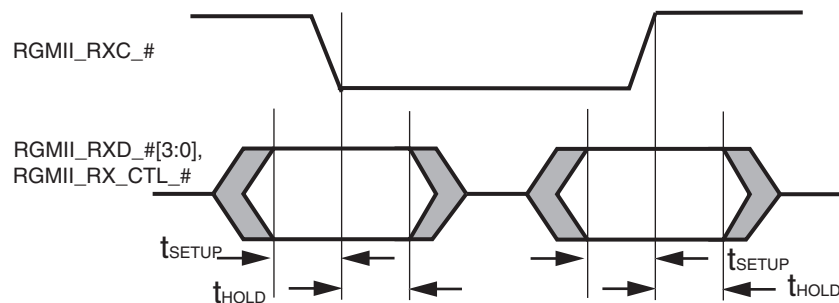


Figure C-2: RGMII Input Timing

Figure C-2 and Table C-2 illustrate the input setup and hold time window for the input RGMII signals. These are the worst-case data valid window presented to the FPGA device pins.

Table C-2: Input RGMII Timing

Symbol	Min	Max	Units
t_{SETUP}	1.00	-	ns
t_{HOLD}	1.00	-	ns

There is, in total, a 2 ns data valid window of guaranteed data that is presented across the RGMII input bus. This must be correctly sampled by the FPGA.

To do this IDELAY elements are placed on the RGMII_RXC_#, RGMII_RXD_#[3:0] and RGMII_RX_CTL_# inputs. The IDELAY_VALUE parameters of these elements is set in the UCF so that the data is sampled correctly. The following constraints are for the example placement given.

```
# Data and control inputs
INST "*rgmii#?rgmii_rx_ctl_delay" IDELAY_VALUE = 27;
INST "*rgmii#?rgmii_rx_d0_delay" IDELAY_VALUE = 27;
INST "*rgmii#?rgmii_rx_d1_delay" IDELAY_VALUE = 27;
INST "*rgmii#?rgmii_rx_d2_delay" IDELAY_VALUE = 27;
INST "*rgmii#?rgmii_rx_d3_delay" IDELAY_VALUE = 27;

# Clock input
INST "*rgmii_rxc#_delay" IDELAY_VALUE = 0;
```

Setup and hold information is included in the timing report when the `trce` command is invoked with the `-u` option. This gives information that can be used to set the `IDELAY_VALUE` parameters correctly.

LocalLink Level Constraints

The LocalLink level UCF (<component_name>_locallink.ucf) includes constraints to handle clock domain crossing in the client FIFOs.

```
# EMAC1 LocalLink client FIFO constraints.

INST "*client_side_FIFO_emac#?tx_fifo_i?rd_tran_frame_tog"
TNM = "tx_fifo_rd_to_wr_1";
INST "*client_side_FIFO_emac#?tx_fifo_i?rd_retran_frame_tog"
TNM = "tx_fifo_rd_to_wr_1";
INST "*client_side_FIFO_emac#?tx_fifo_i?rd_col_window_pipe_1"
TNM = "tx_fifo_rd_to_wr_1";
INST "*client_side_FIFO_emac#?tx_fifo_i?rd_addr_txfer*"
TNM = "tx_fifo_rd_to_wr_1";
INST "*client_side_FIFO_emac#?tx_fifo_i?rd_txfer_tog"
TNM = "tx_fifo_rd_to_wr_1";
INST "*client_side_FIFO_emac#?tx_fifo_i?wr_frame_in_fifo"
TNM = "tx_fifo_wr_to_rd_1";

TIMESPEC "TS_tx_fifo_rd_to_wr_#" = FROM "tx_fifo_rd_to_wr_#" TO
"tx_local_link_clock_#" 8 ns DATAPATHONLY;
TIMESPEC "TS_tx_fifo_wr_to_rd_#" = FROM "tx_fifo_wr_to_rd_#" TO
"tx_client_clk_#" 8 ns DATAPATHONLY;

# Reduce clock period to allow 3 ns for metastability settling time
INST "*client_side_FIFO_emac#?tx_fifo_i?wr_tran_frame_tog"
TNM = "tx_metastable_#";
INST "*client_side_FIFO_emac#?tx_fifo_i?wr_rd_addr*"
TNM = "tx_metastable_#";
INST "*client_side_FIFO_emac#?tx_fifo_i?wr_txfer_tog"
TNM = "tx_metastable_#";
INST "*client_side_FIFO_emac#?tx_fifo_i?frame_in_fifo"
TNM = "tx_metastable_#";
INST "*client_side_FIFO_emac#?tx_fifo_i?wr_retran_frame_tog*"
TNM = "tx_metastable_#";
INST "*client_side_FIFO_emac#?tx_fifo_i?wr_col_window_pipe_0"
TNM = "tx_metastable_#";

TIMESPEC "ts_tx_meta_protect_#" = FROM "tx_metastable_#" 5 ns
DATAPATHONLY;

INST "*client_side_FIFO_emac#?tx_fifo_i?rd_addr_txfer*"
```

```

TNM = "tx_addr_rd_#";
INST "*client_side_FIFO_emac#?tx_fifo_i?wr_rd_addr*"
TNM = "tx_addr_wr_#";
TIMESPEC "TS_tx_fifo_addr_#" = FROM "tx_addr_rd_#" TO "tx_addr_wr_#"
10ns;

```

Example Design Level Constraints

The top-level example design UCF (<component_name>_example_design.ucf) sets the part to a 5v1x50tff1136-1 device. This should be changed to the desired Virtex-5 FPGA.

The file also contains example placement and IO standard specification. In addition constraints are provided for the management interface and IODELAY controller clocks.

GMII/MII Interface

The GMII and MII interfaces are specified to operate at the LVTTTL standard.

```

# GMII Logic Standard Constraints
INST "gmii_txd_#<?>"      IOSTANDARD = LVTTTL;
INST "gmii_tx_en_#"        IOSTANDARD = LVTTTL;
INST "gmii_tx_er_#"        IOSTANDARD = LVTTTL;
INST "gmii_rxd_#<?>"      IOSTANDARD = LVTTTL;
INST "gmii_rx_dv_#"        IOSTANDARD = LVTTTL;
INST "gmii_rx_er_#"        IOSTANDARD = LVTTTL;
INST "gmii_tx_clk_#"        IOSTANDARD = LVTTTL;
INST "gmii_rx_clk_#"        IOSTANDARD = LVTTTL;
INST "mii_tx_clk_#"         IOSTANDARD = LVTTTL;

```

RGMII v2.0 Interface

The RGMII version 2.0 interface is constrained to operate at the HSTL_I standard.

```

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;

```

Example Placement

Example pin placement is specified for the GMII and RGMII interfaces. For all interfaces the clock inputs are constrained to banks that contain global clock capable inputs.

```

INST "rgmii_rxd_#<0>"      LOC = "BANK4";
INST "rgmii_rxd_#<1>"      LOC = "BANK4";
INST "rgmii_rxd_#<2>"      LOC = "BANK4";
INST "rgmii_rxd_#<3>"      LOC = "BANK4";
INST "rgmii_rx_ctl_#"       LOC = "BANK4";

INST "rgmii_rxc_#"          LOC = "AF18";

INST "GTX_CLK"              LOC = "BANK4";

```

```
INST "REFCLK" LOC = "BANK4";
```

When a serial IO interface is selected, the transceiver is placed in a specific GTP_DUAL or GTX_DUAL site. This should be changed to the transceiver that is being used:

For Virtex-5 LXT Devices

```
INST "*GTP_DUAL_1000X_inst?GTP_1000X" LOC = "GTP_DUAL_X0Y2";
INST "MGTCLK_N" LOC = "AF3";
INST "MGTCLK_P" LOC = "AF4";
```

For Virtex-5 SXT Devices

```
INST "*GTP_DUAL_1000X_inst?GTP_1000X" LOC = "GTP_DUAL_X0Y2";
INST "MGTCLK_N" LOC = "Y3"; # or AF3 for QVirtex-5 SXT parts
INST "MGTCLK_P" LOC = "Y4"; # or AF4 for QVirtex-5 SXT parts
```

For Virtex-5 FXT Devices

```
INST "*GTX_DUAL_1000X_inst?GTX_1000X" LOC = "GTX_DUAL_X0Y3";
INST "MGTCLK_N" LOC = "Y3";
INST "MGTCLK_P" LOC = "Y4";
```

For Virtex-5 TXT Devices

```
INST "*GTX_DUAL_1000X_inst?GTX_1000X" LOC = "GTX_DUAL_X1Y5";
INST "MGTCLK_N" LOC = "V3";
INST "MGTCLK_P" LOC = "V4";
```

GMII/RGMII IODELAY Controller Clock Constraint

In GMII and RGMII modes a 200MHz clock must be provided to control the IODELAY components. This is constrained in the `<component_name>_example_design.ucf` file.

```
NET "*refclk_bufg_i" TNM_NET = "clk_ref_clk";
TIMEGRP "ref_clk" = "clk_ref_clk";
TIMESPEC "TS_ref_clk" = PERIOD "ref_clk" 5 ns HIGH 50 %;
```

Host Interface Clock Constraint

If the optional host interface is selected, the following clock constraint is applied in `<component_name>_example_design.ucf`. The host interface can share any of the other Ethernet MAC clocks.

```
NET "*host_clk_i" TNM_NET = "host_clock";
TIMEGRP "clk_host" = "host_clock";
TIMESPEC "TS_clk_host" = PERIOD "clk_host" 10 ns HIGH 50 %;
```

DCR Interface Clock Constraint

If the optional DCR interface is selected, the following clock constraint is applied in `<component_name>_example_design.ucf`. The DCR interface can share any of the other Ethernet MAC clocks.

```
NET "*dcr_clk_i" TNM_NET = "host_clock";
TIMEGRP "clk_host" = "host_clock";
TIMESPEC "TS_clk_host" = PERIOD "clk_host" 10 ns HIGH 50 %;
```

SGMII Receiver Elastic Buffer

SGMII Capabilities

The Ethernet MAC wrapper GUI provides two SGMII Capabilities options:

- **10/100/1000 Mb/s (no clock constraints required).** Default setting; provides the implementation using the Receiver Elastic Buffer in FPGA fabric. This alternative Receiver Elastic Buffer utilizes a single block RAM to create a buffer twice as large as the one present in the transceiver, subsequently consuming extra logic resources. However, this default mode provides reliable SGMII operation under all conditions.
- **10/100/1000 Mb/s OR 100/1000 Mb/s (clock constraints required).** Uses the receiver elastic buffer present in the RocketIO™ transceivers. This is half the size and can potentially under- or overflow during SGMII frame reception at 10 Mb/s operation. However, there are logical implementations where this can be proven reliable: if so it is favored because of its lower logic utilization.

FPGA Fabric Rx Elastic Buffer Requirement

Figure D-1 illustrates a simplified diagram of a common situation where the core, in SGMII mode, is interfaced to an external PHY device. Separate oscillator sources are used for the FPGA and the external PHY. The Ethernet specification uses clock sources with a tolerance of 100 parts per million (ppm). In Figure D-1, the clock source for the PHY is slightly faster than the clock source to the FPGA. For this reason, during frame reception, the receiver elastic buffer (shown here as implemented in the RocketIO transceiver) starts to fill.

Following frame reception, in the interframe gap period, idles are removed from the received data stream to return the Rx Elastic Buffer to half full occupancy: this is performed by the clock correction circuitry (see the [Virtex-5 FPGA RocketIO GTP Transceiver User Guide](#) and [Virtex-5 FPGA RocketIO GTX Transceiver User Guide](#)).

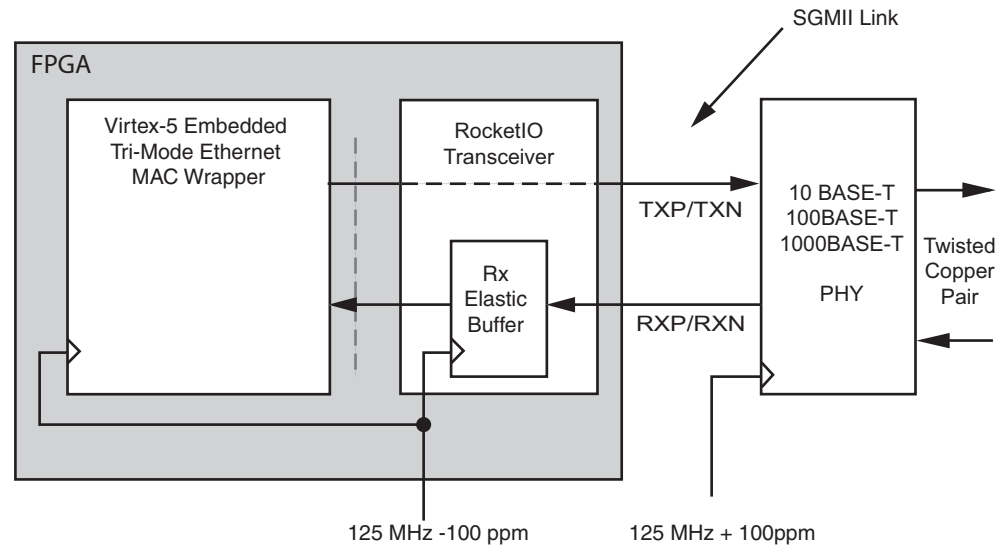


Figure D-1: SGMII Implementation: Separate Clock Sources

Analysis

Assuming separate clock sources, each with a tolerance of 100 ppm, the maximum frequency difference between the two devices can be 200 ppm. It can be shown that this translates into a full clock period difference every 5000 clock periods.

Relating this to an Ethernet frame, a single byte of difference every 5000 bytes of received frame data occurs, causing the Rx Elastic Buffer to either fill or empty by an occupancy of one.

The maximum sized Ethernet frame (non-jumbo) is of size 1522 bytes for a VLAN frame:

- At 1 Gb/s operation, this translates into 1522 clock cycles
- At 100 Mb/s operation, this translates into 15220 clock cycles (because each byte is repeated 10 times)
- At 10 Mb/s operation, this translates into 152200 clock cycles (because each byte is repeated 100 times).

Considering the 10 Mb/s case, we would need $152200/5000 = 31$ FIFO entries in the Elastic Buffer above and below the half way point to guarantee that the buffer will not under or overflow during frame reception. This assumes that frame reception begins when the buffer is exactly half full.

The size of the Rx Elastic Buffer in the RocketIO transceivers is of size 64 entries. However, we cannot assume that the buffer is exactly half-full at the start of frame reception. Additionally, the underflow and overflow thresholds are not exact. See the RocketIO User Guides, available from www.xilinx.com:

- *Virtex-5 FPGA RocketIO GTP Transceiver User Guide (UG196)*
- *Virtex-5 FPGA RocketIO GTX Transceiver User Guide (UG198)*

To guarantee reliable SGMII operation at 10 Mb/s (non-jumbo frames), the RocketIO Elastic Buffer must be bypassed and a larger buffer implemented in the FPGA fabric. The fabric buffer, provided by the example design, is twice the size and so nominally provides 64 entries above and below the half full threshold. This has been proven to cope with standard (non-jumbo) Ethernet frames at all three SGMII speeds.

The RocketIO Rx Elastic Buffer

The Elastic Buffer in the RocketIO transceiver can be used reliably under the following conditions:

- When 10 Mb/s operation is not required (for example, when connecting the core to the 1-Gigabit Ethernet MAC to provide only 1 Gb/s operation). Both 1 Gb/s and 100 Mb/s operation are guaranteed.
- When the clocks are closely related (see the next section).

If any uncertainty exists, select the FPGA fabric Rx Elastic Buffer Implementation.

Closely Related Clock Sources

Scenario 1

Figure D-2 illustrates a simplified diagram of a common situation where the core, in SGMII mode, is interfaced to an external PHY device. A common oscillator source is used for both the FPGA and the external PHY.

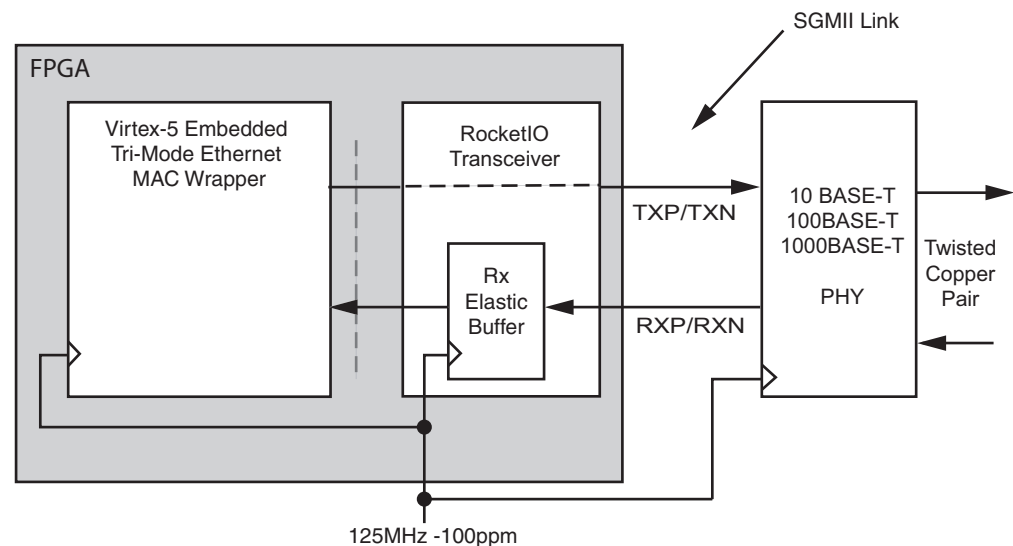


Figure D-2: SGMII Implementation: Shared Clock Sources

If the PHY device sources the receiver SGMII stream synchronously from the shared oscillator (check PHY data sheet), then the RocketIO transceiver receives data at exactly the same rate as that used by the core: the receiver elastic buffer neither empties nor fills, having the same frequency clock on either side.

In this situation, the receiver elastic buffer does not under or overflow and the elastic buffer implementation in the RocketIO transceiver should be used to save logic resources.

Scenario 2

Now consider again the case illustrated by [Figure D-1](#). However, this time, assume that the clock sources used are both 50 ppm. Now the maximum frequency difference between the two devices is 100 ppm. It can be shown that this translates into a full clock period difference every 10000 clock periods, resulting in a requirement for 16 FIFO entries above and below the half-full point. It can be demonstrated that this provides reliable operation with the RocketIO Rx Elastic Buffers. Again, see the PHY data sheet to ensure that the PHY device sources the receiver SGMII stream synchronously to its reference oscillator.

Jumbo Frame Reception

A jumbo frame is an Ethernet frame that is deliberately larger than the maximum-size Ethernet frame allowed in the *IEEE802.3* specification. Jumbo frames require special consideration to reliably receive frames. [Table D-1](#) defines the maximum-size jumbo frames that can be received reliably when using the Receiver Elastic Buffer.

Table D-1: Maximum Frame Sizes for Fabric Rx Elastic Buffers (100 ppm Clock Tolerance)

Standard/Speed	Maximum Frame Size
1000BASE-X (1 Gb/s only)	280000
SGMII (1 Gb/s)	280000
SGMII (100 Mb/s)	28000
SGMII (10 Mb/s)	2800

Debugging Designs

This appendix defines a step-by-step debugging procedure to assist in the identification and resolution of any issues that might arise during each phase of the design process. It contains the following sections:

- [Debug Tools](#)
- [Simulation Debug](#)
- [Implementation and Timing Errors](#)
- [Hardware Debug](#)

If this appendix does not help to resolve the issue, see [Additional Resources](#) and [Technical Support](#) in Chapter 1 for additional support.

Debug Tools

There are many tools available to debug Ethernet MAC design issues. It is important to know which tools are useful for debugging various situations. This section references the following tools.

Example Design

Virtex®-5 FPGA Embedded Tri-Mode Ethernet MAC Wrappers come with a synthesizable example design complete with functional and post-place and route simulation test benches. Information on the example design can be found throughout this document.

ChipScope Pro Tool

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

Available Reference Boards

The ML505 and ML507 are Xilinx development boards supporting 10/100/1000 Mb/s Ethernet. The ML505 and ML507 boards can be used to prototype designs and establish that the core can communicate with the system.

Xilinx application note 957 describes a Virtex-5 Embedded Tri-Mode Ethernet MAC Hardware Demonstration Platform based on the Ethernet MAC wrapper and targeted to the ML505 and ML507. See www.xilinx.com/support/documentation for [XAPP 957](#) and supporting files.

Link Analyzers

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

- Spirent SmartBits
- IXIA brand 10/100/1000 Ethernet test chassis
- Wireshark (a free packet sniffer software application)

Simulation Debug

The simulation debug flow for ModelSim follows.

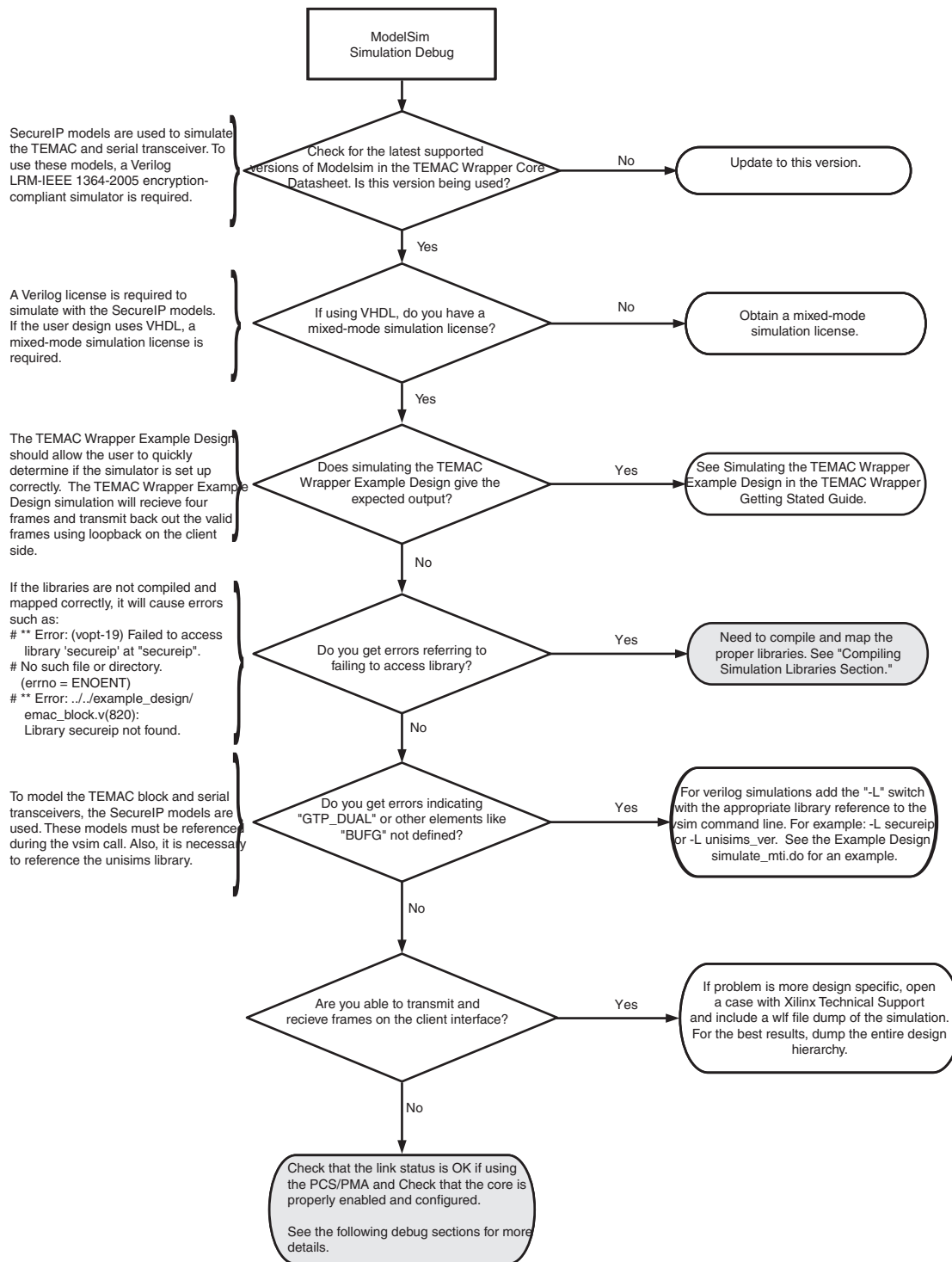


Figure E-1: Simulation Debug Flow Chart

Compiling Simulation Libraries

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

Xilinx Simulation Library Compilation Wizard

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

Compxlib

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

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

```
compxlib -s mti_se -arch virtex5 -l verilog -lib secureip -lib unisims  
-dir ./
```

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

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

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

For example:

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

Implementation and Timing Errors

The example design provided with the Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Wrappers comes complete with implementation scripts. For more details on using these scripts, see [Chapter 5, Detailed Example Design](#). If implementation or timing errors are encountered with the Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC, it is recommended to first try running the example design to see if the failures are seen there. If the failures do not exist in the example design, then differences between the example design and the design in which failures are seen can be compared.

Timing Failed for GMII/RGMII/MII OFFSET IN Constraint

To satisfy setup and hold requirements for these standards, fixed-mode IODELAYs are placed on the receive clock, data and control signals when using the GMII, RGMII, or MII wrapper files. In the example design UCF, the fixed value delays are set based on the pinout used in the example design. With a different pinout, the fixed DELAY value might have to be adjusted to still meet the setup and hold requirements. For more details on how to adjust this delay to meet setup and hold requirements, see [Appendix C, Constraining the Example Design](#).

Hardware Debug

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

- [General Checks](#)
- [Problems with Transmitting and Receiving Frames](#)
- [Link Bring-up Using 1000BASE-X or SGMII](#)
- [Problems with the MDIO](#)
- [Configuring the Ethernet MAC to the Correct Speed](#)

General Checks

- Ensure that all the timing constraints for the core were properly incorporated from the example design delivered from the CORE Generator™ software and are met during place and route.
- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue.
- Ensure that all clock sources are active and clean. If using DCMs in the design, ensure that all DCMs have obtained lock by monitoring the LOCKED port.

Problems with Transmitting and Receiving Frames

Problems with data reception or transmission can be caused by a wide range of factors. The following list contains common causes to check for:

- Verify that the whole TEMAC block is not being held in reset. The whole block is held in reset if the main reset input is asserted or if `CLIENTTEMAC#DCMLOCKED` is held low.
- Verify that both the receiver and transmitter are enabled and not being held in reset. For more information, see “Receiver and Transmitter Configuration Words” in the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide*, Chapter 4, “Host/DCR Bus Interfaces”.
- Verify that the Ethernet MAC is configured correctly and that the latest cores from the CORE Generator software or EDK are being used. Try running a simulation to check if the failure is hardware-specific.
- If using GMII or RGMII, check if setup and hold requirements are met using IDELAY components. For more information, see the section on debugging [Implementation and Timing Errors](#).
- Verify that the link is up between the PHY and its link partner. If using 1000BASE-X or SGMII configurations of the Ethernet MAC, see the [Link Bring-up Using 1000BASE-X or SGMII](#) section for more details.
- If using an external PHY, is data received correctly if the PHY is put in loopback? If so, the issue might be on the link between the PHY and its link partner.
- Check if the address filter is enabled. If frames are not being received correctly, try disabling the address filter to ensure that the frame is not being dropped by the

address filter. The output signals `EMAC#CLIENTRXDVLD` and `EMAC#CLIENTRXFRAMEDROP` can also be monitored to check if frames are dropped due to the address filter. For more information, see the Address Filtering section in the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

- Verify that the Ethernet MAC has been configured to operate at the correct speed negotiated with the PHY. For more information, see the [Configuring the Ethernet MAC to the Correct Speed](#) section.
- Are received frames being dropped by client logic because `EMAC#CLIENTRXBADFRAME` is asserted? See Frame Reception with Errors, in the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide* for details on why frames are marked bad by the Ethernet MAC. The ChipScope tool can be inserted to get more details on the bad frames.
- Add the ChipScope tool to the design to look at the RX and TX client and physical interface data signals, control signals and statistics vectors.

Link Bring-up Using 1000BASE-X or SGMII

Problems with Data Reception or Transmission

When no data is being received or transmitted:

- Ensure that a valid link has been established between the core and its link partner, either by auto-negotiation or manual configuration.
 - `EMAC#PHYSYNACQSTATUS` should be high to indicate that the `SYNC_ACQUIRED` state from the IEEE Std 802.3-2005, clause 36 state machine has been achieved.
 - If auto-negotiation is enabled, then PCS Status register bit 1.5 should be read to verify that auto-negotiation has completed. The auto-negotiation interrupt output can also be used to verify that auto-negotiation has completed.

If no link has been established, see the topics discussed in the next section.

- [Problems with Auto-Negotiation](#)
- [Problems in Obtaining a Link \(Auto-Negotiation Disabled\)](#)

Note: Transmission through the core is not allowed unless a link has been established. This behavior can be overridden by setting the Unidirectional Enable attribute.

- Ensure that the Isolate state has been disabled.

By default, the Isolate state is set by the attribute `EMAC#_PHYISOLATE`. The Isolate state can be changed by writing to PCS Control Register bit 0.10 after power-up. If the Isolate state is enabled, this results in no data transferred across the internal GMII interface between the PCS/PMA and MAC. See Physical Interface Attributes, Table 2-18 and Control Register, Table 5-3 and Table 5-15 in the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide* for more information.

If data is being transmitted and received between the core and its link partner, but with a high rate of packet loss, see [Problems with a High Bit Error Rate](#).

Problems with Auto-Negotiation

Determine whether auto-negotiation has completed successfully by doing one of the following.

- Poll the auto-negotiation completion bit 1.5 in "Status Register (Register 1)"

- Use the auto-negotiation interrupt port of the core (see 1000BASE-X Auto-Negotiation in the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide*)

If auto-negotiation is not completing:

1. Ensure that auto-negotiation is enabled in both the core and in the link partner (the device or test equipment connected to the core). Auto-negotiation cannot complete successfully unless both devices are configured to perform auto-negotiation. The auto-negotiation procedure requires that the auto-negotiation handshaking protocol between the core and its link partner, which lasts for several link timer periods, occurs without a bit error. If a bit error is detected, auto-negotiation restarts. Therefore, a link with an exceptionally high bit error rate might not be capable of completing auto-negotiation, or can lead to a long auto-negotiation period caused by the numerous restarts. If this appears to be the case, try the next step and see [Problems with a High Bit Error Rate](#).
2. Try disabling auto-negotiation in both the core and the link partner and see if both devices report a valid link and are able to pass traffic. If they do, it proves that the core and link partner are otherwise configured correctly. If they do not pass traffic, see the next section, [Problems in Obtaining a Link \(Auto-Negotiation Disabled\)](#).

Problems in Obtaining a Link (Auto-Negotiation Disabled)

Determine whether the device has successfully obtained a link with its link partner by doing the following:

- Monitoring the state of `EMAC#PHYSYNCACQSTATUS`. If this is logic '1,' then synchronization, and therefore a link, has been established.
- Reading bit 1.2, Link Status, in "Status Register (Register 1)" when using the MDIO management interface.

If the devices have failed to form a link then do the following:

- Ensure that auto-negotiation is disabled in both the core and in the link partner (the device or test equipment connected to the core).
- Monitor the state of the `PHYEMAC#SIGNALEDIT` input to the core. This should either be:
 - connected to an optical module to detect the presence of light. Logic '1' indicates that the optical module is correctly detecting light; logic '0' indicates a fault. Therefore, ensure that this is driven with the correct polarity, or
 - tied to logic '1' (if not connected to an optical module).

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

- See the section, [Problems with a High Bit Error Rate](#).

Serial Transceiver-Specific

- Ensure that the polarities of the TXN/TXP and RXN/RXP lines are not reversed. If they are, this can be fixed by using the TXPOLARITY and RXPOLARITY ports of the serial transceiver.
- Check that the serial transceiver is not being held in reset by monitoring the `mgt_tx_reset` and `mgt_rx_reset` signals between the core and the serial transceiver.
- Monitor the `RXBUFSTATUS` signal when auto-negotiation is disabled. If this is being asserted, the elastic buffer in the receiver path of the serial transceiver is either

underflowing or overflowing. This indicates a clock correction issue caused by differences between the transmitting and receiving ends. Check all clock management circuitry and clock frequencies applied to the core and to the serial transceiver.

Note: It is normal to see buffer errors during auto-negotiation because clock correction sequences are not sent during auto-negotiation. The PCS/PMA logic masks buffer errors during auto-negotiation and resets the RX buffer so that it recovers.

Problems with a High Bit Error Rate

Symptoms

The severity of a high-bit error rate can vary and cause any of the following symptoms:

- Failure to complete auto-negotiation when auto-negotiation is enabled.
- Failure to obtain a link when auto-negotiation is disabled in both the core and the link partner.
- High proportion of lost packets when passed between two connected devices that are capable of obtaining a link through auto-negotiation or otherwise. This can usually be accurately measured if the Ethernet MAC is attached to the Ethernet Statistics core.

Note: All bit errors detected by the PCS/PMA logic during frame reception show up as frame check sequence (FCS) errors in the Ethernet MAC statistics vector.

Debugging

- Compare the issue across several devices or PCBs to ensure that the issue is not a one-off case.
- Try using an alternative link partner or test equipment and then compare results.
- Try putting the core into loopback (both by placing the core into internal loopback, and by looping back the optical cable) and compare the behavior. The core should always be capable of auto-negotiating with itself and looping back its transmitter to receiver so direct comparisons can be made. If the core exhibits correct operation when placed into internal loopback, but not when loopback is performed via an optical cable, this might indicate a faulty optical module or a PCB issue.
- Try swapping the optical module on an erroneous device and repeat the tests.

Serial Transceiver-Specific Checks

Perform these additional checks when using a serial transceiver:

- Directly monitor the following ports of the serial transceiver by attaching error counters to them, or by triggering on them using the ChipScope tool or an external logic analyzer.

RXDISPERR
RXNOTINTABLE

These signals should not be asserted over the duration of a few seconds, minutes or even hours. If they are frequently asserted, it might indicate an issue with the serial transceiver. Consult *UG196, Virtex-5 FPGA GTP Transceivers* or *UG198, Virtex-5 FPGA GTX Transceivers* for debugging serial transceiver issues.

- Place the serial transceiver into parallel or serial loopback.
 - If correct operation is seen in serial loopback, but not when loopback is performed via an optical cable, it might indicate a faulty optical module or issues on the PCB between the serial transceiver pins and the optical module.

- If the core exhibits correct operation in serial transceiver parallel loopback but not in serial loopback, this might indicate a serial transceiver issue.
- Minor bit error rates can be solved by adjusting the transmitter TXPREEMPHASIS, TXDIFFCTRL and TERMINATION_CTRL attributes of the serial transceiver.

Problems with the MDIO

See "MDIO Implementation in the Ethernet MAC" in the *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide* for detailed information about performing MDIO transactions.

Things to check for:

- Ensure that the MDIO is driven properly and correctly terminated. Even if only using the internal MDIO interface, correct termination is needed to ensure the MDIO interface operates correctly.
- Check that the mdc clock is running and that the frequency is 2.5 MHz or less. If using the host interface to access the MDIO registers, the MDIO interface does not work until the clock frequency is set with CLOCK_DIVIDE. The MDIO clock with a maximum frequency of 2.5 MHz is derived from the host clock.
- Ensure that the TEMAC and PHY are not held in reset. Be sure to check the polarity of the reset to your external PHY. Many PHYs have an active-low reset.
- Read from a configuration register that does not have all 0s as a default. If all 0s are read back, the read was unsuccessful.
- If using the host interface to access the MDIO registers, check if the issue is just with the MDIO interface or if there are also problems reading and writing MAC registers with the host interface.
- If using the host interface to access the MDIO registers, make sure the HOSTMIIMSEL on the host interface is held until a read is complete.
- If accessing MDIO registers for the internal PCS/PMA, check that the PHYAD field placed into the MDIO frame matches the value placed on the PHYEMAC#PHYAD[4:0] port of the Ethernet MAC.
- If an external PHY is being used, check the PHY address. PHY address 0 is a global address for writing to all PHYs on the MDIO bus at the same time. If you have more than one PHY on the MDIO bus, there is contention reading address 0. The internal PCS/PMA responds to address 0 if it is not held in reset. This is the case even if the TEMAC is not configured for a 1000BASE-X or SGMII interface.
- Has a simulation been run? Verify in simulation and/or a ChipScope tool capture that the waveform is correct for accessing the host interface for a MDIO read/write. The Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Wrappers example design test bench generated with 1000BASE-X or SGMII configurations performs MDIO writes to disable auto-negotiation.

Configuring the Ethernet MAC to the Correct Speed

When operating in tri-mode, the PHY negotiates the highest speed available with its link partner. The speed of the Ethernet MAC can be set by the user client application after auto-negotiation completes by doing the following:

1. The user application can either monitor auto-negotiation interrupt from the external PHY or internal PCS/PMA, or poll for auto-negotiation, register bit 1.5, to complete via MDIO.

2. When auto-negotiation completes the user application can read the MDIO auto-negotiation registers to obtain the negotiated speed.
3. The user application then needs to set this speed in the Ethernet MAC configuration registers via the host interface.

If auto-negotiation is disabled, the Ethernet MAC, PHY, and the PHY's link partner must all be set to the same speed.