

# **Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper v1.5**

## ***Getting Started Guide***

UG545 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.

© 2009- 2011 Xilinx, Inc. All rights reserved. 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.

Date	Version	Revision
04/24/09	1.0	Initial Xilinx release.
06/24/09	1.1	Updated core to v1.2 and ISE® to v11.2. Added Virtex®-6 CXT support.
09/16/09	2.0	Updated core to v1.3 and ISE to v11.3. Added Virtex-6 HXT and Virtex-6 -1L support.
10/15/09	2.0.1	Added <a href="#">Appendix E, Debugging Designs</a> .
04/19/10	2.1	Updated core to v1.4 and ISE to v12.1.
03/01/11	2.2	Updated core to v1.5 and Xilinx tools to v13.1

# Table of Contents

---

Revision History .....	2
<b>Schedule of Figures</b> .....	7
<b>Preface: About This Guide</b>	
Guide Contents .....	9
Conventions .....	10
Typographical .....	10
Online Document .....	11
<b>Chapter 1: Introduction</b>	
System Requirements .....	13
About the Ethernet MAC Wrapper Core .....	13
Designs Using Serial Transceivers .....	13
Recommended Design Experience .....	14
Additional Resources .....	14
Technical Support .....	14
Feedback .....	14
Ethernet MAC Wrapper .....	14
Document .....	14
<b>Chapter 2: Licensing the Core</b>	
Before you Begin .....	15
License Options .....	15
Obtaining Your Full License Key .....	15
Installing Your License File .....	15
<b>Chapter 3: Quick Start Example Design</b>	
Overview .....	17
Generating the Ethernet MAC Wrapper .....	19
Implementing the Example Design .....	20
Running the Simulation .....	20
Functional Simulation .....	20
Virtex-6 Devices .....	20
VHDL Simulation .....	21
Verilog Simulation .....	21
Timing Simulation .....	21
VHDL Simulation .....	22
Verilog Simulation .....	22
Example Design Detail .....	22

## Chapter 4: Customizing the Core

<b>Ethernet MAC Wrapper Screens</b> .....	23
Interface Configuration Options: Screen 1 .....	23
Component Name .....	24
Physical Interface .....	24
Client Interface .....	25
Host and Management Interfaces .....	25
Transmitter and Receiver Configuration: Screen 2 .....	27
Initial Transmitter Configuration .....	28
Receiver Configuration .....	28
Initial Address Filter Configuration .....	29
MDIO Configuration: Screen 3 .....	30
Initial MDIO Configuration .....	30

## Chapter 5: Detailed Example Design

<b>Directory Structure and File Descriptions</b> .....	31
<project directory> .....	32
<project directory>/<component name> .....	32
<component name>/doc .....	33
<component name>/example_design .....	33
<component name>/example_design/client .....	34
<component_name>/example_design/client/fifo .....	34
<component_name>/example_design/physical .....	35
<component name>/implement .....	36
implement/results .....	37
<component name>/simulation .....	37
simulation/functional .....	38
simulation/timing .....	39
<b>Implementation and Test Scripts</b> .....	40
Setting up for Simulation .....	40
Virtex-6 Device Requirements .....	40
Implementation Scripts for Timing Simulation .....	40
Test Scripts For Timing Simulation .....	41
For ModelSim .....	41
For IES .....	41
For VCS .....	41
Test Scripts For Functional Simulation .....	42
For ModelSim .....	42
For IES .....	42
For VCS .....	42
<b>Example Design</b> .....	43
HDL Example Design .....	43
10 Mb/s, 100 Mb/s, 1 Gb/s Ethernet FIFO .....	44
rx_client_fifo .....	45
tx_client_fifo .....	45
Address Swap Module .....	46
Physical Interface .....	46
<b>Demonstration Test Bench</b> .....	47
Test Bench Functionality .....	47
Demonstration Test Bench Tasks .....	48
Changing the Test Bench .....	49

Changing Frame Data . . . . .	49
Changing Frame Error Status . . . . .	49
Changing the Tri-Mode Ethernet MAC Configuration . . . . .	49

## Appendix A: Using the Client Side FIFO

Overview of LocalLink Interface . . . . .	51
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 and SGMII . . . . .	55
1000BASE-X PCS/PMA with 16-bit Client Interface . . . . .	57
GMII/RGMII at 1000 Mb/s . . . . .	59
Multi-Speed Clocking . . . . .	60
SGMII at Multiple Speeds . . . . .	60
GMII/MII/RGMII at Multiple Speeds . . . . .	62
GMII/MII at Multiple Speeds with Clock Enable . . . . .	65
RGMII at Multiple Speeds with Clock Enable . . . . .	66

## Appendix C: Constraining the Example Design

Basic Constraints . . . . .	67
Clock Constraints . . . . .	67
Clock Constraints for MII Configurations . . . . .	67
Clock Constraints for GMII Configurations . . . . .	68
Clock Constraints for RGMII Configurations . . . . .	69
Clock Constraints for 1000BASE-X PCS/PMA Configurations . . . . .	70
Clock Constraints for SGMII Configurations . . . . .	70
Clock Constraints for the Host Interface . . . . .	71
Physical Interface Constraints . . . . .	71
Physical Interface Constraints for MII Configurations . . . . .	71
Physical Interface Constraints for GMII Configurations . . . . .	72
GMII IDELAY_VALUE Constraints . . . . .	73
Physical Interface Constraints for RGMII Configurations . . . . .	74
RGMII IDELAY_VALUE Constraints . . . . .	76
Physical Interface Constraints for 1000BASE-X PCS/PMA Configurations . . . . .	76
Physical Interface Constraints for SGMII Configurations . . . . .	77
LocalLink FIFO Constraints . . . . .	77

## Appendix D: SGMII Capabilities

SGMII Receiver Elastic Buffer . . . . .	79
FPGA Fabric Rx Elastic Buffer Requirement . . . . .	79
Analysis . . . . .	80
The Serial Transceiver Rx Elastic Buffer . . . . .	81

Closely Related Clock Sources .....	81
Jumbo Frame Reception .....	82
<b>SGMII / 1000BASE-X PCS/PMA Mode Switching</b> .....	82
Switching Modes and Speeds .....	83
Operational Requirements .....	83
Dynamic Switching .....	83
Host Interface Arbitration .....	83
Auto-Negotiation .....	84

## Appendix E: Debugging Designs

<b>Debug Tools</b> .....	85
Example Design .....	85
ChipScope Pro Tool .....	85
Available Reference Boards .....	86
Link Analyzers .....	86
<b>Simulation Debug</b> .....	87
Compiling Simulation Libraries .....	88
Xilinx Simulation Library Compilation Wizard .....	88
Compplib .....	88
<b>Implementation and Timing Errors</b> .....	89
Regional Clocking Errors in Map .....	89
Timing Failed for GMII/RGMII/MII OFFSET IN Constraint .....	90
<b>Hardware Debug</b> .....	90
General Checks .....	90
Problems with Transmitting and Receiving Frames .....	90
Link Bring-up Using 1000BASE-X or SGMII .....	91
Problems with Data Reception or Transmission .....	91
Problems with Auto-Negotiation .....	92
Problems in Obtaining a Link (Auto-Negotiation Disabled) .....	92
Problems with a High Bit Error Rate .....	93
Problems with the MDIO .....	94
Configuring the Ethernet MAC to the Correct Speed .....	95

# 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-6 Embedded Tri-Mode Ethernet MAC Wrapper Main Screen* ..... 19

## Chapter 4: Customizing the Core

*Figure 4-1: Interface Configuration* ..... 24

*Figure 4-2: Transmitter and Receiver Configuration* ..... 27

*Figure 4-3: MDIO Configuration* ..... 30

## Chapter 5: Detailed Example Design

*Figure 5-1: HDL Example Design* ..... 43

*Figure 5-2: Frame Transfer across LocalLink Interface* ..... 45

*Figure 5-3: Modification of Frame Data by Address Swap Module* ..... 46

*Figure 5-4: Demonstration Test Bench* ..... 47

## 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 with 8-bit Client Interface or SGMII Clocking at 1000 Mb/s* ... 56

*Figure B-2: PCS/PMA Clocking When Using the 16-bit Client Interface* ..... 58

*Figure B-3: GMII/RGMII Clocking at 1000 Mb/s* ..... 59

*Figure B-4: SGMII Clocking at 10/100/1000 Mb/s* ..... 61

*Figure B-5: GMII Clocking at 10/100/1000 Mb/s* ..... 62

*Figure B-6: RGMII Clocking at 10/100/1000 Mb/s* ..... 63

*Figure B-7: MII Clocking at 10/100 Mb/s* ..... 64

*Figure B-8: GMII/MII Clocking at 10/100/1000 Mb/s with Clock Enables* ..... 65

*Figure B-9: RGMII Clocking at 10/100/1000 Mb/s with Clock Enable* ..... 66

## Appendix C: Constraining the Example Design

*Figure C-1: Input GMII Timing* ..... 73

*Figure C-2: RGMII Input Timing* ..... 76

## Appendix D: SGMII Capabilities

<i>Figure D-1: SGMII Implementation: Separate Clock Sources</i> .....	80
<i>Figure D-2: SGMII Implementation: Shared Clock Sources</i> .....	81
<i>Figure D-3: Host Interface Arbiter and Serial Mode Switching Plug-in</i> .....	84

## Appendix E: Debugging Designs

<i>Figure E-1: Simulation Debug Flow Chart</i> .....	87
--	----



# About This Guide

---

The *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper v1.5 Getting Started Guide* provides information about generating, customizing and simulating wrappers for the embedded Tri-Mode Ethernet MAC blocks in Virtex®-6 FPGA devices. This guide also describes running the design files through implementation using 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-6 FPGA Embedded Tri-Mode Ethernet MAC wrapper and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- [Chapter 2, Licensing the Core](#) provides information about licensing the core.
- [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 Capabilities](#) defines the SGMII receiver elastic buffer and SGMII / 1000BASE-X PCS/PMA mode switching capabilities for the core.
- [Appendix E, Debugging Designs](#) 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.

## 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
<b>Courier bold</b>	Literal commands that you enter in a syntactical statement	<b>ngdbuild</b> <i>design_name</i>
<b>Helvetica bold</b>	Commands that you select from a menu	<b>File → Open</b>
	Keyboard shortcuts	<b>Ctrl+C</b>
<i>Italic font</i>	Variables in a syntax statement for which you must supply values	<b>ngdbuild</b> <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 <b>bus [7:0]</b> , they are required.	<b>ngdbuild</b> [ <i>option_name</i> ] <i>design_name</i>
Braces { }	A list of items from which you must choose one or more	<b>lowpwr</b> = { <b>on</b>   <b>off</b> }
Vertical bar	Separates items in a list of choices	<b>lowpwr</b> = { <b>on</b>   <b>off</b> }
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	<b>allow block</b> <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 <a href="#">Guide Contents</a> for details. See "Title Formats" in Chapter 1 for details.
<a href="#">Blue, underlined text</a>	Hyperlink to a website (URL)	Go to <a href="http://www.xilinx.com">www.xilinx.com</a> for the latest speed files.



## Introduction

---

This chapter introduces the Virtex®-6 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 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 v5.0 32-bit/64-bit (with Workstation option)
- SuSE Linux Enterprise (SLE) desktop and server v10.1 32-bit/64-bit

#### Software

- ISE® v13.1 software

### About the Ethernet MAC Wrapper Core

The Ethernet MAC wrapper is included in the latest IP Update on the Xilinx® IP Center. 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 Serial Transceivers

The Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC uses GTX serial transceivers. Throughout this guide, the generic term *serial transceiver* is used to represent the GTX serial transceivers.

## 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-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*, available from the [Virtex-6 FPGA product page](#).

## Technical Support

The fastest method for obtaining specific technical support for the Ethernet MAC wrapper is through the [www.xilinx.com/support](http://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-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper Data Sheet*, *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper Getting Started Guide*, and the *Virtex-6 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](http://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](http://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®-6 FPGA Ethernet MAC Wrapper. However, if you are using ISE v11.2 software or older, follow the instructions below for obtaining a license key before you use the core in your design. The Virtex-6 FPGA Ethernet MAC 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.

## License Options

The Virtex-6 FPGA Ethernet MAC Wrapper is made available with a Full License key in ISE v11.2 software and older. After installing the required Xilinx® ISE software and IP updates, install 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 Full License Key

To obtain a Full license key for ISE v11.2 software or older follow these instructions. In ISE v11.3 software and later, the license key requirement has been removed.

1. Navigate to the product page for this core:  
[www.xilinx.com/products/ipcenter/V6\\_Embedded\\_TEMAC\\_Wrapper.htm](http://www.xilinx.com/products/ipcenter/V6_Embedded_TEMAC_Wrapper.htm)
2. Click **Get License**.
3. Follow the instructions to install the required Xilinx® ISE software and IP updates, and generate the required license key on the Xilinx Product Download and Licensing Site.

## Installing Your License File

An email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document, which is available at [www.xilinx.com/support/documentation/dt\\_ise11-1.htm](http://www.xilinx.com/support/documentation/dt_ise11-1.htm).





# Quick Start Example Design

---

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

## Overview

The Ethernet MAC wrapper consists of the following:

- An instance-level wrapper file that assigns the attributes of the Ethernet MAC to the values selected in the Core Generator GUI software. In addition, unused inputs are tied to the appropriate logic level 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 the selected physical interface. If selected, the host interface arbiter is also instantiated at this level.
  - The LocalLink wrapper connects the transmit and receive client interfaces of the Ethernet MAC to its own LocalLink FIFOs.
  - The example design wrapper connects the FIFOs so that data received at the client is 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 MMCM and 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® v13.1 software, Cadence Incisive Enterprise Simulator (IES) 10.2, Mentor Graphics ModelSim 6.6d, and Synopsys VCS and VCS MX 2010.06.

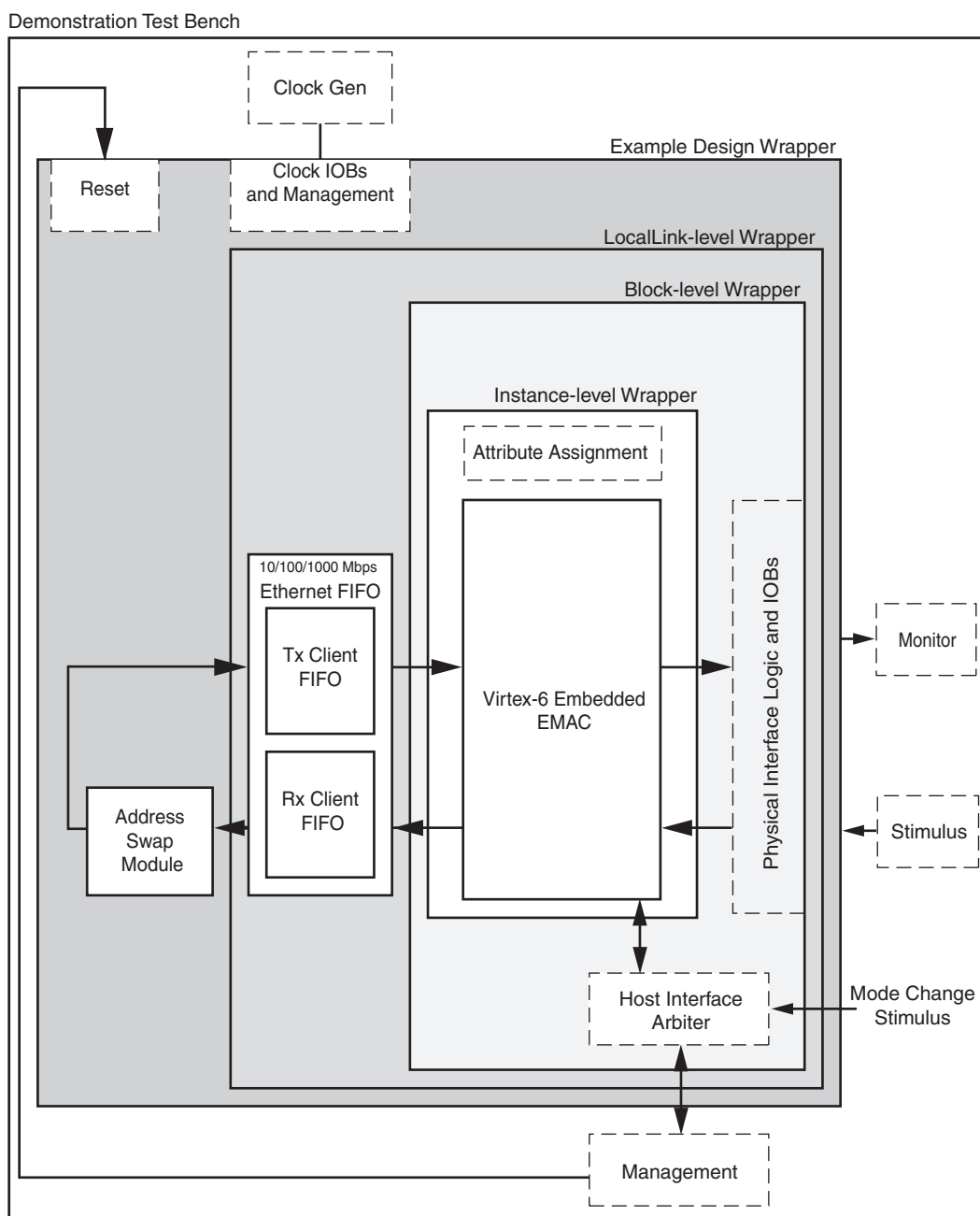


Figure 3-1: Default Example Design and Test Bench

## 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. From the Project Options window, set the following:
  - In the Part screen, choose the Virtex®-6 part appropriate for your application.

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

  - In the Generation screen, select either VHDL or Verilog for Design Entry. Other options should be left in the default setting.
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-6 Embedded Tri-Mode Ethernet MAC Wrapper**. The initial customization screen appears.

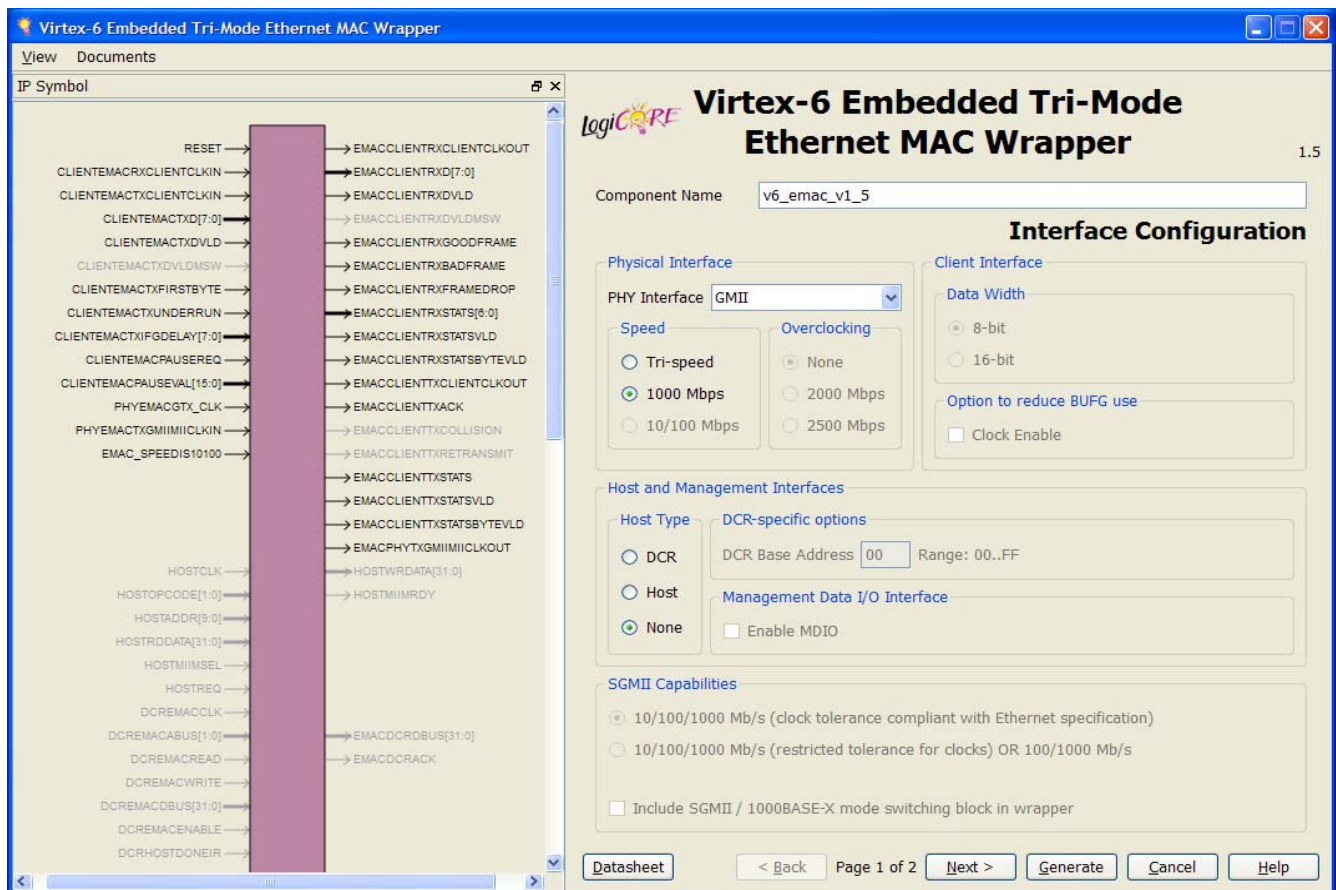


Figure 3-2: Virtex-6 Embedded Tri-Mode Ethernet MAC Wrapper Main Screen

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 instance-level wrapper and its supporting files, including the example design, are generated in your CORE Generator software 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 following examples, <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, places, routes, and performs static timing analysis on the example design. It then creates a gate-level netlist HDL file, along with an associated timing information (SDF) file. The resulting files are placed in the results directory.

## 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](http://www.xilinx.com/support/software_manuals.htm). In addition, use the following guidelines to determine the simulator required for your design:

### Virtex-6 Devices

Virtex-6 device designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. See [Overview, page 17](#) for supported simulators. When running VHDL simulations, a mixed-language 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 libraries:  

```
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
```

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 MAC.

## 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 libraries:  

```
ModelSim> vmap unisims_ver <path to compiled libraries>/unisims_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 MAC.

## 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](http://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 libraries:  

```
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 MAC.

## 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 SimPrims\_ver and SecureIP libraries:  

```
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
VCS> ./simulate_vcs.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 MAC.

## 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®-6 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 either two or three screens, depending on configuration.

- **Interface Configuration Options: Screen 1.** Used to name the core, select the desired speed and physical, client, host and management interface characteristics.
- **Transmitter and Receiver Configuration: Screen 2.** Used to set the default transmitter, receiver, and address filter configurations.
- **MDIO Configuration: Screen 3.** This screen is only displayed if the Enable MDIO option is selected on the first screen.

#### Interface Configuration Options: Screen 1

Use the initial configuration screen to name the core, select the desired speed as well as physical, client, host and management interface characteristics.

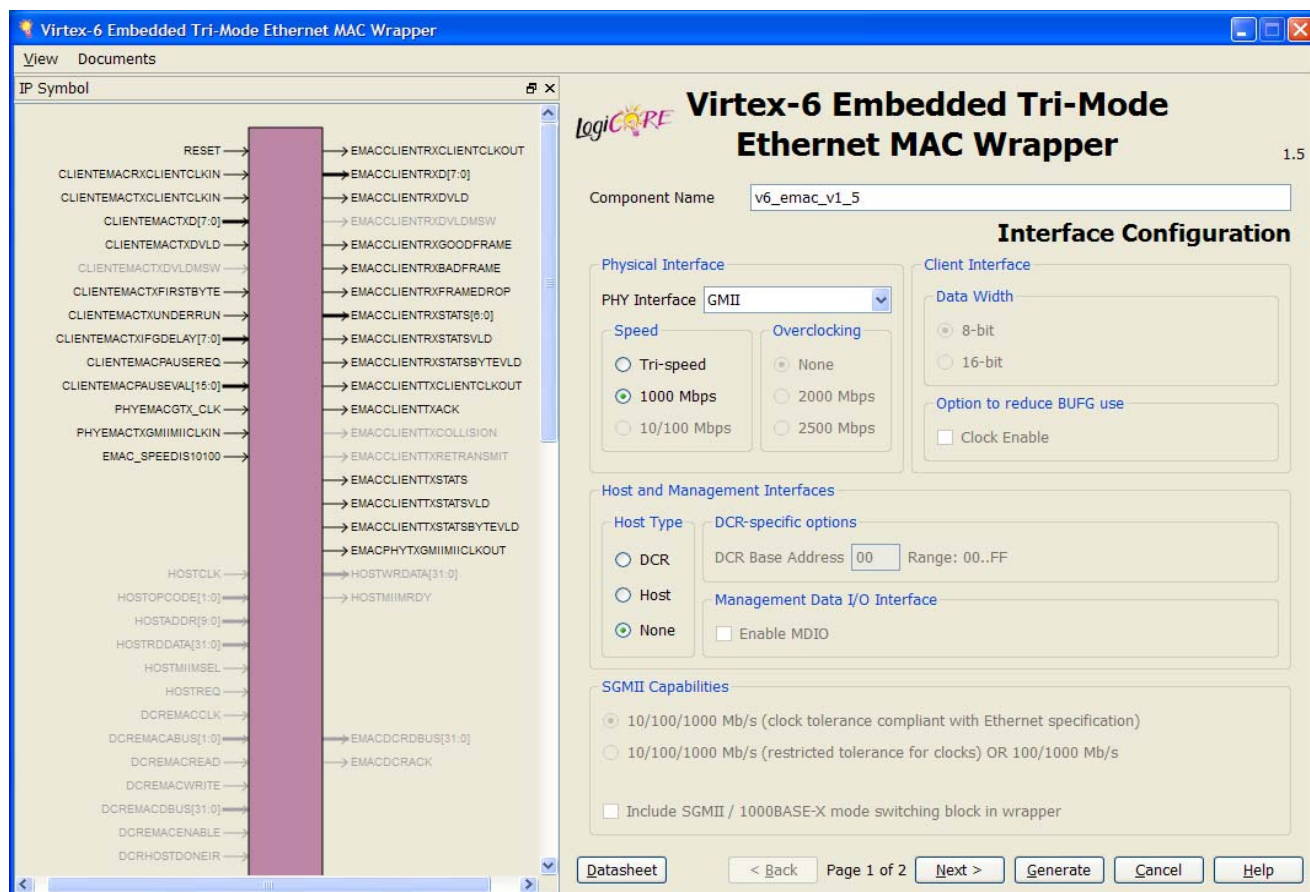


Figure 4-1: Interface Configuration

## 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 “\_.”

## Physical Interface

### PHY Interface

Select the physical 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 1000 Mb/s only.
- **10/100 Mb/s.** Configures the core to run at 10 or 100 Mb/s.

### Overclocking

In supported devices, allows for overclocking when the 1000BASE-X PCS/PMA physical interface is selected. See [Single-Speed Clocking in Appendix B](#) for more details on overclocking implementation.

- **2000 Mb/s.** Overclock the 1000BASE-X physical interface to 2000 Mb/s.
- **2500 Mb/s.** Overclock the 1000BASE-X physical interface to 2500 Mb/s.

## Client Interface

### Data Width

Configures the data width at the client interface.

- **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 Ethernet MAC to be overclocked if desired.

### Option to Reduce BUFG Use

Allows for the selection of an advanced clocking scheme to reduce the usage of global buffers.

- **Clock Enable.** Selecting Clock Enable reduces the number of BUFGs by requiring the user logic to use a separate clock-enable signal. See the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide* for more information about determining the Clock Enable signal setup. This option is available for 10 or 100 Mb/s operation using the MII physical interface as well as for Tri-speed operation in GMII and RGMII physical interfaces.

## Host and Management Interfaces

### Host Type

Allows access to the host configuration registers via one of the following interfaces:

- **Host.** Accesses the host interface directly, through the fabric.
- **DCR (Device Control Registers).** Accesses the host registers through the DCR interface. An internal bus bridge translates DCR commands to compatible host bus signaling.
- **None.** No run time access to the host interface is possible. The Ethernet MAC operates based on the attributes set in the GUI.

### DCR-specific Options

- **DCR Base Address.** Enter a unique DCR base address for the Ethernet MAC.

## Management Data I/O Interface

- **Enable MDIO.** When selected, the MDIO option enables the MDIO ports on the Ethernet MAC to access the registers in the internal and external PHY. When the MDIO option is selected, an MDIO configuration screen appears before generating the core. When unselected, the MDIO configuration screen is not displayed.

## SGMII Capabilities

Select the desired mode of operation to enable the appropriate receive buffering.

- **10/100/1000 Mb/s (clock tolerance compliant with Ethernet specification).** 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 serial transceiver, subsequently consuming extra logic resources. However, this default mode provides reliable SGMII operation under all conditions.
- **10/100/1000 Mb/s (restricted tolerance for clocks) or 100/1000 Mb/s.** Uses the receiver elastic buffer present in the serial 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 and favored because of its lower logic utilization.
- **Include SGMII / 1000BASE-X PCS/PMA mode switching block in wrapper.** The Ethernet MAC supports switching between SGMII and 1000BASE-X modes of operation via configuration registers. If this option is selected, a block is additionally included in the wrappers, which facilitates mode switching based on a single, level input.

For detailed information about SGMII capabilities, see [Appendix D, SGMII Capabilities](#)

## Transmitter and Receiver Configuration: Screen 2

The next configuration screen defines the default transmitter, receiver, and address filter configurations. Some of these options are overwritten when running simulation using the demonstration test bench. See [Demonstration Test Bench Tasks in Chapter 5](#) for more details.

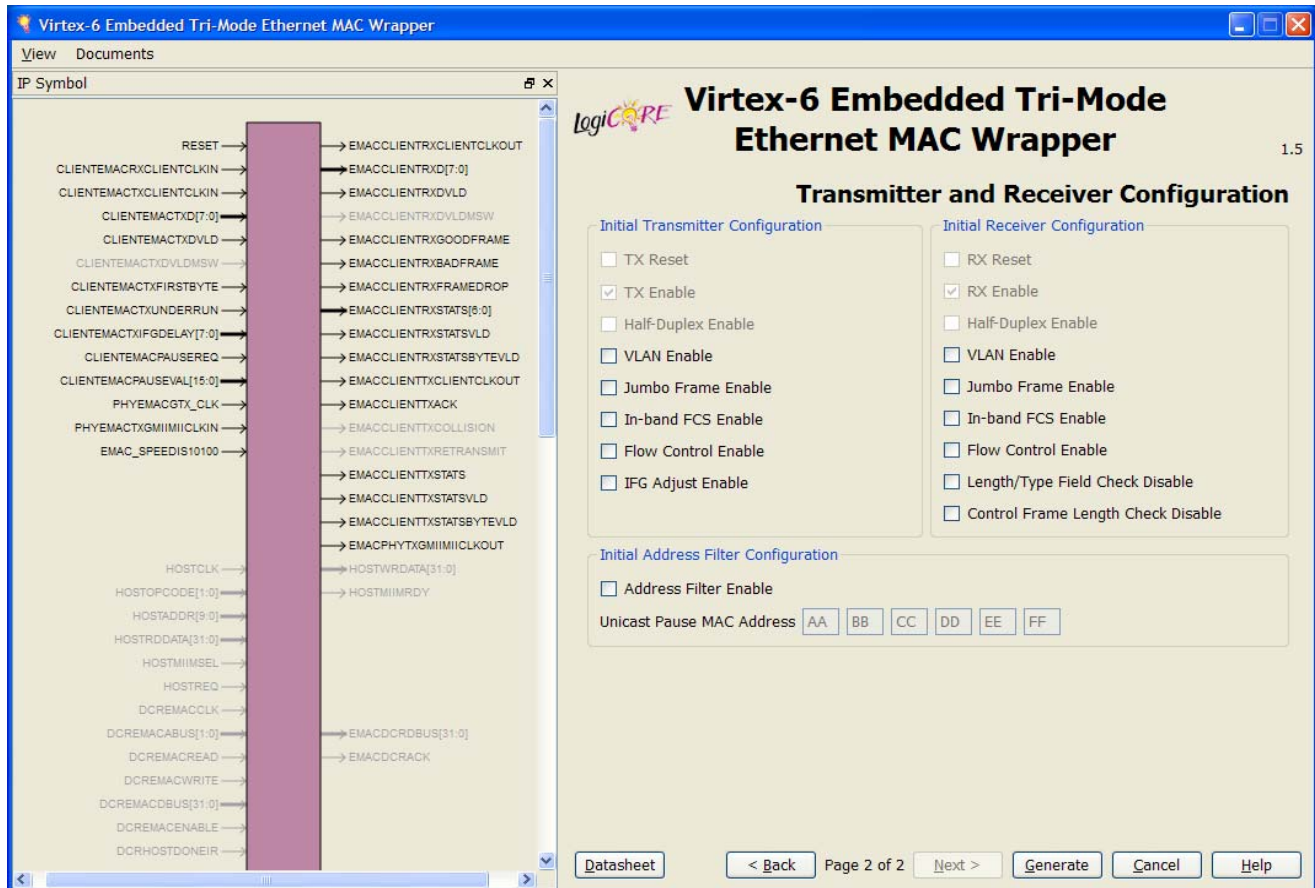


Figure 4-2: Transmitter and Receiver Configuration

## Initial 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 host interface access, if enabled. For more information, see “*Configuration Registers*,” in the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

- **TX Reset.** When selected, places the transmitter in reset. When Host type is set to None, this option is unselected and cannot be changed.
- **TX Enable.** When selected, enables the transmitter. When Host type is set to None, this option is selected and cannot be changed.
- **Half-Duplex Enable.** When selected, the transmitter operates in half-duplex mode (applicable for MII, GMII, or RGMII physical interfaces at 10 or 100 Mb/s only). When unselected, the transmitter operates in full-duplex mode.
- **VLAN Enable.** When selected, the transmitter allows transmission of the VLAN-tagged frames, as specified in *IEEE Std 802.3-2005*.
- **Jumbo Frame Enable.** When selected, the transmitter sends frames greater than the maximum length specified in the *IEEE Std 802.3-2005*. When unselected, the transmitter sends only frames less than the specified maximum length.
- **In-band FCS Enable.** When selected, this bit causes the Ethernet MAC transmitter to expect that the FCS field is supplied by the client.
- **Flow Control Enable.** When selected, the transmission of flow control PAUSE frames is enabled.
- **IFG Adjust Enable.** When selected, the transmitter reads the value of `CLIENTEMACTXIFGDELAY` at the start of frame transmission and adjusts the IFG. This option is applicable for full-duplex operation only.

## 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 host interface access, if enabled. For more information, see “*Configuration Registers*,” in the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

- **RX Reset.** When selected, places the receiver in reset. When Host type is set to None, this option is unselected and cannot be changed.
- **RX Enable.** When selected, enables the receiver. When Host type is set to None, this option is selected and cannot be changed.
- **Half-Duplex Enable.** When selected, the receiver operates in half-duplex mode (applicable for MII, GMII, or RGMII physical interfaces at 10 or 100 Mb/s only). When unselected, the receiver operates in full-duplex mode.
- **VLAN Enable.** When selected, the receiver accepts VLAN-tagged frames, as specified in *IEEE Std 802.3-2005*. The maximum accepted payload length increases by four bytes.
- **Jumbo Frame Enable.** When selected, the Ethernet MAC receiver accepts frames over the maximum length specified in the *IEEE Std 802.3-2005* 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 of each frame is checked for correctness.
- **Flow Control Enable.** When selected, flow control PAUSE frames are received and acted upon.
- **Length/Type Field Check Disable.** When selected, disables the Length/Type field check on the frame.
- **Control Frame Length Check Disable.** When selected, control frames larger than the legal minimum frame size are accepted.

### Initial Address Filter Configuration

- **Address Filter Enable.** When selected, received frames are subject to the address filter.
- **Unicast Pause MAC Address.** The value entered by you is used by the Ethernet MAC 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, a MAC address of AA-BB-CC-DD-EE-FF is entered as FF-EE-DD-CC-BB-AA.

## MDIO Configuration: Screen 3

The MDIO Configuration screen is only displayed if the 1000BASE-X PCS/PMA or SGMII PHY interface is selected and the Enable MDIO option is selected in the [Management Data I/O Interface](#) section of the first EMAC configuration screen. Some of these options are overwritten when running simulation using the demonstration test bench. See [Demonstration Test Bench Tasks in Chapter 5](#) for more details.

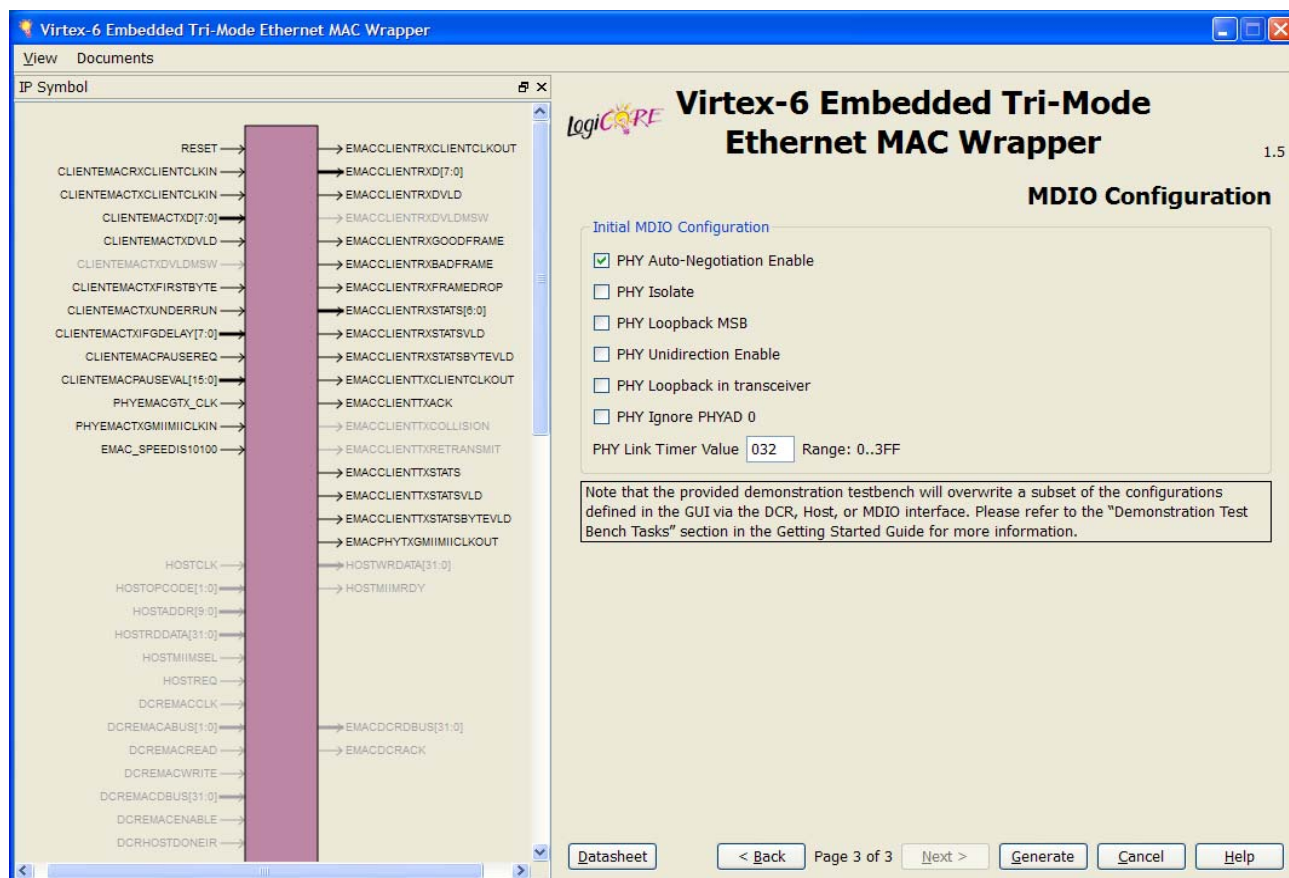


Figure 4-3: MDIO Configuration

### Initial MDIO Configuration

- **PHY Auto-Negotiation 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 serial transceiver. Otherwise loopback occurs in the Ethernet MAC.
- **PHY Ignore PHYAD 0.** If selected, the PHY ignores the MDIO broadcast address, PHYAD 0.
- **PHY Link Timer Value.** Programmable auto negotiation link timer value.

The PHY Reset and PHY Power-down MDIO options need to be set manually in the Ethernet MAC instance-level wrapper. These options cannot be configured through the GUI. For details on the correlation between the hexadecimal link timer value and the actual time, see the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.













## Detailed Example Design

---

This chapter provides detailed information about working with the example design, including a description of the 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®-6 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 a 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 or VHDL design files
-  [<component name>/example\\_design/client](#)  
Support files for the example client loopback logic and host interface arbiter, if applicable
-  [<component\\_name>/example\\_design/client/fifo](#)  
Files for the FIFO instances in the LocalLink client
-  [<component\\_name>/example\\_design/physical](#)  
Files for the physical interface of the Ethernet MAC
-  [<component name>/implement](#)  
Implementation script files
-  [implement/results](#)  
Results directory, created after implementation scripts are run, and contains implementation 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 software for each core that it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator software.
<component_name>_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 changes and updates beyond other documentation sources.

**Table 5-2: Component Name Directory**

Name	Description
<project_dir>/<component_name>	
v6_emac_readme.txt	Virtex-6 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-6 FPGA Embedded Tri-Mode Ethernet MAC, see the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

**Table 5-3: Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
v6_emac_ds710.pdf	Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper Data Sheet.
v6_emac_gsg545.pdf	Virtex-6 FPGA 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 43](#) for more information. The main Ethernet MAC wrapper files 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 instance-level wrapper file.
<component_name>_block.v[hd]	Block-level Ethernet MAC wrapper with instantiation of physical interface circuitry and the host interface arbiter, if applicable.
<component_name>_locallink.v[hd]	An intermediate-level wrapper 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 clocking resource instantiation.
<component_name>_example_design.ucf	UCF for the design. See <a href="#">Appendix C, Constraining the Example Design</a> 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. If the SGMII / 1000BASE-X PCS/PMA mode switching block is enabled, this directory also contains the files necessary to support mode switching automation.

**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.
host_if_arbiter.v[hd]	The host interface arbiter manages host interface accesses. Generated if the SGMII / 1000BASE-X PCS/PMA mode switching block is enabled.
host_if_plugin_sms.v[hd]	The serial mode switching block is a plug-in to the host interface arbiter and executes mode switching commands. Generated if the SGMII / 1000BASE-X PCS/PMA mode switching block is enabled.

[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 44.

**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]	Generated if the GMII physical interface is selected for the Ethernet MAC.
mii_if.v[hd]	Generated if the MII physical interface is selected for the Ethernet MAC.
rgmii_if.v[hd]	Generated if the RGMII v1.3 physical interface is selected for the Ethernet MAC.
rgmii_v2_0_if.v[hd]	Generated if the RGMII v2.0 physical interface is selected for the Ethernet MAC.
v6_gtxwizard.v[hd]	If an SGMII or 1000BASE-X PCS/PMA interface is selected, this is the intermediate-level serial transceiver wrapper file.
v6_gtxwizard.xco	Generated if an SGMII or 1000BASE-X PCS/PMA interface is selected. This XCO file contains the configuration parameters necessary to regenerate the v6_gtxwizard.v[hd] and v6_gtxwizard_gtx.v[hd] files using the CORE Generator software.
v6_gtxwizard_gtx.v[hd]	If an SGMII or 1000BASE-X PCS/PMA interface is selected, this is the instance-level serial transceiver wrapper file.
v6_gtxwizard_top.v[hd]	If an SGMII or 1000BASE-X PCS/PMA interface is selected, this file connects the serial transceiver wrappers to the physical interface of the Ethernet MAC.

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

Name	Description
rx_elastic_buffer.v[hd]	If the Tri-speed SGMII interface and SGMII Capabilities 10/100/1000 Mb/s (no clock constraints required) options are selected (Screen 1 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 block RAM.

[Directory Structure and File Descriptions](#)

## <component name>/implement

The implement directory contains the core implementation script files.

Table 5-8: Implement Directory

Name	Description
<project_dir>/<component_name>/implement	
implement.bat	A Windows batch file that processes the example design through the Xilinx® tool flow.
implement.sh	A Linux shell script that processes the example design through the Xilinx tool flow.
xst.scr	The XST script file for the top-level example design.
xst.prj	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 files through the Xilinx implementation tools. After these scripts are run, results and timing simulation files appear in the directory.

**Table 5-9: Results Directory**

Name	Description
<b>&lt;project_dir&gt;/&lt;component_name&gt;/implement/results</b>	
routed.v[hd]	The back-annotated SimPrim based Verilog or VHDL design. Used for timing simulation.
routed.sdf	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
<b>&lt;project_dir&gt;/&lt;component_name&gt;/simulation</b>	
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 MAC via the selected management interface.
phy_tb.v[hd]	The physical interface test bench. This stimulates the receiver ports and monitors the transmitter ports of the physical interface. This is instantiated in demo_tb.vhd.

[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
<b>&lt;project_dir&gt;/&lt;component_name&gt;/simulation/functional</b>	
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.
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
<b>&lt;project_dir&gt;/&lt;component_name&gt;/simulation/timing</b>	
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 window 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-6 Device Requirements

Virtex-6 device designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. See [Overview in Chapter 3](#) for supported simulators. When running VHDL simulations, a mixed-language 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 synthesized 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 or 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**, or **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

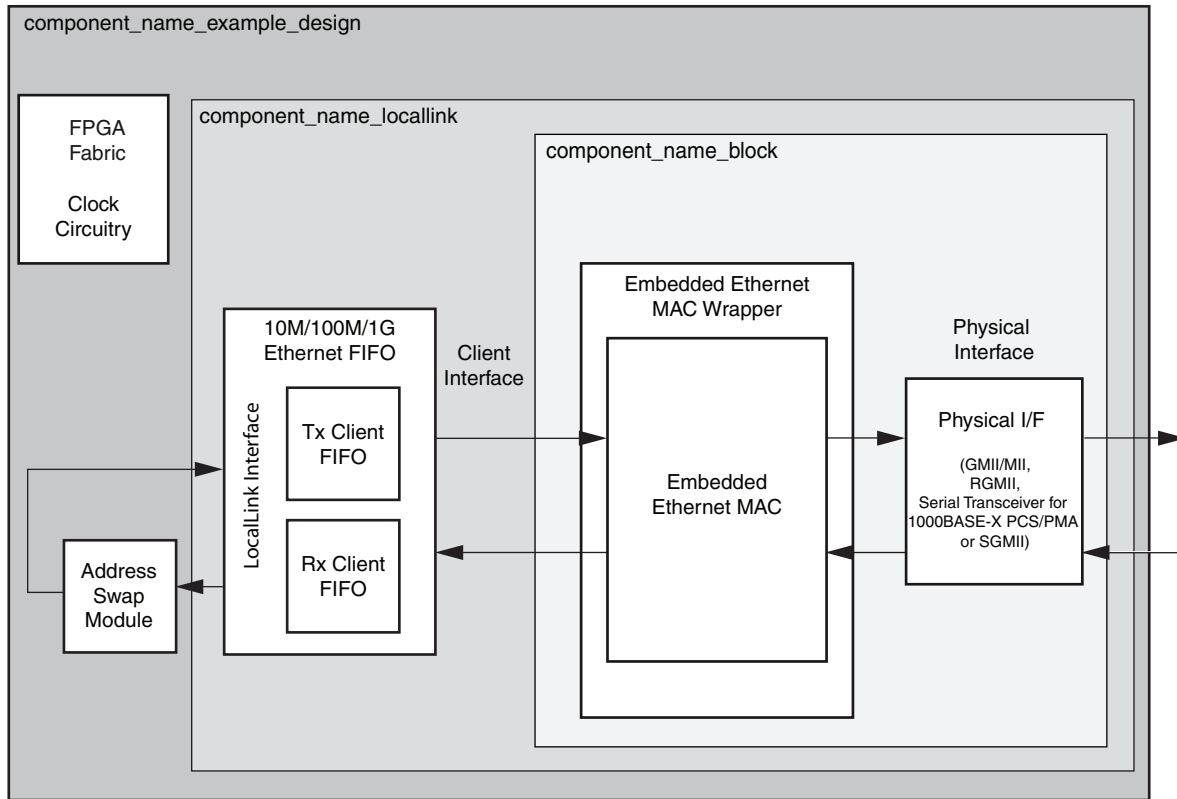


Figure 5-1: HDL Example Design

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:

- The Ethernet MAC instance-level wrapper
- An instance of the block-level Ethernet MAC wrapper containing GMII/MII, RGMII, SGMII or 1000BASE-X PCS/PMA interface logic, and the host interface arbiter if applicable.
- An instance of the LocalLink-level 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 an MMCM and 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](#).

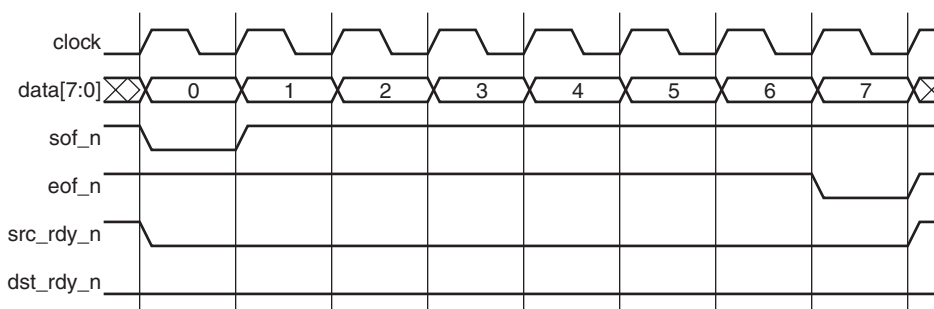


Figure 5-2: Frame Transfer across LocalLink Interface

### rx\_client\_fifo

The `rx_client_fifo` is built around a dual port block RAM, 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 you, (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 a dual port block RAM, 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

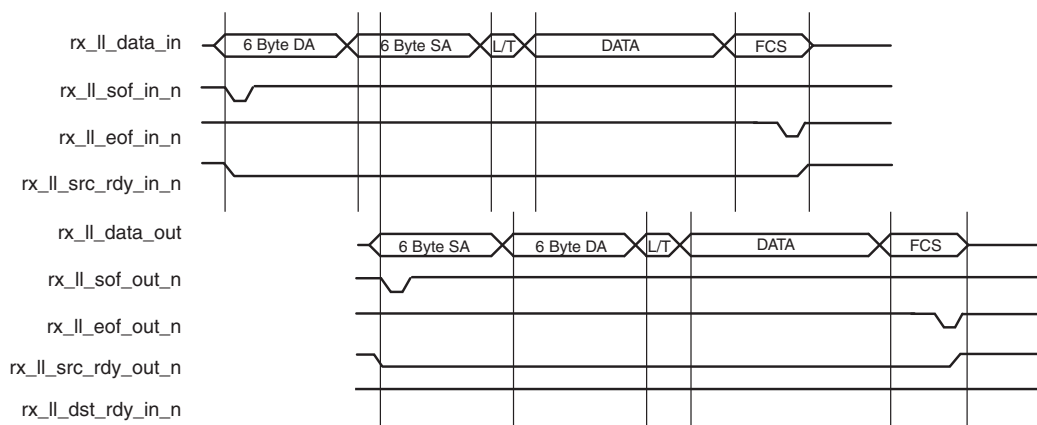


Figure 5-3: Modification of Frame Data by Address Swap Module

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

Appropriate physical interface logic is provided, which 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 GMII, IODELAYs are also instantiated on the receiver data input. These are configured in FIXED mode and align the received data with the clock.
- 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 2 ns required by the specification.
- For 1000BASE-X PCS/PMA or SGMII, this component instantiates and connects the serial transceiver.

# 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.

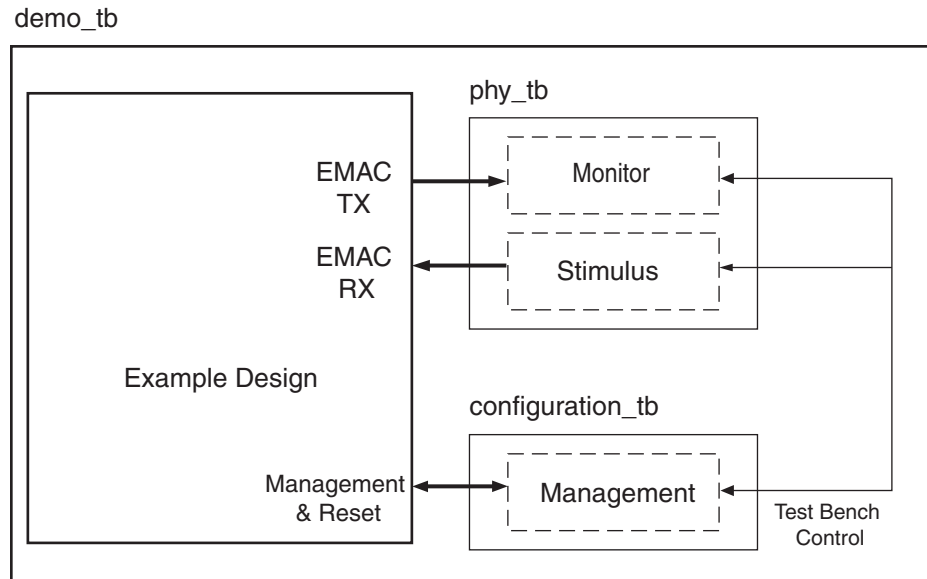


Figure 5-4: Demonstration Test Bench

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/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/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 MAC through the configuration vector
- Semaphores to indicate configuration status to the top level test bench

The physical layer test bench (`phy_tb.vhd`, `phy_tb.v`) consists 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 Ethernet MAC is configured through the management or configuration interface, and:
  - Sets up the MDC clock frequency
  - Disables auto-negotiation in SGMII and 1000BASE-X PCS/PMA modes (overwriting GUI configuration)
  - Disables flow control (overwriting GUI configuration)
  - Disables receiver and transmitter reset configuration registers (overwriting GUI configuration)
  - Disables receiver and transmit half duplex mode (overwriting GUI configuration)
  - Disables the PHY isolate feature for SGMII and 1000BASE-X PCS/PMA modes (overwriting GUI configuration)
  - Disables the transceiver and Ethernet MAC loopback feature for SGMII and 1000BASE-X PCS/PMA modes (overwriting GUI configuration)
  - Disables the unidirectional enable bit in PCS/PMA management register for SGMII and 1000BASE-X PCS/PMA modes (overwriting GUI configuration)
  - Enables transmitter and receiver. The configuration test bench then sets the speed of the Ethernet MAC.
- If the Ethernet MAC is selected to run at 1000 Mb/s or in Tri-Speed mode, the following four frames are pushed into the Ethernet MAC 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
- The frames received at the transmitter of the 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 Ethernet MAC is 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 Ethernet MAC is 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.



- If applicable, the `BASE_X_ENABLE` signal is asserted, and later deasserted, in order to command the Ethernet MAC to switch its operating mode from SGMII to 1000BASE-X and back again. Additional frames are sent after each switch. This demonstrates the function of the serial mode switching plug-in.

## 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 MAC 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 MAC 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 a dual port block RAM, providing a memory capacity of 4096 bytes in each FIFO.

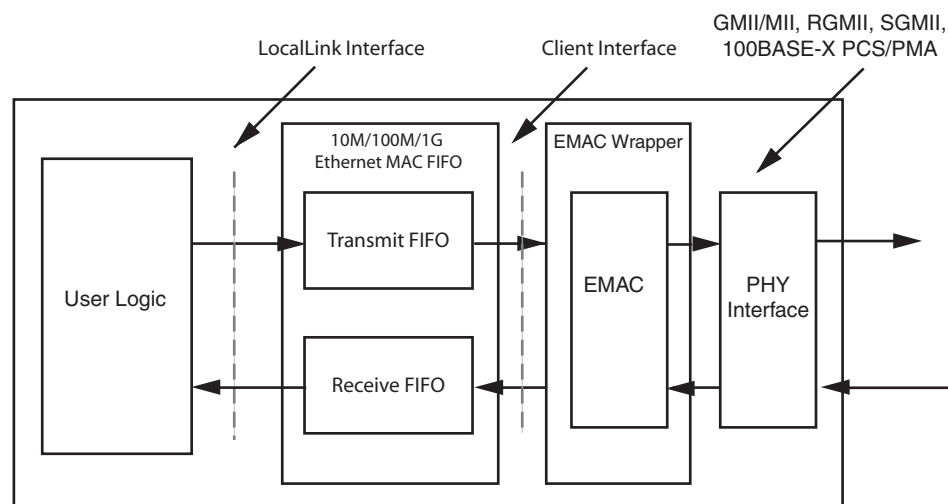


Figure A-1: Typical 10M/100M/1G Ethernet FIFO Implementation

## 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.

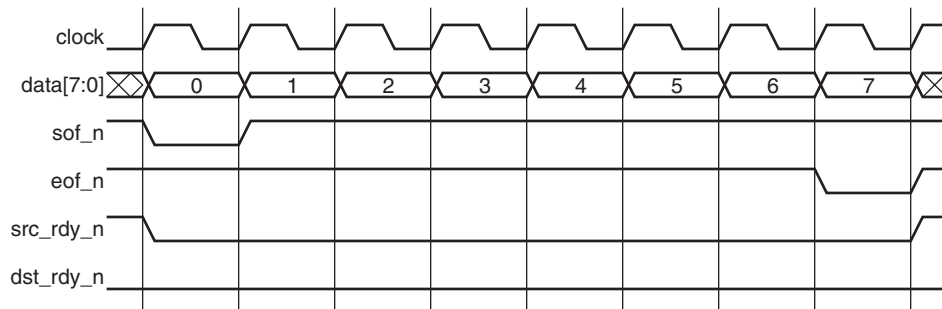


Figure A-2: Frame Transfer across LocalLink Interface

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.

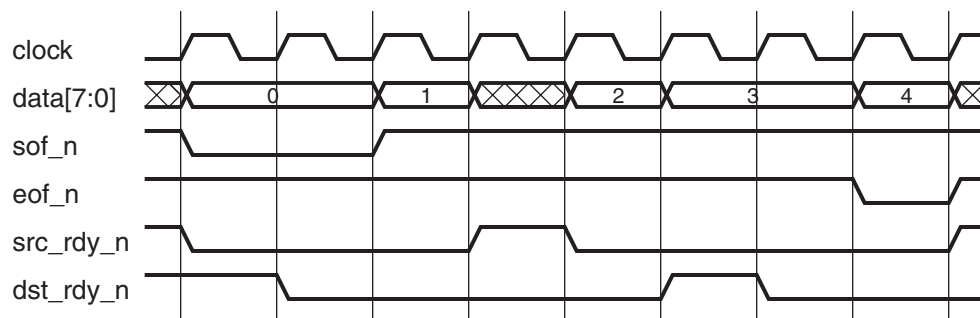


Figure A-3: Frame Transfer with Flow Control

## 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®-6 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 Ethernet MAC. 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-6 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 Ethernet MAC, 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 FIFO can also operate at 156.25 MHz when a supported overclocking mode is in use. 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[hd]` file. For more information about Ethernet MAC clock management, including detailed block diagrams of all supported clocking schemes, see the *Virtex®-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

## Single-Speed Clocking

### 1000BASE-X PCS/PMA and SGMII

In 1000BASE-X PCS/PMA (8-bit client interface) and SGMII modes ([Figure B-1](#)), the user supplies a high quality differential clock to the serial 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 `TXOUTCLK` 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.

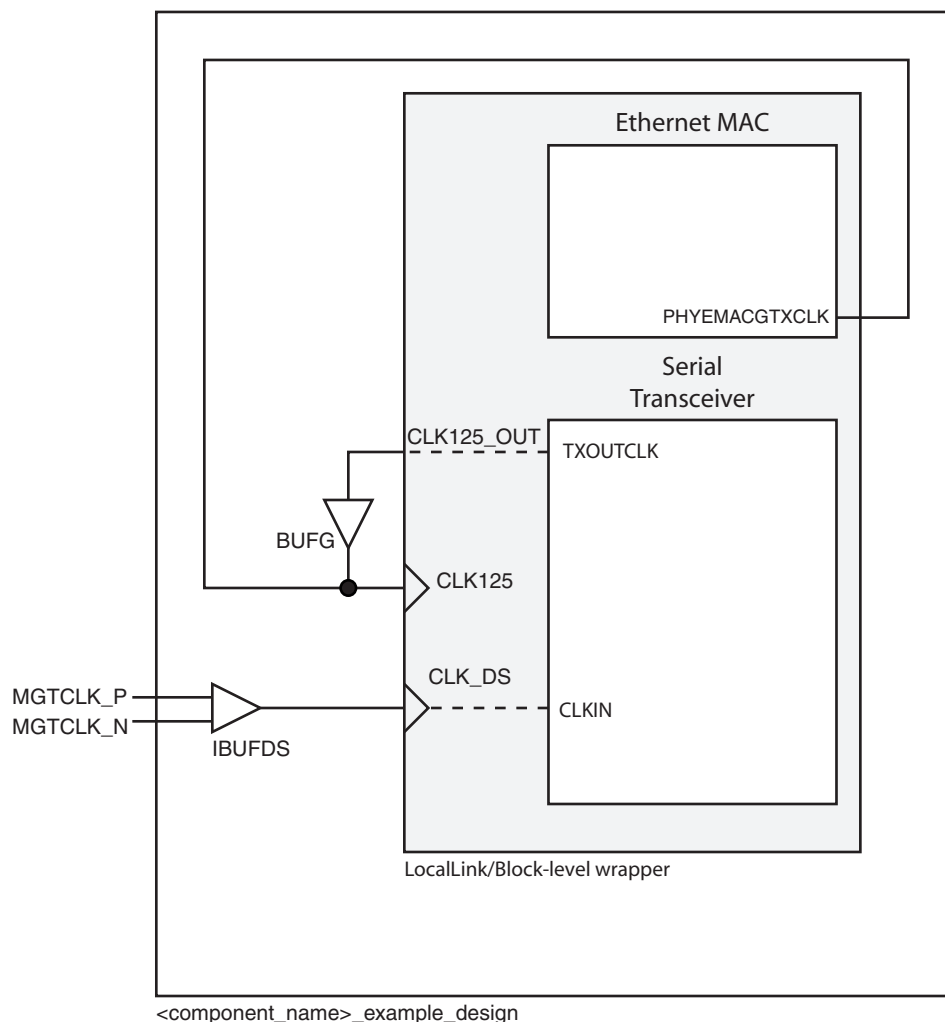


Figure B-1: PCS/PMA with 8-bit Client Interface or SGMII Clocking at 1000 Mb/s



## 1000BASE-X PCS/PMA with 16-bit Client Interface

When in 1000BASE-X PCS/PMA mode with the 16-bit client interface, the Ethernet MAC is capable of operating at 1000, and, in supported devices, 2000 or 2500 Mb/s.

When operating with the 16-bit client interface, an additional clock (CLK2X) is input to the wrappers. This is used to clock the serial transceiver and the physical side of the Ethernet MAC. CLK2X is generated from an MMCM, which is driven by the TXOUTCLK output of the transceiver and can be shared between multiple instantiations of the wrappers. The frequency of CLK2X depends on the rate in use as follows:

- For 1000 Mb/s, CLK2X is 125 MHz.
- For 2000 Mb/s, CLK2X is 250 MHz.
- For 2500 Mb/s, CLK2X is 312.5 MHz.

As in the 1000BASE-X PCS/PMA scheme in [Figure B-2](#), the client logic is driven by the clock supplied on the CLK125 port. This can be shared between multiple instantiations of the wrappers. For 2500 Mb/s operation only, the serial transceiver reference clock (MGTCLK\_P/N) should be supplied at 156.25 MHz. As a result, CLK125 also operates at 156.25 MHz in this mode.

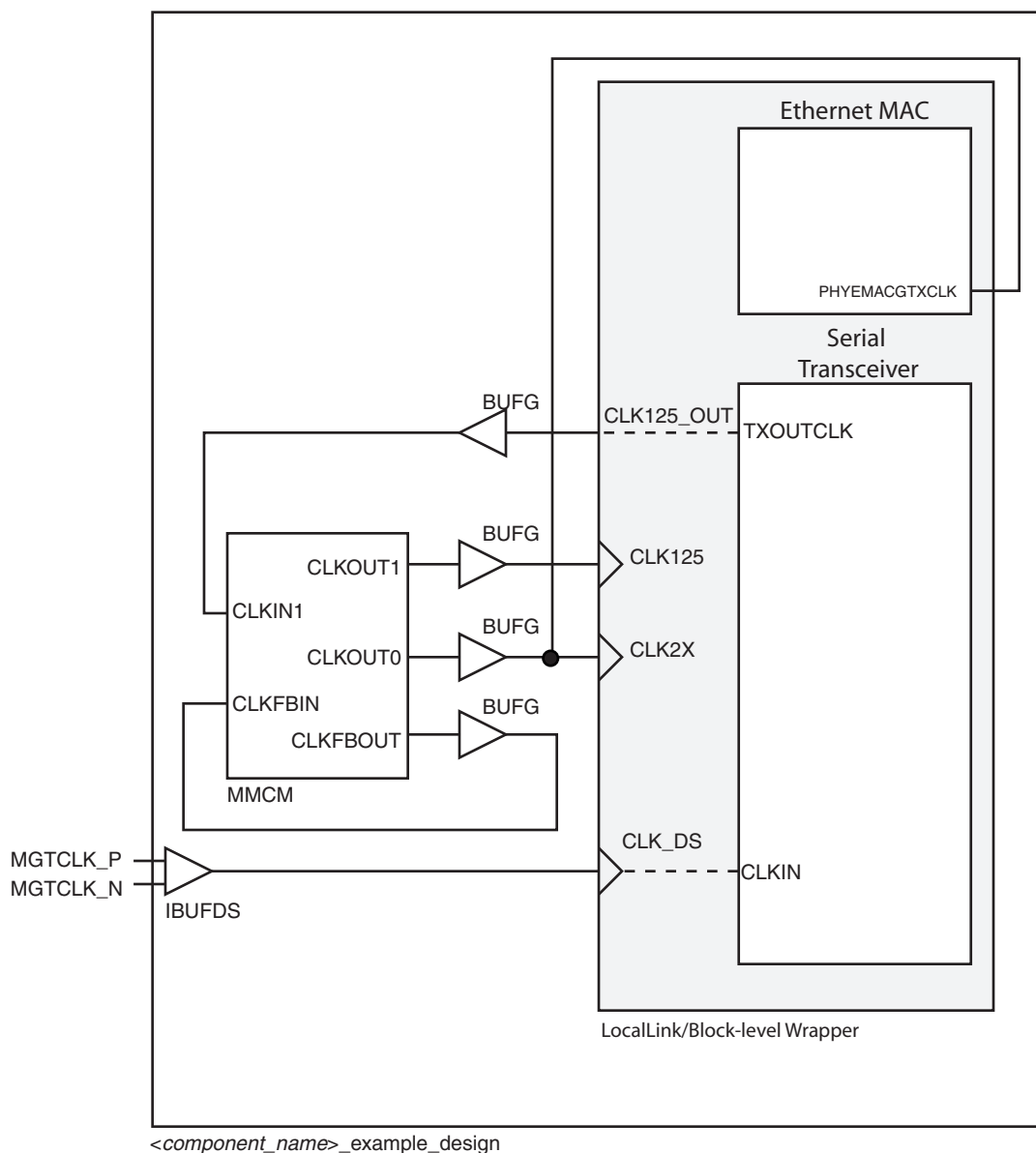


Figure B-2: PCS/PMA Clocking When Using the 16-bit Client Interface

## GMII/RGMII at 1000 Mb/s

Figure B-3 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 IODELAY element. The IODELAY is used to align the clock to the data inputs as they enter the FPGA. The delayed clock is routed through BUFIO and BUFR clock buffers and cannot be shared between multiple Ethernet MACs.

**Caution!** Use of the BUFIO and BUFR regional clock buffers impose certain location requirements on PHY-side receiver data and clock pins, and the Embedded TEMAC instance. For more information on regional clocking for the Ethernet MAC, see Pinout Requirements in Appendix A of the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

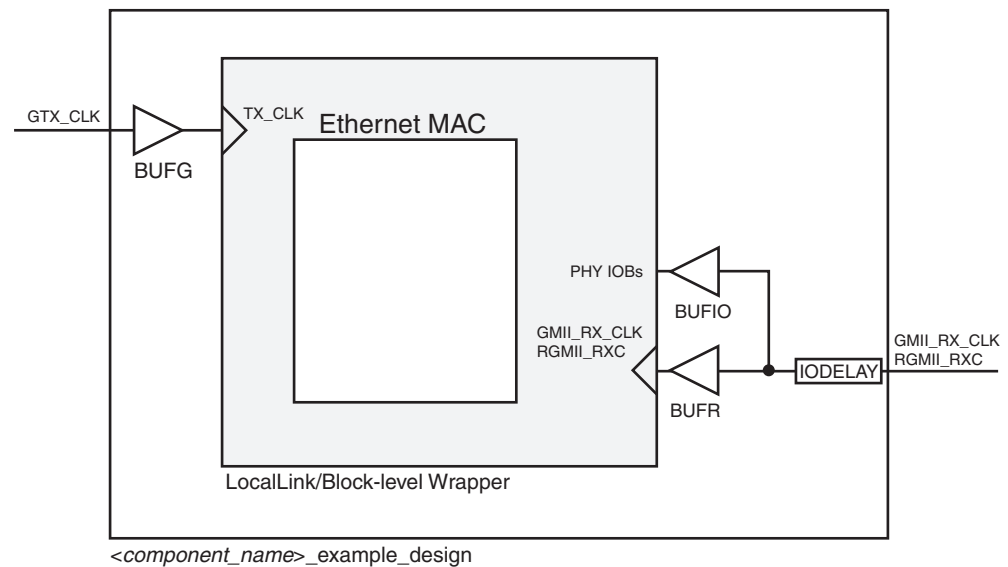


Figure B-3: GMII/RGMII Clocking at 1000 Mb/s

## Multi-Speed Clocking

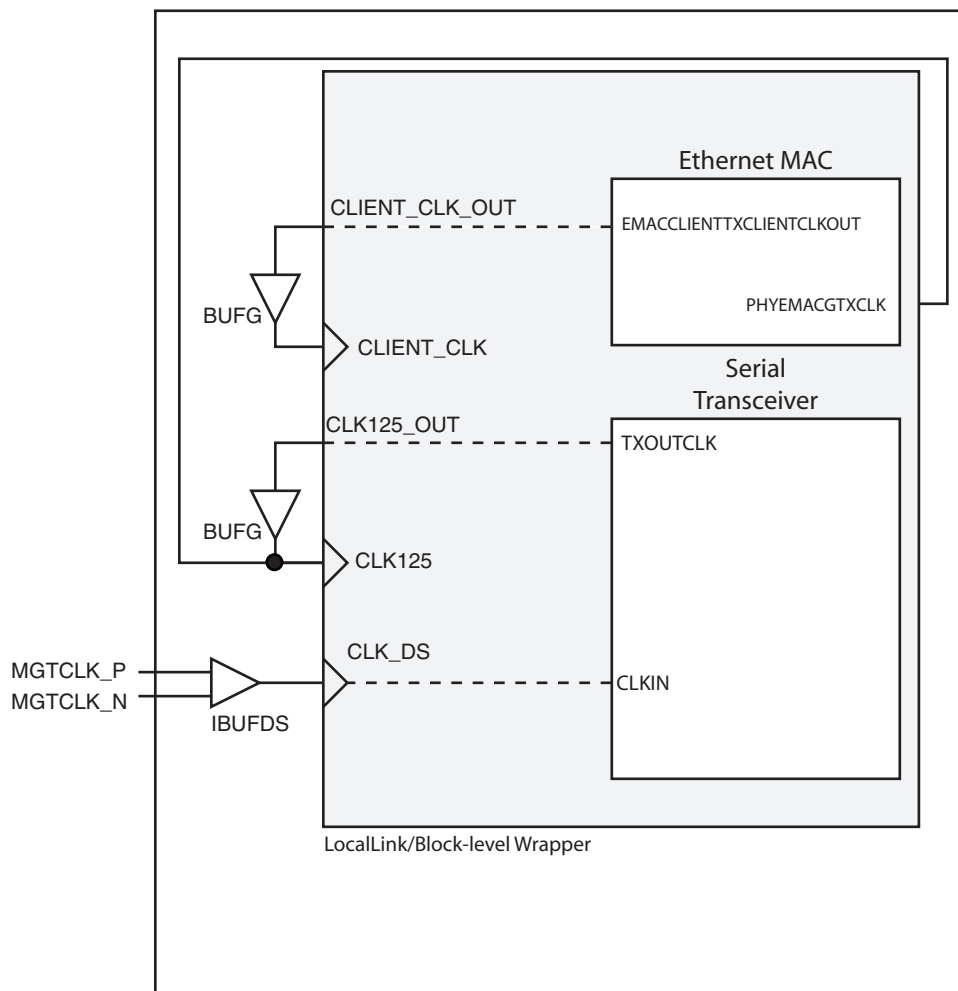
This section illustrates the clocking for the Ethernet MAC wrapper when operating at multiple speeds. For most interfaces, the Ethernet MAC supplies a clock output that can be used to drive the client logic at all speeds. The frequency of the clock output is dependent on the setting of the speed selection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

For the implementation of some interfaces the Ethernet MAC `EMACSPEDIS10100` 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-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

### SGMII at Multiple Speeds

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-4](#). These are used to clock the serial 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 `EMACCLIENTTXCLIENTCLKOUT` 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-4](#) shows the supplied clocking scheme.



<component\_name>\_example\_design

Figure B-4: SGMII Clocking at 10/100/1000 Mb/s

## 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-5, Figure B-6, and Figure B-7 show the default methods, where the user supplies a reference clock and the PHY interface clocks, and the Ethernet MAC generates the client interface clocks. In the case of GMII, a BUFGMUX is used to select between the MII\_TX\_CLK and GTX\_CLK based on the operating speed.

Because these methods use a large amount of clocking resources, Xilinx recommends that you use the Clock Enable method shown in Figure B-8, and Figure B-9.

**Caution!** Use of the BUFIO and BUFR regional clock buffers impose certain location requirements on PHY-side receiver data and clock pins, and the Embedded TEMAC instance. For more information on regional clocking for the Ethernet MAC, see Pinout Requirements in Appendix A of the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

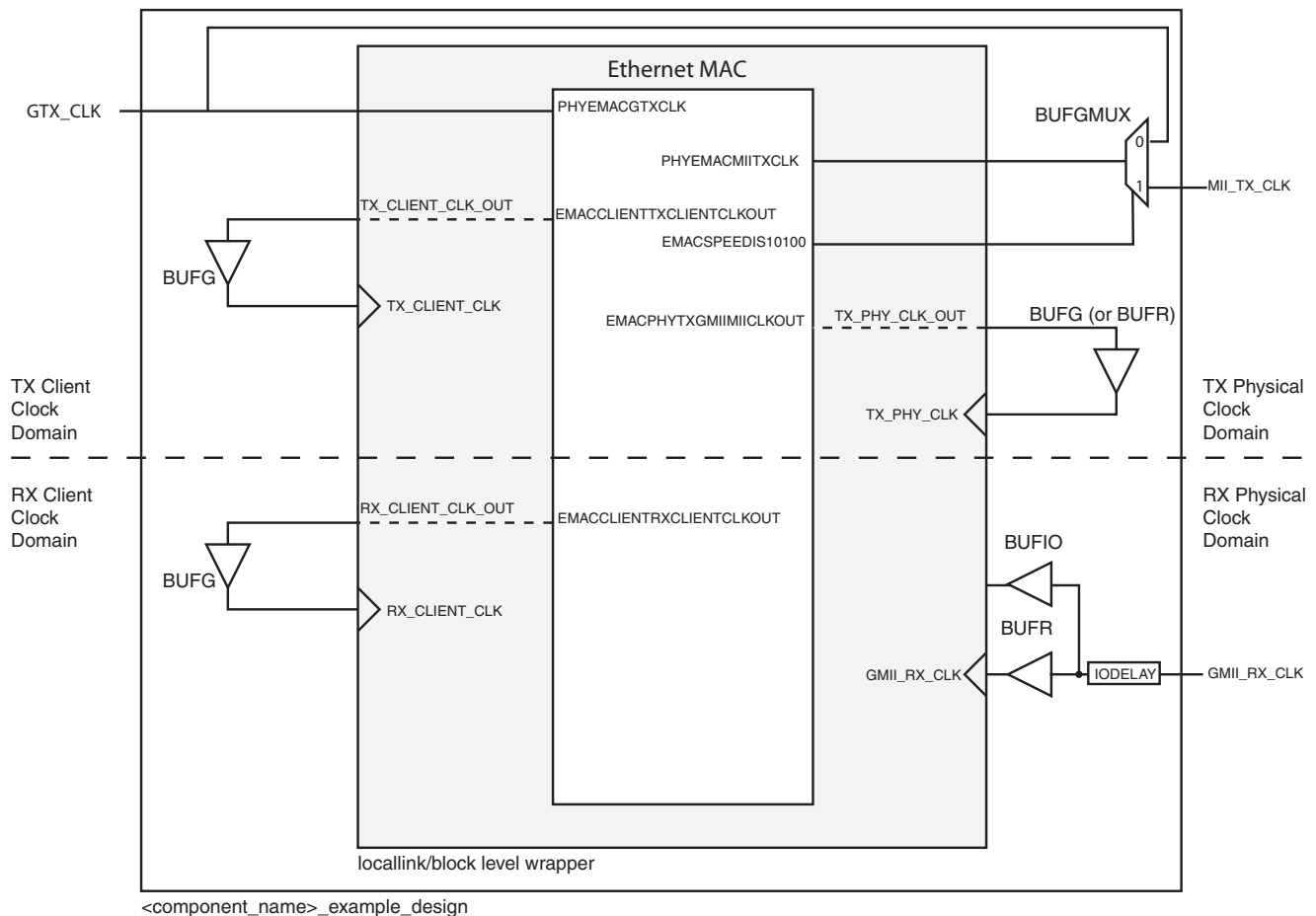


Figure B-5: GMII Clocking at 10/100/1000 Mb/s

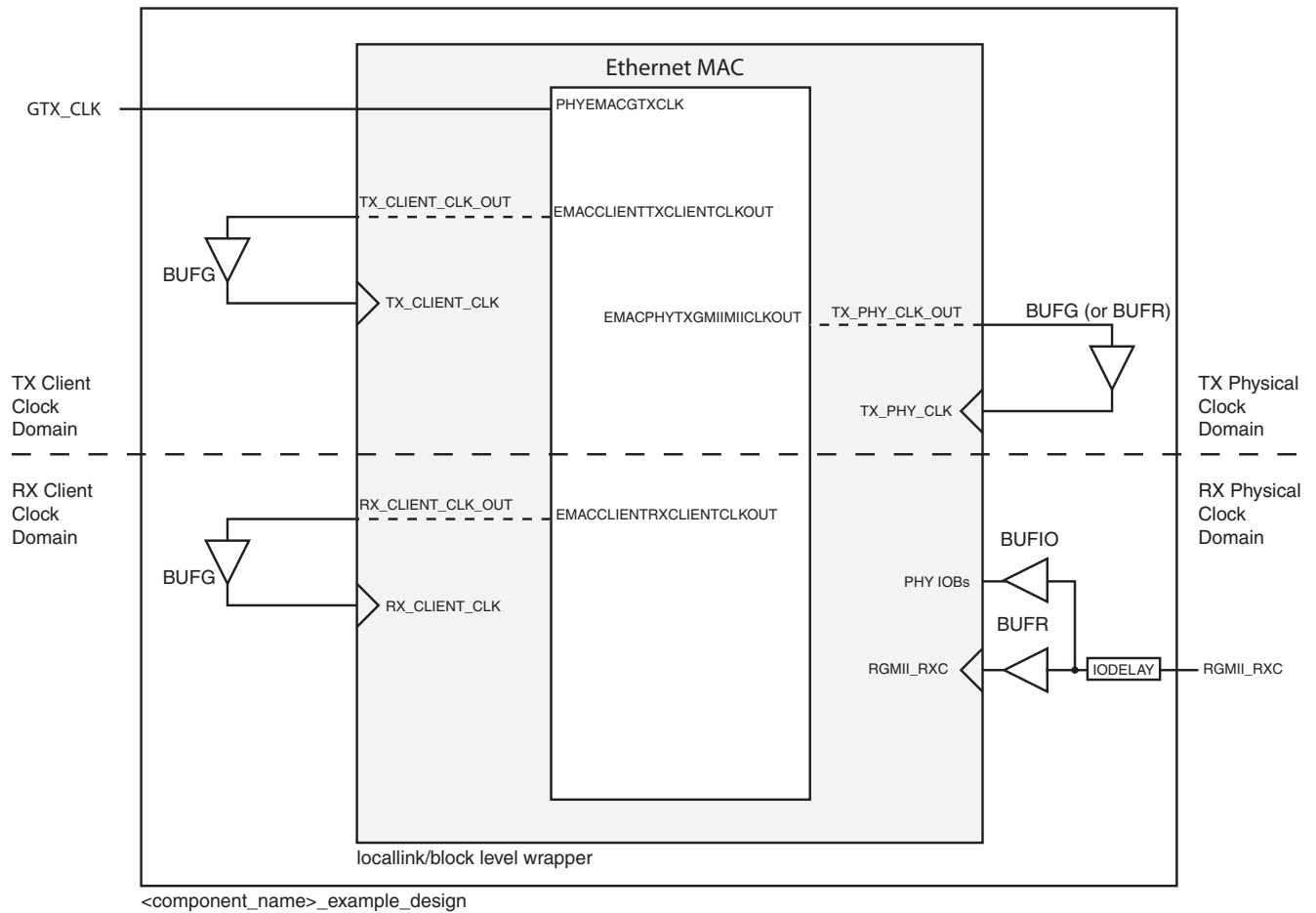
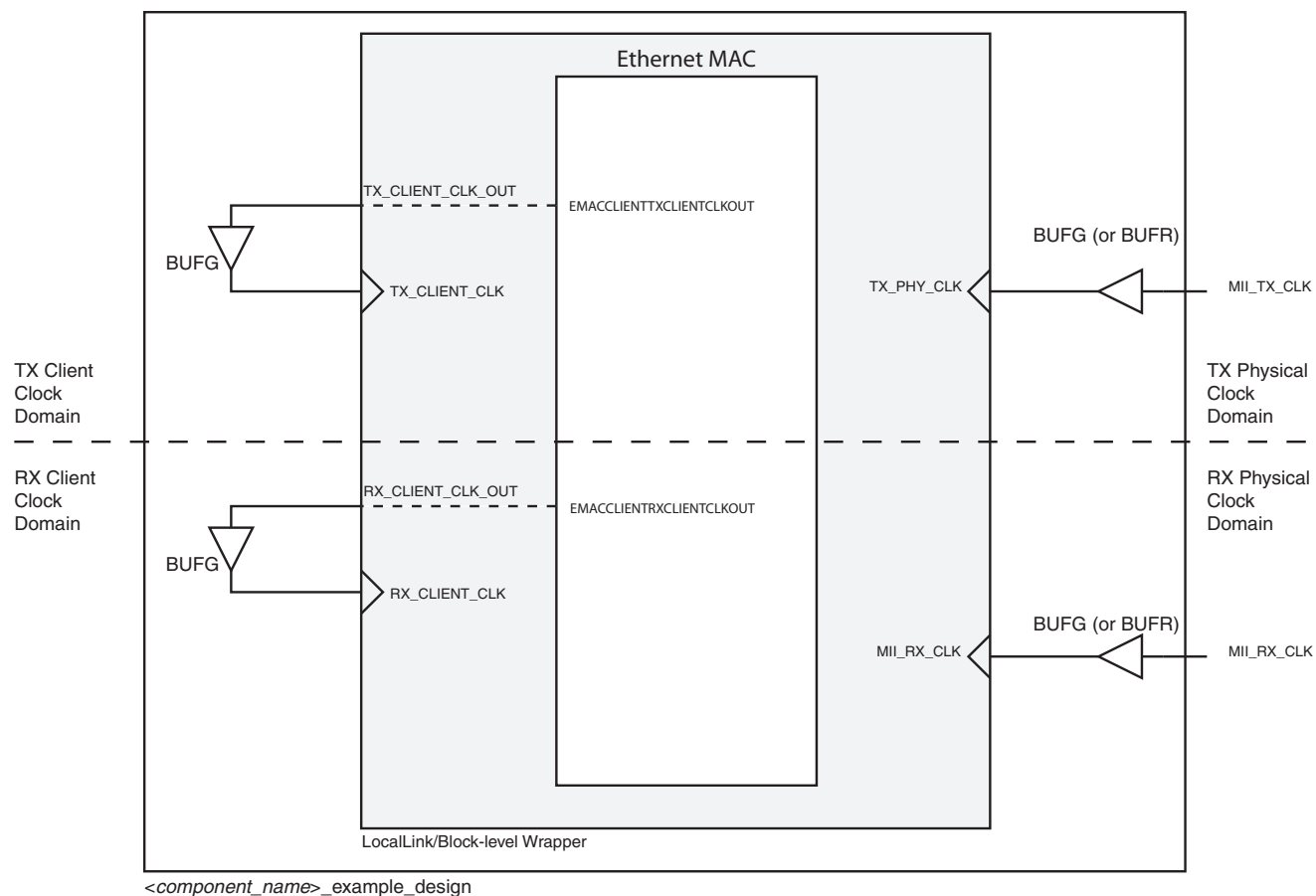


Figure B-6: RGMII Clocking at 10/100/1000 Mb/s



**Figure B-7: 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-8). 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 IODELAY 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 1000Mb/s.

**Caution!** Use of the BUFIO and BUFR regional clock buffers impose certain location requirements on PHY-side receiver data and clock pins, and the Embedded TEMAC instance. For more information on regional clocking for the Ethernet MAC, see Pinout Requirements in Appendix A in the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

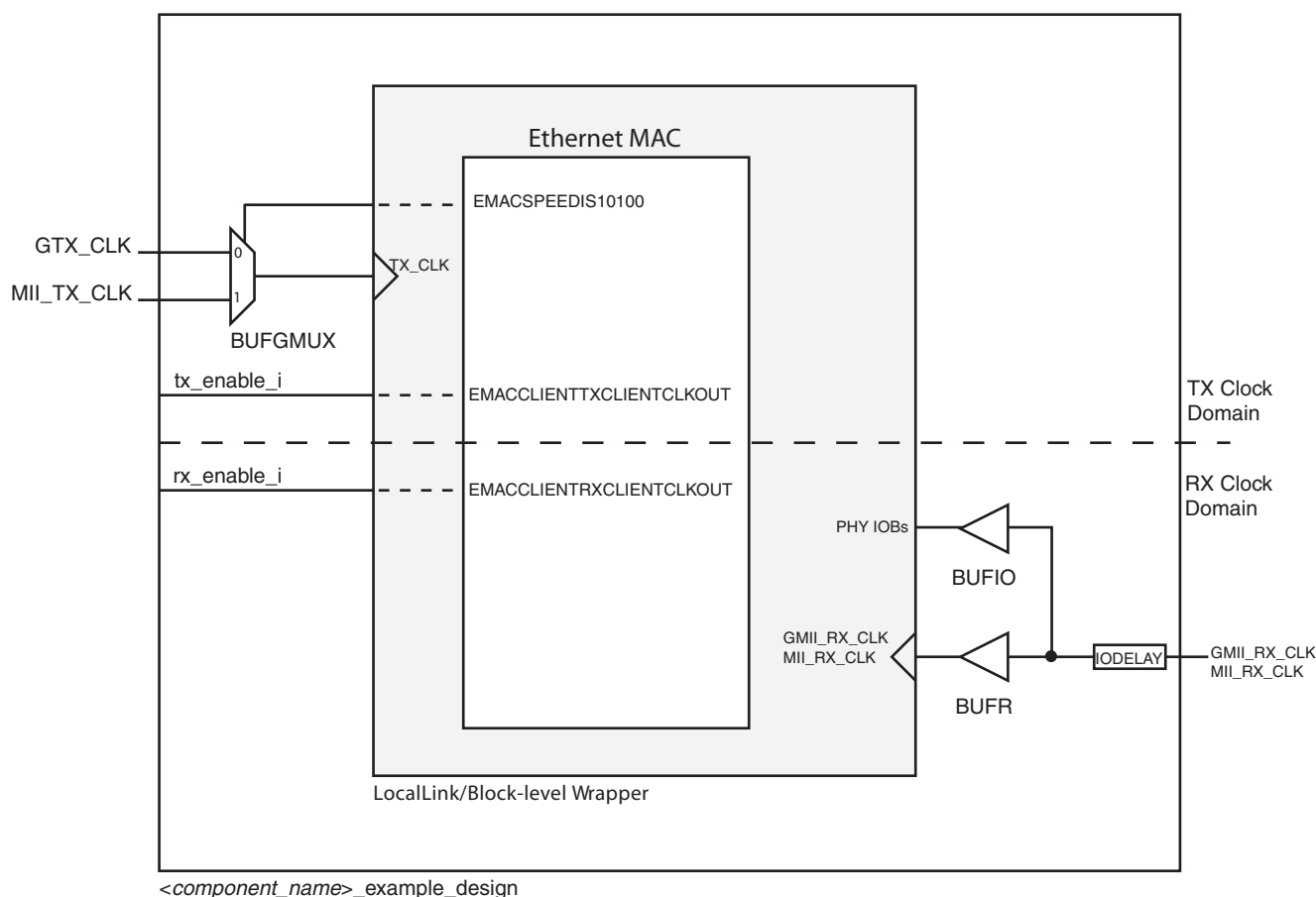


Figure B-8: GMII/MII Clocking at 10/100/1000 Mb/s with Clock Enables

## RGMII at Multiple Speeds with Clock Enable

Figure B-9 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 EMACPHYTXGMIIIMICLKOUT 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 IODELAY and a BUFIO-BUFR pair of clock buffers. The IODELAY 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.

**Caution!** Use of the BUFIO and BUFR regional clock buffers impose certain location requirements on PHY-side receiver data and clock pins, and the Embedded TEMAC instance. For more information on regional clocking for the Ethernet MAC, see Pinout Requirements in Appendix A in the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

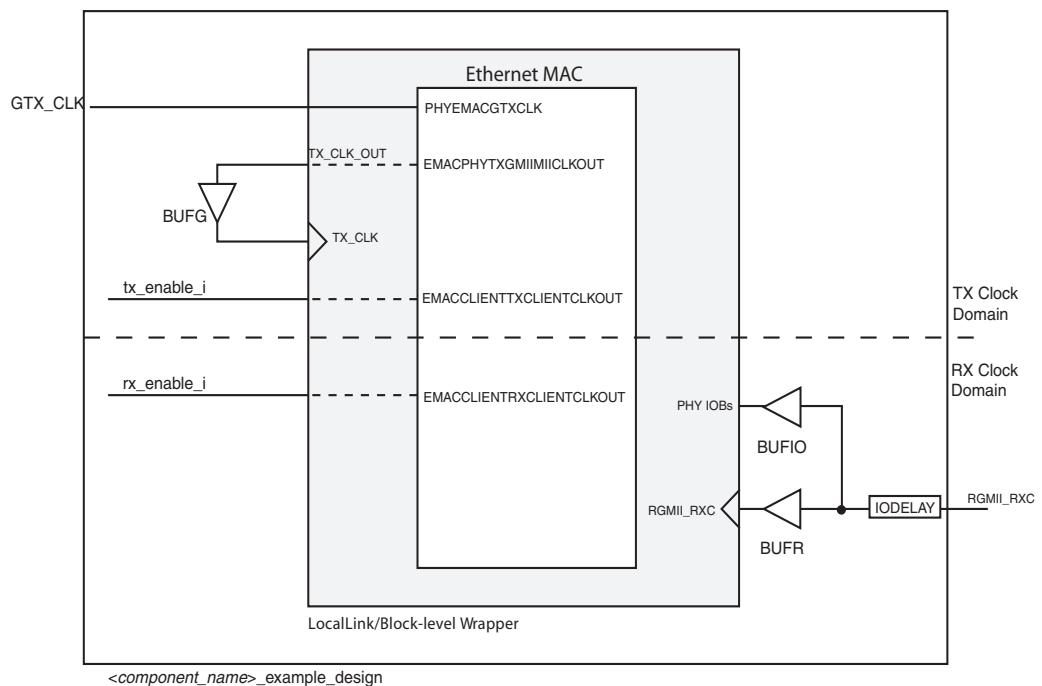


Figure B-9: RGMII Clocking at 10/100/1000 Mb/s with Clock Enable

## Constraining the Example Design

---

A UCF (<component\_name>\_example\_design.ucf) is provided with the example design, and includes the necessary constraints for proper Ethernet MAC operation. The file is separated into four sections: basic constraints, clock constraints, physical interface constraints, and LocalLink FIFO constraints.

The UCF details the application of the constraints for the specific hierarchical level of wrappers used.

### Basic Constraints

An example CONFIG PART constraint is provided for the Virtex®-6 FPGA family chosen within the CORE Generator™ tool. In this example, the 6VLX240TFF1156-1 part results from choosing a Virtex-6 LXT device.

```
# The 6vlx240tff1156-1 part is chosen for this example
# design.
# This value should be modified to match your device.
CONFIG PART = 6vlx240tff1156-1;
```

To facilitate placement, routing, and timing closure, an example Ethernet MAC instance location constraint is provided. Some configurations can include additional location and area group constraints.

```
# Locate the Tri-Mode Ethernet MAC instance
INST "v6_emac" LOC = "TEMAC_X0Y0";
```

### Clock Constraints

The clock constraints section of the UCF is tuned to the selected clocking and physical interface configuration.

#### Clock Constraints for MII Configurations

When the Clock Enable advanced clocking scheme is used, the transmit and receive PHY-side clocks must be constrained as if operation were at 1 Gb/s. These clocks are also used for client-side logic.

```
# Ethernet MII PHY-side transmit clock
NET "MII_TX_CLK" TNM_NET = "phy_clk_tx";
TIMEGRP "<component_name>_clk_phy_tx" = "phy_clk_tx";
TIMESPEC "TS_<component_name>_clk_phy_tx" = PERIOD
"<component_name>_clk_phy_tx" 8 ns HIGH 50 %;

# Ethernet MII PHY-side receive clock
```

```
NET "MII_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" 7.5 ns HIGH 50 %;
```

When the standard clocking methodology is used, both the transmit and receive, client- and PHY-side clocks are constrained for nominal 100 Mb/s operation. See the generated UCF for this scenario's full syntax.

## Clock Constraints for GMII Configurations

When configured for single-speed 1 Gb/s operation, the high-quality Ethernet reference clock is constrained to the specified 125 MHz, and the receive PHY-side clock is slightly over-constrained to handle the transient case. Additionally, the delay element reference clock is constrained to 200 MHz.

```
# Ethernet GTX_CLK high quality 125 MHz reference clock
NET "GTX_CLK" TNM_NET = "ref_gtx_clk";
TIMEGRP "<component_name>_clk_ref_gtx" = "ref_gtx_clk";
TIMESPEC "TS_<component_name>_clk_ref_gtx" = PERIOD
"<component_name>_clk_ref_gtx" 8 ns HIGH 50 %;

# Ethernet GMII PHY-side receive 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" 7.5 ns HIGH 50 %;

# IDELAYCTRL 200 MHz reference clock
NET "REFCLK" TNM_NET = "clk_ref_clk";
TIMEGRP "ref_clk" = "clk_ref_clk";
TIMESPEC "TS_ref_clk" = PERIOD "ref_clk" 5 ns HIGH 50 %;
```

When configured for tri-speed operation and the Clock Enable advanced clocking scheme is used, the high-quality Ethernet reference clock (and the output of the BUFGMUX it drives) are constrained to the specified 125 MHz. The receive, PHY-side clock is constrained for nominal 1 Gb/s operation, and is slightly over-constrained to handle the transient case. This clock is also used for client-side logic. Additionally, the delay element reference clock is constrained to 200 MHz.

```
# Ethernet GTX_CLK high quality 125 MHz reference clock
NET "GTX_CLK" TNM_NET = "ref_gtx_clk";
TIMEGRP "<component_name>_clk_ref_gtx" = "ref_gtx_clk";
TIMESPEC "TS_<component_name>_clk_ref_gtx" = PERIOD
"<component_name>_clk_ref_gtx" 8 ns HIGH 50 %;

# Multiplexed 1 Gb/s, 10/100 Mb/s output inherits constraint from
# GTX_CLK
NET "tx_clk" TNM_NET = "ref_mux_clk";
TIMEGRP "<component_name>_clk_ref_mux" = "ref_mux_clk";
TIMESPEC "TS_<component_name>_clk_ref_mux" = PERIOD
"<component_name>_clk_ref_mux" TS_<component_name>_clk_ref_gtx HIGH
50%;

# Ethernet GMII PHY-side receive 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" 7.5 ns HIGH 50 %;
```

```
# IDELAYCTRL 200 MHz reference clock
NET "REFCLK" TNM_NET = "clk_ref_clk";
TIMEGRP "ref_clk" = "clk_ref_clk";
TIMESPEC "TS_ref_clk" = PERIOD "ref_clk" 5 ns HIGH 50 %;
```

When configured for tri-speed operation and the standard clocking methodology is used, the high-quality Ethernet reference clock is constrained to the specified 125 MHz, and the PHY clock is constrained for nominal 100 Mb/s operation; their multiplexed output is constrained to the higher frequency. Both the transmit and receive, client- and PHY-side clocks are constrained for nominal 1 Gb/s operation, and the receive PHY-side clock is slightly over-constrained to handle the transient case. Additionally, the delay element reference clock is constrained to 200 MHz. See the generated UCF for this scenario's full syntax.

## Clock Constraints for RGMII Configurations

When configured for single-speed 1 Gb/s operation, the high-quality Ethernet reference clock is constrained to the specified 125 MHz, and the receive PHY-side clock is slightly over-constrained to handle the transient case. Additionally, the delay element reference clock is constrained to 200 MHz.

```
# Ethernet GTX_CLK high quality 125 MHz reference clock
NET "GTX_CLK" TNM_NET = "ref_gtx_clk";
TIMEGRP "<component_name>_clk_ref_gtx" = "ref_gtx_clk";
TIMESPEC "TS_<component_name>_clk_ref_gtx" = PERIOD
"<component_name>_clk_ref_gtx" 8 ns HIGH 50 %;

# Ethernet RGMII PHY-side receive clock
NET "RGMII_RXC" 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" 7.5 ns HIGH 50 %;

# IDELAYCTRL 200 MHz reference clock
NET "REFCLK" TNM_NET = "clk_ref_clk";
TIMEGRP "ref_clk" = "clk_ref_clk";
TIMESPEC "TS_ref_clk" = PERIOD "ref_clk" 5 ns HIGH 50 %;
```

When configured for tri-speed operation and the Clock Enable advanced clocking scheme is used, the high-quality Ethernet reference clock is constrained to the specified 125 MHz. The transmit and receive PHY-side clocks are constrained for nominal 1 Gb/s operation, and the receive PHY-side clock is slightly over-constrained to handle the transient case. These clocks are also used for client-side logic. Additionally, the delay element reference clock is constrained to 200 MHz.

```
# Ethernet GTX_CLK high quality 125 MHz reference clock
NET "GTX_CLK" TNM_NET = "ref_gtx_clk";
TIMEGRP "<component_name>_clk_ref_gtx" = "ref_gtx_clk";
TIMESPEC "TS_<component_name>_clk_ref_gtx" = PERIOD
"<component_name>_clk_ref_gtx" 8 ns HIGH 50 %;

# Ethernet RGMII PHY-side transmit clock
NET "tx_clk_o" TNM_NET = "phy_clk_tx";
TIMEGRP "<component_name>_clk_phy_tx" = "phy_clk_tx";
TIMESPEC "TS_<component_name>_clk_phy_tx" = PERIOD
"<component_name>_clk_phy_tx" 8 ns HIGH 50 %;
```

```
# Ethernet RGMII PHY-side receive clock
NET "RGMII_RXC" 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" 7.5 ns HIGH 50 %;

# IDELAYCTRL 200 MHz reference clock
NET "REFCLK" TNM_NET = "clk_ref_clk";
TIMEGRP "ref_clk" = "clk_ref_clk";
TIMESPEC "TS_ref_clk" = PERIOD "ref_clk" 5 ns HIGH 50 %;
```

When configured for tri-speed operation and the standard clocking methodology is used, the high-quality Ethernet reference clock is constrained to the specified 125 MHz. Both the transmit and receive, client- and PHY-side clocks are constrained for nominal 1 Gb/s operation, and the receive PHY-side clock is slightly over-constrained to handle the transient case. Additionally, the delay element reference clock is constrained to 200 MHz. See the generated UCF for this scenario's full syntax.

## Clock Constraints for 1000BASE-X PCS/PMA Configurations

The serial transceiver's TXOUTCLK is used as the high-quality Ethernet reference clock. When the 8-bit client interface is used, this clock is constrained to 125 MHz.

```
# Ethernet MAC reference clock driven by transceiver
NET "clk125_o" TNM_NET = "clk_gt_clk";
TIMEGRP "<component_name>_gt_clk" = "clk_gt_clk";
TIMESPEC "TS_<component_name>_gt_clk" = PERIOD
"<component_name>_gt_clk" 8 ns HIGH 50 %;
```

When the 16-bit client interface is used, an MMCM derives two outputs from the serial transceiver's TXOUTCLK: a clock for client-side logic, and a PHY-side (serial transceiver) clock. A period constraint is only applied to the input of the MMCM, because this constraint is automatically propagated to its outputs. Overclocking up to 2.5 Gb/s is supported in this case, and the constrained period depends on the selected rate. The 2.5 Gb/s case is shown in this example.

```
# Ethernet MAC reference clock driven by transceiver
NET "clk125_o" TNM_NET = "clk_gt_clk";
NET "clk125_pre_bufg" TNM_NET = "clk125_clk";
NET "clk2x_pre_bufg" TNM_NET = "clk2x_clk";
TIMEGRP "<component_name>_gt_clk" = "clk125_clk" "clk2x_clk";
# In 2.5 Gb/s overclocking mode, note that clk125_o
# operates at 156.25 MHz
TIMESPEC "TS_<component_name>_gt_clk" = PERIOD "clk_gt_clk" 6.4 ns HIGH
50 %;
```

## Clock Constraints for SGMII Configurations

The serial transceiver's TXOUTCLK is used as the high-quality Ethernet reference clock, and is constrained to 125 MHz. When configured for tri-speed operation, the client-side clock is constrained for nominal 1 Gb/s operation.

```
# Ethernet MAC reference clock driven by transceiver
NET "clk125_o" TNM_NET = "clk_gt_clk";
TIMEGRP "<component_name>_gt_clk" = "clk_gt_clk";
TIMESPEC "TS_<component_name>_gt_clk" = PERIOD
"<component_name>_gt_clk" 8 ns HIGH 50 %;
```

```
# Tri-speed client clock from Ethernet MAC
NET "client_clk_o" TNM_NET = "clk_client";
TIMEGRP "<component_name>_gt_clk_client" = "clk_client";
TIMESPEC "TS_<component_name>_gt_clk_client" = PERIOD
"<component_name>_gt_clk_client" 8 ns HIGH 50 %;
```

When configured for single-speed 1 Gb/s operation, only the high-quality Ethernet reference clock is constrained. This is identical to the 8-bit client, 1000BASE-X case.

## Clock Constraints for the Host Interface

When the host or DCR interface is selected, an example constraint is provided which constrains HOSTCLK or DCREMACCLK, respectively, to 100 MHz. The host interface case is shown here.

```
# Constrain the host interface clock to an example
# frequency of 100 MHz
NET "HOSTCLK" TNM_NET = "host_clock";
TIMEGRP "clk_host" = "host_clock";
TIMESPEC "TS_clk_host" = PERIOD "clk_host" 10 ns HIGH 50 %;
```

## Physical Interface Constraints

The physical interface constraints section of the UCF is tuned to the selected physical interface. The location constraints shown here should be considered an example only, and can differ with device family. See the generated UCF for the appropriate location constraints for your target device.

### Physical Interface Constraints for MII Configurations

Constraints are provided which locate the physical interface flip-flops in IOBs, check whether the appropriate receive MII bus setup and hold times are present at the input to the FPGA, and provide example pin locations.

```
# Constrain the MII physical interface flip-flops to IOBs
INST "*mii?RXD_TO_MAC*" IOB = true;
INST "*mii?RX_DV_TO_MAC" IOB = true;
INST "*mii?RX_ER_TO_MAC" IOB = true;
INST "*mii?MII_TXD_?" IOB = true;
INST "*mii?MII_TX_EN" IOB = true;
INST "*mii?MII_TX_ER" IOB = true;

# Constrain MII inputs for 10 ns setup time and 10 ns hold
# time with respect to MII_RX_CLK
INST "MII_RXD<?>" TNM = "mii_rx";
INST "MII_RX_DV" TNM = "mii_rx";
INST "MII_RX_ER" TNM = "mii_rx";
TIMEGRP "mii_rx" OFFSET = IN 10 ns VALID 20 ns BEFORE "MII_RX_CLK"
RISING;

# Location constraints are chosen for this example design.
# These values should be modified to suit your design.

# Locate the MII physical interface pins
INST "MII_COL" LOC = "BANK33";
INST "MII_CRS" LOC = "BANK33";
INST "MII_TXD<?>" LOC = "BANK33";
```

```

INST "MII_TX_EN" LOC = "BANK33";
INST "MII_TX_ER" LOC = "BANK33";
INST "MII_TX_CLK" LOC = "BANK33";
INST "MII_RXD<?>" LOC = "BANK33";
INST "MII_RX_DV" LOC = "BANK33";
INST "MII_RX_ER" LOC = "BANK33";
INST "MII_RX_CLK" LOC = "AP11";

```

## Physical Interface Constraints for GMII Configurations

Constraints are provided which align the receiver bus to the receiver clock and check whether the appropriate setup time results from these adjustments. More detail on these constraints immediately follows.

```

# Set the IDELAY values on the PHY inputs, tuned for this
# example design. These values should be modified to suit
# your design.
INST "*gmii?ideladv" IDELAY_VALUE = 28;
INST "*gmii?ideld0" IDELAY_VALUE = 28;
INST "*gmii?ideld1" IDELAY_VALUE = 28;
INST "*gmii?ideld2" IDELAY_VALUE = 28;
INST "*gmii?ideld3" IDELAY_VALUE = 28;
INST "*gmii?ideld4" IDELAY_VALUE = 28;
INST "*gmii?ideld5" IDELAY_VALUE = 28;
INST "*gmii?ideld6" IDELAY_VALUE = 28;
INST "*gmii?ideld7" IDELAY_VALUE = 28;
INST "*gmii?ideler" IDELAY_VALUE = 28;
INST "*gmii_rxc_delay" IDELAY_VALUE = 0;
INST "*gmii_rxc_delay" SIGNAL_PATTERN = CLOCK;

# Group all IDELAY-related blocks to use a single
# IDELAYCTRL
INST "*dlyctrl" IODELAY_GROUP = gmii_idelay;
INST "*ideld?" IODELAY_GROUP = gmii_idelay;
INST "*ideldv" IODELAY_GROUP = gmii_idelay;
INST "*ideler" IODELAY_GROUP = gmii_idelay;
INST "*gmii_rxc_delay" IODELAY_GROUP = gmii_idelay;

# The following constraints work in conjunction with
# IDELAY_VALUE settings to check that the GMII receive
# bus remains in alignment with the rising edge of
# GMII_RX_CLK, to within 2ns setup time and 750ps
# hold time. In addition to adjusting IDELAY_VALUE
# settings for your system's timing characteristics, you
# may wish to refine these constraints to match the
# GMII specification; see Answer Record 33195 on
# xilinx.com for details.
INST "GMII_RXD<?>" TNM = "gmii_rx";
INST "GMII_RX_DV" TNM = "gmii_rx";
INST "GMII_RX_ER" TNM = "gmii_rx";
TIMEGRP "gmii_rx" OFFSET = IN 2 ns VALID 2.75 ns BEFORE "GMII_RX_CLK"
RISING;

```

Constraints are also provided to locate the physical interface flip-flops in IOBs, and for example pin and clock buffer resource locations.

```

# Constrain the GMII physical interface flip-flops to IOBs
INST "*gmii?RXD_TO_MAC*" IOB = true;
INST "*gmii?RX_DV_TO_MAC" IOB = true;

```



```

INST "*gmii?RX_ER_TO_MAC" IOB = true;
INST "*gmii?GMII_TXD_?" IOB = true;
INST "*gmii?GMII_TX_EN" IOB = true;
INST "*gmii?GMII_TX_ER" IOB = true;

# Location constraints are chosen for this example design.
# These values should be modified to suit your design.
# * Note that regional clocking imposes certain
# requirements on the location of the physical interface
# pins and the TEMAC instance. See the
# Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide
# for additional details. *

# Locate the GMII physical interface pins
INST "GMII_TXD<?>" LOC = "BANK33";
INST "GMII_TX_EN" LOC = "BANK33";
INST "GMII_TX_ER" LOC = "BANK33";
INST "GMII_TX_CLK" LOC = "BANK33";
INST "GMII_RXD<?>" LOC = "BANK33";
INST "GMII_RX_DV" LOC = "BANK33";
INST "GMII_RX_ER" LOC = "BANK33";
INST "GMII_RX_CLK" LOC = "AP11";

# Locate the 125 MHz reference clock buffer
INST "bufg_tx" LOC = "BUFGCTRL_X0Y6";

# Locate the 200 MHz delay controller clock buffer
INST "refclk_bufg" LOC = "BUFGCTRL_X0Y7";

```

## GMII IDELAY\_VALUE Constraints

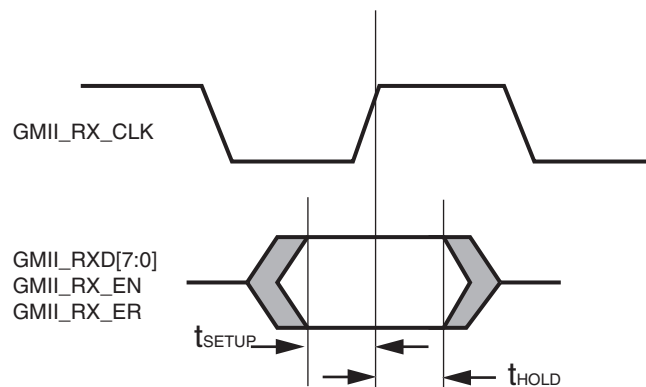


Figure C-1: Input GMII Timing

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
$t_{\text{SETUP}}$	2.00	-	ns
$t_{\text{HOLD}}$	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 IODELAY 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 IDELAY\_VALUE settings delivered in the example design UCF are intended as an example only. They must be modified to accommodate each design's specific timing details. Xilinx does not recommend that a single tap value is effective across all hardware families. This OFFSET constraint syntax provided in the example design UCF causes the Xilinx® implementation tools to analyze the input setup and hold constraints for the input GMII bus. If these constraints are not met, the tools will report timing errors. However, the tools do not attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the IDELAY\_VALUE settings in the UCF.

## Physical Interface Constraints for RGMII Configurations

Constraints are provided which align the receiver bus to the rising and falling edges of the receiver clock, and check whether the appropriate setup and hold times result from these adjustments. More detail on these constraints immediately follows.

```
# Set the IDELAY and ODELAY values, tuned for this example
# design. These values should be modified to suit your
# design.
INST "*rgmii?rgmii_rx_ctl_delay" IDELAY_VALUE = 20;
INST "*rgmii?rgmii_rx_d0_delay" IDELAY_VALUE = 20;
INST "*rgmii?rgmii_rx_d1_delay" IDELAY_VALUE = 20;
INST "*rgmii?rgmii_rx_d2_delay" IDELAY_VALUE = 20;
INST "*rgmii?rgmii_rx_d3_delay" IDELAY_VALUE = 20;
INST "*rgmii_rxc_delay" IDELAY_VALUE = 0;
INST "*rgmii_rxc_delay" SIGNAL_PATTERN = CLOCK;
INST "*rgmii?rgmii_tx_clk_delay" ODELAY_VALUE = 6;
INST "*rgmii?rgmii_tx_clk_delay" SIGNAL_PATTERN = CLOCK;

# Group all IODELAY-related blocks to use a single
# IDELAYCTRL
INST "*dlyctrl"
IODELAY_GROUP = rgmii_v2_0_iodelay;
INST "*rgmii_rx_ctl_delay"
IODELAY_GROUP = rgmii_v2_0_iodelay;
INST "*rgmii_rx_d?_delay"
IODELAY_GROUP = rgmii_v2_0_iodelay;
INST "*rgmii_rxc_delay"
IODELAY_GROUP = rgmii_v2_0_iodelay;
INST "*rgmii_tx_clk_delay"
IODELAY_GROUP = rgmii_v2_0_iodelay;

# The following constraints work in conjunction with
# IDELAY_VALUE settings to check that the RGMII
# receive bus remains in alignment with the rising and
# falling edges of RGMII_RXC, to within 1.35ns setup
# time and 1.35ns hold time. In addition to adjusting
# IDELAY_VALUE settings for your system's timing
# characteristics, you may wish to refine these
# constraints to match the RGMII specification; see
# Answer Record 33195 on xilinx.com for details.
INST "RGMII_RXD<?>" TNM = "rgmii_rx";
```

```

INST "RGMIIRX_CTL" TNM = "rgmii_rx";
TIMEGRP "rgmii_rx" OFFSET = IN 1.35 ns VALID 2.7 ns BEFORE "RGMIIRXC"
RISING;
TIMEGRP "rgmii_rx" OFFSET = IN 1.35 ns VALID 2.7 ns BEFORE "RGMIIRXC"
FALLING;

```

If RGMII v2.0 is selected, an OFFSET OUT constraint is provided in 1 Gb/s configurations to report the skew between output data bits, relative to the transmitted clock. This can be helpful in tuning ODELAY\_VALUES to meet PHY setup and hold time requirements.

```

# Constrain the RGMII transmitter signals to provide edge skew analysis
# in the timing report. This can be used to tune the ODELAY_VALUE for
# your system.
INST "RGMIITXD<?>" TNM = "rgmii_tx";
INST "RGMIITX_CTL" TNM = "rgmii_tx";
INST "RGMIITXC" TNM = "rgmii_tx";
TIMEGRP "rgmii_tx" OFFSET = OUT AFTER "GTX_CLK" REFERENCE_PIN
"RGMIITXC" RISING;
TIMEGRP "rgmii_tx" OFFSET = OUT AFTER "GTX_CLK" REFERENCE_PIN
"RGMIITXC" FALLING;

```

Example pin and clock resource location constraints are provided. If RGMII v2.0 is selected, the 1.5V HSTL\_I logic standard is also specified for the physical interface I/Os.

```

# Location constraints are chosen for this example design.
# These values should be modified to suit your design.
# * Note that regional clocking imposes certain
# requirements on the location of the physical interface
# pins and the TEMAC instance. Refer to the
# Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide
# for additional details. *

# Locate the RGMII physical interface pins and IDELAYCTRL
# element
INST "RGMIITXD<?>" LOC = "BANK33";
INST "RGMIITX_CTL" LOC = "BANK33";
INST "RGMIITXC" LOC = "BANK33";
INST "RGMIIRXD<?>" LOC = "BANK33";
INST "RGMIIRX_CTL" LOC = "BANK33";
INST "RGMIIRXC" LOC = "AP11";

# RGMII v2.0 logic standards
INST "RGMIITXD<?>" IOSTANDARD = HSTL_I;
INST "RGMIITX_CTL" IOSTANDARD = HSTL_I;
INST "RGMIITXC" IOSTANDARD = HSTL_I;
INST "RGMIIRXD<?>" IOSTANDARD = HSTL_I;
INST "RGMIIRX_CTL" IOSTANDARD = HSTL_I;
INST "RGMIIRXC" IOSTANDARD = HSTL_I;

# Locate the 125 MHz reference clock
INST "GTX_CLK" IOSTANDARD = HSTL_I;
INST "GTX_CLK" LOC = "BANK33";

# Locate the 200 MHz delay controller clock
INST "REFCLK" IOSTANDARD = HSTL_I;
INST "REFCLK" LOC = "BANK33";
INST "refclk_bufg" LOC = "BUFGCTRL_X0Y7";

```

## RGMII IDELAY\_VALUE Constraints

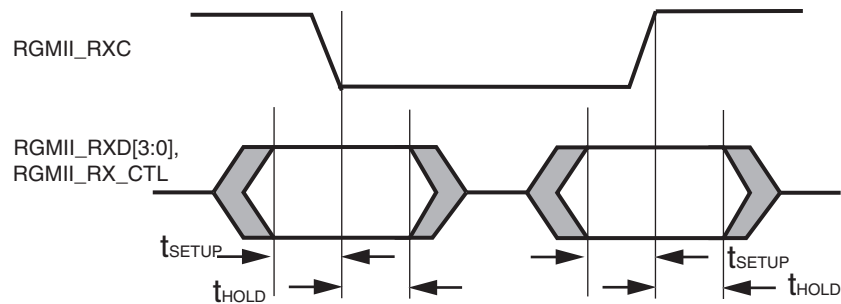


Figure C-2: RGMII Input Timing

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
$t_{\text{SETUP}}$	1.00	-	ns
$t_{\text{HOLD}}$	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 IDELAY\_VALUE settings delivered in the example design UCF are intended as an example only. They must be modified to accommodate each design's specific timing details. Xilinx does not recommend that a single tap value is effective across all hardware families. This OFFSET constraint syntax provided in the example design UCF causes the Xilinx implementation tools to analyze the input setup and hold constraints for the input RGMII bus. If these constraints are not met, the tools will report timing errors. However, the tools do not attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the IDELAY\_VALUE settings in the UCF.

## Physical Interface Constraints for 1000BASE-X PCS/PMA Configurations

Example location constraints are provided for the serial transceiver and its differential reference clock pins.

```
# Place the transceiver components, chosen for this example
# design. These values should be modified according to your
# specific design.
INST "MGTCLK_N" LOC = "V5";
INST "MGTCLK_P" LOC = "V6";
INST "TXP"      LOC = "Y1";
INST "TXN"      LOC = "Y2";
INST "RXP"      LOC = "AA3";
INST "RXN"      LOC = "AA4";
INST "*gtx0_v6_gtxwizard_i?gtxe1_i" LOC = "GTXE1_X0Y8";
```

## Physical Interface Constraints for SGMII Configurations

Example location constraints are provided for the serial transceiver and its differential reference clock pins. If the SGMII fabric elastic buffer is used, its necessary timing constraints are also included.

```
# Place the transceiver components, chosen for this example
# design. These values should be modified according to your
# specific design.
INST "MGTCLK_N" LOC = "V5";
INST "MGTCLK_P" LOC = "V6";
INST "TXP"      LOC = "Y1";
INST "TXN"      LOC = "Y2";
INST "RXP"      LOC = "AA3";
INST "RXN"      LOC = "AA4";
INST "*gtx0_v6_gtxwizard_i?gtxe1_i" LOC = "GTXE1_X0Y8";

#####
# SGMII FABRIC RX ELASTIC BUFFER TIMING CONSTRAINTS
# The following constraints are necessary for proper
# operation of the SGMII fabric Rx elastic buffer.
#####

# Constrain the recovered clock frequency to 125 MHz
NET "*v6_gtxwizard_top_inst?RXRECCLK" TNM_NET = "clk_rec_clk";
TIMEGRP "<component_name>_client_rec_clk" = "clk_rec_clk";
TIMESPEC "TS_<component_name>_client_rec_clk" = PERIOD
"<component_name>_client_rec_clk" 8 ns HIGH 50 %;

# Control gray code delay and skew
INST "*v6_gtxwizard_top_inst?rx_elastic_buffer_inst?rd_addr_gray?"
TNM = "rx_elastic_rd_to_wr";
TIMESPEC "TS_rx_elastic_rd_to_wr" = FROM "rx_elastic_rd_to_wr" TO
"clk_rec_clk" 7.5 ns DATAPATHONLY;
INST "*v6_gtxwizard_top_inst?rx_elastic_buffer_inst?wr_addr_gray?"
TNM = "elastic_metastable";
TIMESPEC "TS_elastic_meta_protect" = FROM "elastic_metastable" 5 ns
DATAPATHONLY;

# Reduce clock period to allow for metastability settling
# time
INST "*v6_gtxwizard_top_inst?rx_elastic_buffer_inst?rd_wr_addr_gray*"
TNM = "rx_graycode";
INST "*v6_gtxwizard_top_inst?rx_elastic_buffer_inst?rd_occupancy*"
TNM = "rx_binary";
TIMESPEC "TS_rx_buf_meta_protect" = FROM "rx_graycode" TO "rx_binary"
5 ns;
```

## LocalLink FIFO Constraints

The LocalLink FIFO section of the UCF includes FROM-TO constraints to facilitate clock domain crossing and reduce the probability of metastable events.

```
# LocalLink client FIFO transmit-side constraints
# -----

# Group the clock crossing signals into timing groups
INST "*client_side_FIFO?tx_fifo_i?rd_tran_frame_tog"
TNM = "tx_fifo_rd_to_wr";
INST "*client_side_FIFO?tx_fifo_i?rd_retran_frame_tog"
```

```

TNM = "tx_fifo_rd_to_wr";
INST "*client_side_FIFO?tx_fifo_i?rd_col_window_pipe_1"
TNM = "tx_fifo_rd_to_wr";
INST "*client_side_FIFO?tx_fifo_i?rd_addr_txfer*"
TNM = "tx_fifo_rd_to_wr";
INST "*client_side_FIFO?tx_fifo_i?rd_txfer_tog"
TNM = "tx_fifo_rd_to_wr";
INST "*client_side_FIFO?tx_fifo_i?wr_frame_in_fifo"
TNM = "tx_fifo_wr_to_rd";

TIMESPEC "TS_tx_fifo_rd_to_wr" = FROM "tx_fifo_rd_to_wr"
TO "<component_name>_gt_clk_client" 8 ns DATAPATHONLY;
TIMESPEC "TS_tx_fifo_wr_to_rd" = FROM "tx_fifo_wr_to_rd"
TO "<component_name>_gt_clk_client" 8 ns DATAPATHONLY;

# Reduce clock period to allow for metastability settling time
INST "*client_side_FIFO?tx_fifo_i?wr_tran_frame_tog"
TNM = "tx_metastable";
INST "*client_side_FIFO?tx_fifo_i?wr_rd_addr*"
TNM = "tx_metastable";
INST "*client_side_FIFO?tx_fifo_i?wr_txfer_tog"
TNM = "tx_metastable";
INST "*client_side_FIFO?tx_fifo_i?frame_in_fifo"
TNM = "tx_metastable";
INST "*client_side_FIFO?tx_fifo_i?wr_retran_frame_tog*"
TNM = "tx_metastable";
INST "*client_side_FIFO?tx_fifo_i?wr_col_window_pipe_0"
TNM = "tx_metastable";
TIMESPEC "TS_tx_meta_protect" = FROM "tx_metastable" 5 ns DATAPATHONLY;

# Transmit-side client FIFO address bus timing
INST "*client_side_FIFO?tx_fifo_i?rd_addr_txfer*"
TNM = "tx_addr_rd";
INST "*client_side_FIFO?tx_fifo_i?wr_rd_addr*"
TNM = "tx_addr_wr";
TIMESPEC "TS_tx_fifo_addr" = FROM "tx_addr_rd" TO "tx_addr_wr" 10 ns;

# LocalLink client FIFO receive-side constraints
# -----

# Group the clock crossing signals into timing groups
INST "*client_side_FIFO?rx_fifo_i?wr_store_frame_tog"
TNM = "rx_fifo_wr_to_rd";
INST "*client_side_FIFO?rx_fifo_i?rd_addr_gray*"
TNM = "rx_fifo_rd_to_wr";

TIMESPEC "TS_rx_fifo_wr_to_rd" = FROM "rx_fifo_wr_to_rd"
TO "<component_name>_gt_clk_client" 8 ns DATAPATHONLY;
TIMESPEC "TS_rx_fifo_rd_to_wr" = FROM "rx_fifo_rd_to_wr"
TO "<component_name>_gt_clk_client" 8 ns DATAPATHONLY;

# Reduce clock period to allow for metastability settling time
INST "*client_side_FIFO?rx_fifo_i?wr_rd_addr_gray_sync*"
TNM = "rx_metastable";
INST "*client_side_FIFO?rx_fifo_i?rd_store_frame_tog"
TNM = "rx_metastable";
TIMESPEC "TS_rx_meta_protect" = FROM "rx_metastable" 5 ns;

```

# SGMII Capabilities

---

## SGMII Receiver Elastic Buffer

The Ethernet MAC wrapper GUI provides two SGMII Receiver Elastic Buffer options:

- **10/100/1000 Mb/s (clock tolerance compliant with Ethernet specification).** 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 (restricted tolerance for clocks) OR 100/1000 Mb/s.** Uses the receiver elastic buffer present in the serial 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 serial 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-6 FPGA GTX Transceiver User Guide*).

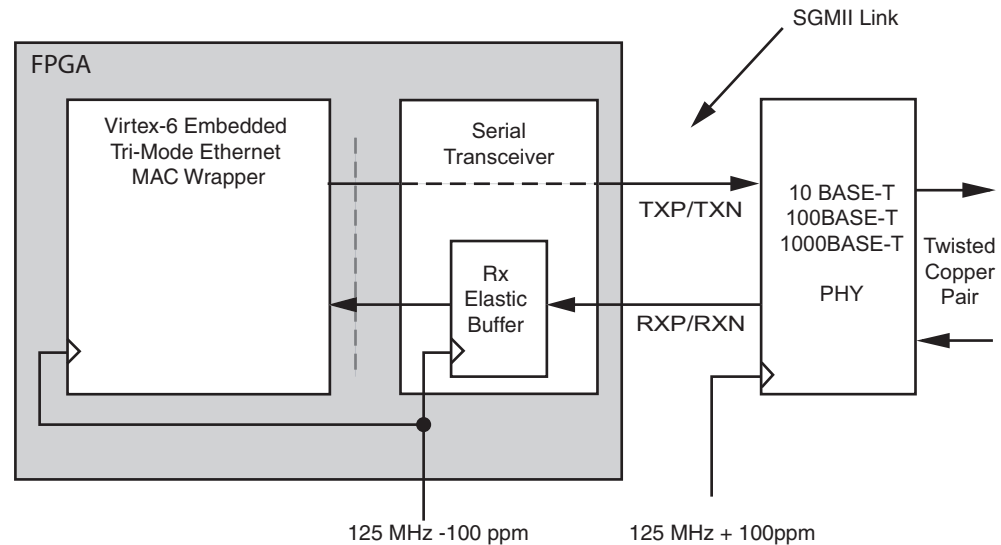


Figure D-1: SGMII Implementation: Separate Clock Sources

## 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 does 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 serial 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 *Virtex-6 FPGA GTX Transceiver User Guide*.



To guarantee reliable SGMII operation at 10 Mb/s (non-jumbo frames), the serial transceiver 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 Serial Transceiver Rx Elastic Buffer

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

- When 10 Mb/s operation is not required. Both 1 Gb/s and 100 Mb/s operation are guaranteed.
- When the clocks are closely related (see the following 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.

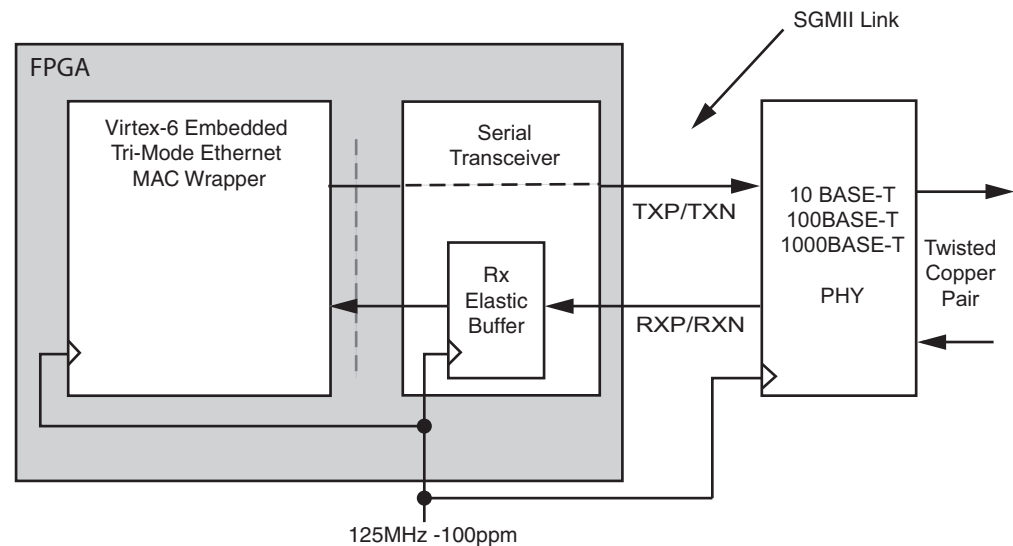


Figure D-2: SGMII Implementation: Shared Clock Sources

If the PHY device sources the receiver SGMII stream synchronously from the shared oscillator (check PHY data sheet), then the serial 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 serial 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 serial transceiver 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 *IEEE Std 802.3-2005* 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

## SGMII / 1000BASE-X PCS/PMA Mode Switching

When the Ethernet MAC wrappers are configured for tri-speed operation with the SGMII physical interface, it is possible to switch between SGMII and 1000BASE-X PCS/PMA modes of operation without reconfiguring the FPGA. This includes both transitions from SGMII to 1000BASE-X, and from 1000BASE-X to SGMII.

This is useful when using off-the-shelf PHY devices with the ability to perform both BASE-X and BASE-T standards, or when a single Virtex<sup>®</sup>-6 FPGA bitstream can be used with either a BASE-X or BASE-T PHY device.

Mode switching is performed by writing to the host interface and is explained in detail in the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

However, the Ethernet MAC wrappers can automate the process if the optional SGMII / 1000BASE-X mode switching block is included. This option is available in the **SGMII Capabilities** section of the CORE Generator™ software when all three of the following selections are made:

- An SGMII physical interface
- Tri-speed operation
- Either the host or DCR interface

## Switching Modes and Speeds

The Ethernet MAC wrappers execute the necessary mode switching commands based on a single signal, `BASE_X_ENABLE`, present as an input to the Block- and LocalLink-level wrappers, and to the top-level example design. When `BASE_X_ENABLE` is driven to logic '0', the Ethernet MAC operates according to the SGMII standard. When `BASE_X_ENABLE` is driven to logic '1', the Ethernet MAC operates according to the 1000BASE-X standard.

When operating in SGMII mode (`BASE_X_ENABLE` = 0), switching between 10, 100, and 1000 Mb/s is supported as usual. When operating in 1000BASE-X PCS/PMA mode (`BASE_X_ENABLE` = 1), only 1000 Mb/s is supported. When switching from SGMII at 10/100 Mb/s to 1000BASE-X PCS/PMA, the speed is automatically changed to 1000 Mb/s.

The `BASE_X_ENABLE` signal can be driven by logic internal to the FPGA, or by an external resource, such as a switch or jumper.

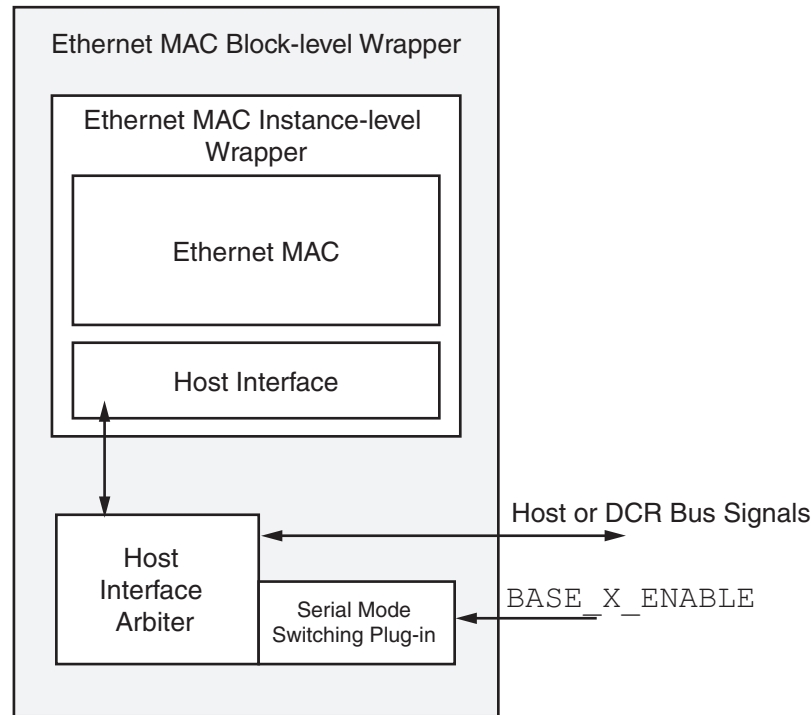
## Operational Requirements

### Dynamic Switching

The Ethernet MAC wrapper must be held in reset when the value of `BASE_X_ENABLE` is changed. Dynamic switching during Ethernet MAC operation is not supported and can result in unknown behavior.

### Host Interface Arbitration

The `BASE_X_ENABLE` signal is translated to the appropriate host interface commands by the serial mode switching plug-in block. The host interface arbiter manages the multiplexing of mode switching commands and standard user-initiated host or DCR commands. The `BASE_X_ENABLE` signal must be driven during Ethernet MAC operation; a floating input can result in unknown behavior.



**Figure D-3: Host Interface Arbiter and Serial Mode Switching Plug-in**

As a result of this multiplexing scheme, the host or DCR bus cannot be used for a short period of time following a change to `BASE_X_ENABLE`:

- If `BASE_X_ENABLE` is logic '0' following deassertion of reset to the Ethernet MAC wrappers, the host or DCR bus can be used immediately.
- If `BASE_X_ENABLE` is logic '1' following deassertion of reset to the Ethernet MAC wrappers, the host or DCR bus cannot be used for an additional 16 `HOSTCLK` cycles beyond when the serial transceiver asserts both of its `RESETDONE` signals.

Runtime modification of `BASE_X_ENABLE` without the assertion of reset to the Ethernet MAC wrappers is not allowed.

## Auto-Negotiation

In addition to changing the physical interface mode of operation, the serial mode switching plug-in sets the Auto-Negotiation link timer to the appropriate value for the new standard in use. When in SGMII mode of operation, the link timer value specified in the GUI is used. When in 1000BASE-X PCS/PMA mode of operation, the hexadecimal value `0x13D` is used, corresponding to approximately 10 ms.

However, Auto-Negotiation is not automatically restarted following a mode change. The user is responsible for programming the PCS/PMA registers as appropriate and restarting Auto-Negotiation.

For more information on SGMII/1000BASE-X PCS/PMA mode switching, including a more in-depth details on Auto-Negotiation requirements, see "Switching Between SGMII and 1000BASE-X Standards" in the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

# *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®-6 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](http://www.xilinx.com/tools/cspro.htm).

## Available Reference Boards

The ML605 is a Xilinx<sup>®</sup> development board supporting 10/100/1000 Mb/s Ethernet. The ML605 board can be used to prototype designs and establish that the core can communicate with the system.

Xilinx application note 1144 describes a Virtex-6 Embedded Tri-Mode Ethernet MAC Hardware Demonstration Platform based on the Ethernet MAC wrapper and targeted to the ML605. See [www.xilinx.com/support/documentation](http://www.xilinx.com/support/documentation) for XAPP 1144 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.

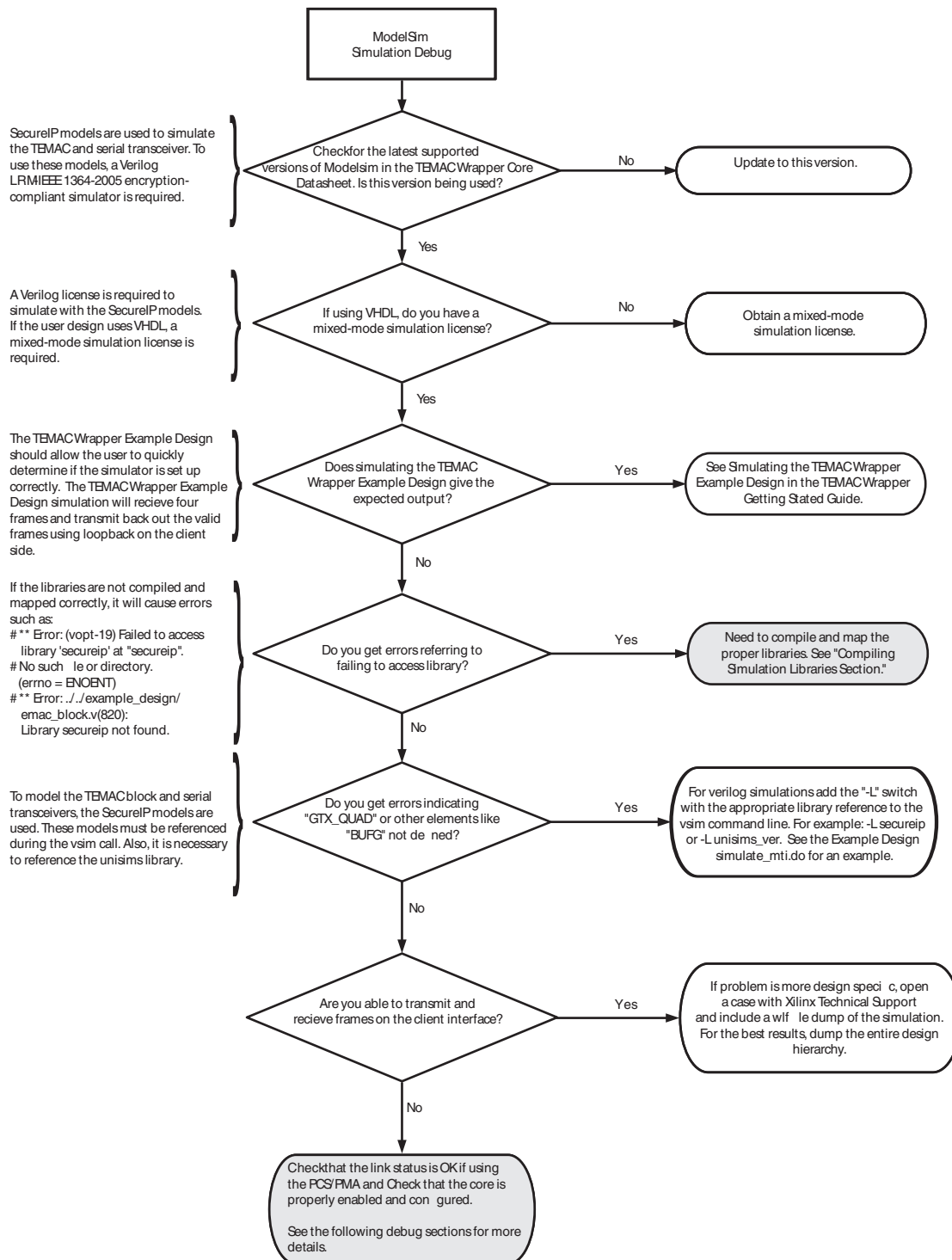


Figure E-1: Simulation Debug Flow Chart

## 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 virtex6 -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-6 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-6 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.

### Regional Clocking Errors in Map

When implementing the Ethernet MAC with either a GMII or RGMII physical interface, regional clocking methodologies are used. This means that there are the following requirements:

1. The receive-side physical interface clock (GMII\_RX\_CLK for GMII, or RGMII\_RXC or RGMII) must be placed at a clock-capable I/O (CCIO) pin. If this requirement is not met, an error similar to the following one will be seen during implementation:

*ERROR:Place:839 - The component GMII\_RX\_CLK has been physically constrained to a location which is an invalid placement for this component.*

2. All receive-side physical interface signals must be placed at package pins that correspond to the same clock region as the receive-side physical interface. If this requirement is not met, an error similar to the following one will be seen during implementation:

*ERROR:Place:901 - IO Clock Net "gmii\_rx\_clk\_bufio" cannot possibly be routed to component v6\_emac\_gmii\_locallink\_inst/v6\_emac\_gmii\_block\_inst/gmii/RXD\_TO\_MAC<2>" (placed in clock region "CLOCKREGION\_X0Y1"), because it is too far away from source BUFIO "bufio\_rx" (placed in clock region "CLOCKREGION\_X1Y1"). The situation might be caused by user constraints, or the complexity of the design. Constraining the components related to the regional clock properly might guide the tool to find a solution.*

3. An available Embedded Tri-Mode Ethernet MAC block must be present in a clock region that is reachable by the regionally buffered physical interface clock net. If this requirement is not met, an error similar to the following one will be seen during implementation:

*ERROR:Place:905 - Components driven by Regional clock net <rx\_clk\_i> cannot be placed and routed because location constraints are causing the clock region rules to be violated. Regional Clock net <rx\_clk\_i> is being driven by BUFR <bufr\_rx> locked to site "BUFR\_X0Y10" Because of this location constraint, <rx\_clk\_i> can only drive clock regions "CLOCKREGION\_X0Y5, CLOCKREGION\_X0Y4".*

*The following components driven by <rx\_clk\_i> have been locked to sites outside of these clock regions:*

*v6\_emac\_gmii\_locallink\_inst/v6\_emac\_gmii\_block\_inst/v6\_emac\_gmii\_inst/v6\_emac*  
(Locked Site: TEMAC\_X0Y0 CLOCKREGION\_X1Y2)

For more information on these requirements, see the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*, Appendix A, Pinout Requirements.

## 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, it might be required to adjust the fixed DELAY value 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 MMCMs in the design, ensure that all MMCMs 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 will be held in reset if the main reset input is asserted or if CLIENTTEMACDCMLOCKED 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-6 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 EMACCLIENTRXDVLD and EMACCLIENTRXFRAMEDROP 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-6 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 EMACCLIENTRXBADFRAME is asserted? See Frame Reception with Errors, in the *Virtex-6 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.
  - EMACPHYSYNACQSTATUS 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 being 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-6 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 Using the Auto-Negotiation Interrupt in the *Virtex-6 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. The detection of a bit error causes auto-negotiation to restart. Therefore, a link with an exceptionally high bit error rate might not be capable of completing auto-negotiation, or might 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 EMACPHYSYNACQSTATUS. 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 SIGNALDET 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 SIGNALDET 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 *UG366, Virtex-6 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. See *UG366, Virtex-6 FPGA GTX Transceivers* for more details.
- Minor bit error rates may 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-6 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 will 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 PHYEMACPHYAD[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, you will have contention reading address 0. Unless the attribute EMAC\_MDIO\_IGNORE\_PHYADZERO is enabled the internal PCS/PMA will respond 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. For more information on the EMAC\_MDIO\_IGNORE\_PHYADZERO attribute see Table 2-18 of the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.
- 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-6 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 will negotiate 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.

