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

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

Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.

© 2006-2010 Xilinx, Inc. Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE and other designated brands included herein are trademarks of Xilinx in the United States and other countries. The PowerPC name and logo are registered trademarks of IBM Corp. and used under license. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/18/06</td>
<td>1.1</td>
<td>Initial Xilinx release; core version 4.1, Xilinx tools 8.1i.</td>
</tr>
<tr>
<td>7/13/06</td>
<td>2.0</td>
<td>Core version 4.2, Xilinx tools 8.2i.</td>
</tr>
<tr>
<td>9/21/06</td>
<td>3.0</td>
<td>Updated core to version 4.3.</td>
</tr>
<tr>
<td>2/15/07</td>
<td>4.0</td>
<td>Updated core to version 4.4, Xilinx tools 9.1i.</td>
</tr>
<tr>
<td>8/8/07</td>
<td>5.0</td>
<td>Updated core to version 4.5; Xilinx tools 9.2i.</td>
</tr>
<tr>
<td>3/24/08</td>
<td>6.0</td>
<td>Updated core to version 4.6; Xilinx tools 10.1.</td>
</tr>
<tr>
<td>4/24/09</td>
<td>7.0</td>
<td>Updated core to version 4.7 and Xilinx tools to version 11.1.</td>
</tr>
<tr>
<td>4/19/10</td>
<td>8.0</td>
<td>Updated core to version 4.8 and Xilinx tools to version 12.1.</td>
</tr>
</tbody>
</table>
# Table of Contents

Revision History .......................................................... 2

Schedule of Figures ..................................................... 7

Preface: About This Guide

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

Chapter 1: Introduction

System Requirements ................................................. 13
About the Ethernet MAC Wrapper ................................. 13
Recommended Design Experience ................................. 13
Additional Resources .................................................. 14
Technical Support ....................................................... 14
Feedback ................................................................... 14
   Embedded Tri-Mode Ethernet MAC Wrapper ..................... 14
   Document ................................................................ 14

Chapter 2: Licensing the Core

Before you Begin ......................................................... 15
License Options .......................................................... 15
Obtaining Your License Key ......................................... 15
Installing Your License File ........................................... 16

Chapter 3: Quick Start Example Design

Overview ................................................................. 17
Generating the Ethernet MAC Wrapper ......................... 19
Implementing the Example Design ............................... 21
Running the Simulation ................................................ 21
   Functional Simulation .............................................. 21
   Timing Simulation .................................................... 23
What’s Next? ............................................................. 23
Chapter 4: Customizing the Core

Ethernet MAC Wrapper GUI Screens .......................................................... 25
  Core Configuration Options–Screen 1 ....................................................... 26
  EMAC Configuration Options–Screen 2 .................................................... 28
  EMAC Configuration–Screen 3 ................................................................. 30
  MDIO/EMAC Configuration–Screen 4 .......................................................... 32

Chapter 5: Detailed Example Design

Directory and File Contents ................................................................................. 36
  <project directory> ......................................................................................... 36
  <project directory>/<component name> ......................................................... 36
  <component name>/drivers ............................................................................ 37
  <component name>/doc .................................................................................. 37
  <component name>/example design ............................................................... 38
  example_design/client ................................................................................... 39
  client/fifo ...................................................................................................... 39
  example_design/physical .............................................................................. 40
  <component name>/implement ..................................................................... 40
  implement/results .......................................................................................... 41
  <component name>/simulation ..................................................................... 41
  simulation/functional .................................................................................... 42
  simulation/timing ........................................................................................... 42

Implementation and Test Scripts ........................................................................ 43
  Setting up for simulation ................................................................................ 43
  Implementation Scripts for Timing Simulation ............................................... 43
  Test Scripts For Timing Simulation ............................................................... 44
  Test Scripts For Functional Simulation ........................................................ 45

Example Design ................................................................................................. 46
  HDL Example Design ................................................................................... 46
  10 Mbps, 100 Mbps, 1 Gbps Ethernet FIFO .................................................. 47
  Address Swap Module .................................................................................... 49
  Physical Interface ........................................................................................... 50

Demonstration Test Bench .................................................................................. 51
  Test Bench Functionality ............................................................................... 51
  Changing the Test Bench ............................................................................... 53

Appendix A: Using the Client Side FIFO

Overview of LocalLink Interface ................................................................. 56
Receive FIFO Operation ................................................................................. 57
  LocalLink Interface ....................................................................................... 57
Transmit FIFO Operation .................................................................................. 58
  LocalLink Interface ....................................................................................... 58
User Interface Data Width Conversion .......................................................... 58
Appendix B: Constraining the Example Design

Device, Package, and Speedgrade Selection ........................................ 59
I/O Location Constraints ................................................................. 59
Timing Constraints ........................................................................... 59
  GMII Constraints .......................................................................... 59
  GMII with Byte PHY Constraints ....................................................... 63
  MII Constraints ........................................................................... 64
  MII with Clock Enable Constraints ..................................................... 65
  RGMII (v1.3 and v2.0) Constraints ...................................................... 66
  1000Base-X PCS/PMA (8-bit Client Interface) Constraints ......................... 70
  1000Base-X PCS/PMA (16-bit Client Interface) Constraints ......................... 70
  SGMII Constraints .................................................................... 71
  SGMII and 1000Base-X PCS/PMA Constraints .................................. 71
  Management Clock Constraints ....................................................... 72
  Example Design Constraints ............................................................. 72

Appendix C: SGMII / Dynamic Standards Switching

  SGMII Capabilities ....................................................................... 73
  FPGA Fabric Rx Elastic Buffer Requirement ........................................ 73
  The RocketIO Transceiver Rx Elastic Buffer ....................................... 75
    Closely Related Clock Sources ...................................................... 75
    Jumbo Frame Reception ............................................................... 76
Schedule of Figures

Chapter 1: Introduction

Chapter 2: Licensing the Core

Chapter 3: Quick Start Example Design

Chapter 4: Customizing the Core

Chapter 5: Detailed Example Design

Appendix A: Using the Client Side FIFO

Appendix B: Constraining the Example Design

Appendix C: SGMII / Dynamic Standards Switching
Preface

About This Guide

The Virtex®-4 FPGA Embedded Tri-Mode Ethernet MAC Wrapper Getting Started Guide provides information about generating an embedded Tri-Mode Ethernet MAC wrapper, customizing and simulating the wrapper files utilizing the provided example design, and running the design files through implementation using the Xilinx® tools.

Guide Contents

This guide contains the following chapters:

- **Preface, “About this Guide”** introduces the organization and purpose of the Getting Started Guide, including the conventions used in the guide.
- **Chapter 1, “Introduction”** describes the wrapper and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- **Chapter 3, “Quick Start Example Design,”** describes how to quickly generate the example design using the default parameters.
- **Chapter 4, “Customizing the Core,”** defines the Graphical User Interface 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, “Constraining the Example Design,”** describes the timing and placement constraints included with the example design.
- **Appendix C, “SGMII / Dynamic Standards Switching”** defines the SGMII capabilities for the core.
Conventions

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

**Typographical**

The following typographical conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning or Use</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courier font</td>
<td>Messages, prompts, and program files that the system displays. Signal names also.</td>
<td>speed grade: - 100</td>
</tr>
<tr>
<td>Courier bold</td>
<td>Literal commands that you enter in a syntactical statement</td>
<td>ngdbuild design_name</td>
</tr>
<tr>
<td>Helvetica bold</td>
<td>Commands that you select from a menu</td>
<td>File ➔ Open</td>
</tr>
<tr>
<td></td>
<td>Keyboard shortcuts</td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Italic font</td>
<td>Variables in a syntax statement for which you must supply values</td>
<td>ngdbuild design_name</td>
</tr>
<tr>
<td></td>
<td>References to other manuals</td>
<td>See the User Guide for more information.</td>
</tr>
<tr>
<td></td>
<td>Emphasis in text</td>
<td>If a wire is drawn so that it overlaps the pin of a symbol, the two nets are not connected.</td>
</tr>
<tr>
<td>Dark Shading</td>
<td>Items that are not supported or reserved</td>
<td>This feature is not supported</td>
</tr>
<tr>
<td>Square brackets</td>
<td>An optional entry or parameter. However, in bus specifications, such as bus[7:0], they are required.</td>
<td>ngdbuild [option_name] design_name</td>
</tr>
<tr>
<td>Braces</td>
<td>A list of items from which you must choose one or more</td>
<td>lowpwr = {on</td>
</tr>
<tr>
<td>Vertical bar</td>
<td>Separates items in a list of choices</td>
<td>lowpwr = {on</td>
</tr>
<tr>
<td>Angle brackets</td>
<td>User-defined variable or in code samples</td>
<td>&lt;directory name&gt;</td>
</tr>
<tr>
<td>Vertical ellipsis</td>
<td>Repetitive material that has been omitted</td>
<td>IOB #1: Name = QOUT’ IOB #2: Name = CLKN’</td>
</tr>
<tr>
<td>Horizontal ellipsis</td>
<td>Repetitive material that has been omitted</td>
<td>allow block block_name loc1 loc2 ... locn;</td>
</tr>
</tbody>
</table>
## Conventions

The following conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning or Use</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notations</strong></td>
<td>The prefix ‘0x’ or the suffix ‘h’ indicate hexadecimal notation</td>
<td>A read of address 0x00112975 returned 45524943h.</td>
</tr>
<tr>
<td></td>
<td>An ‘_n’ means the signal is active low</td>
<td><strong>usr_teof_n</strong> is active low.</td>
</tr>
</tbody>
</table>

## Online Document

The following conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning or Use</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Blue text</strong></td>
<td>Cross-reference link to a location in the current document</td>
<td>See the section “Guide Contents” for details.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See “Title Formats” in Chapter 1 for details.</td>
</tr>
<tr>
<td><strong>Blue, underlined text</strong></td>
<td>Hyperlink to a website (URL)</td>
<td>Go to <a href="http://www.xilinx.com">www.xilinx.com</a> for the latest speed files.</td>
</tr>
</tbody>
</table>
Introduction

The Virtex®-4 FPGA Embedded Tri-Mode Ethernet MAC (EMAC) wrapper supports Verilog®-HDL and VHDL. This chapter introduces the wrapper and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.

System Requirements

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

Linux
- Red Hat Enterprise Linux WS v4.0 32-bit/64-bit
- Red Hat Enterprise Desktop v5.0 32-bit/64-bit (with Workstation Option)
- SUSE Linux Enterprise (SLE) v10.1 32-bit/64-bit

Software
- ISE® 12.1

Check the release notes for the required Service Pack; ISE software Service Packs can be downloaded from www.xilinx.com/xlnx/xil_sw_updates_home.jsp?update=sp.

About the Ethernet MAC Wrapper

The Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC wrapper is included in the latest IP Update on the Xilinx® IP Center. For detailed information, visit the product page. The Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC wrapper is provided to all licensed Xilinx ISE software customers at no cost and can be generated using the Xilinx CORE Generator™ software v12.1 or higher.

Recommended Design Experience

Although the Virtex-4 FPGA Embedded Tri-Mode 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.
Chapter 1: Introduction

Additional Resources

For additional details and updates, see the Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC User Guide, accessible from the Virtex-4 FPGA User Guides page.

Technical Support

The fastest method for obtaining specific technical support for the Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC is through the www.xilinx.com/support website. Questions are routed to a team of engineers with specific expertise using the Virtex-4 FPGA Ethernet MAC wrapper.

Xilinx will provide technical support for use of this product as described in the Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC Data Sheet, Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC Getting Started Guide, and the Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC User Guide. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the Embedded Tri-Mode Ethernet MAC wrapper and the supplied documentation.

Embedded Tri-Mode Ethernet MAC Wrapper

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

- Product name
- Version number
- Explanation of your comments

Document

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

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments
Chapter 2

Licensing the Core

In ISE v12.1 and later, a license key is not required for full access to the Virtex-4 Ethernet MAC Wrapper. However, if you are using ISE 11.4 or older, please follow the instructions below for obtaining a license key before you use the core in your design. The Ethernet wrapper core is provided under the terms of the Xilinx End User Agreement, which conforms to the terms of the SignOnce IP License standard defined by the Common License Consortium.

Before you Begin

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

For information about installing the core, see the Ethernet Wrapper Product page.

License Options

After installing the required Xilinx® ISE® software and IP Service Packs, please see “Obtaining Your License Key” for instructions on obtaining a full license key.

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

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

Obtaining Your License Key

To obtain a Full license key for ISE v11.4 or older, please follow the instructions below. In ISE 12.1 and later, the license key requirement was removed.

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

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

This chapter provides instructions for generating the Virtex®-4 FPGA Embedded Tri-Mode Ethernet MAC wrapper using the default parameters.

Overview

The Virtex-4 FPGA Embedded Tri-Mode Embedded Ethernet MAC wrapper consists of the following:

- A wrapper file that connects the tie-off pins of each Ethernet MAC to the values selected in the CORE Generator™ software Graphical User Interface (GUI). In addition, unused inputs are tied low and unused outputs are disconnected.

- An example design that instantiates the Ethernet MAC wrapper. This design connects the transmit and receive client interfaces of each selected Ethernet MAC to a LocalLink FIFO. These FIFO are connected through an address swap module, which enables loop back of the received data. The interface logic for each of the selected physical interfaces is also instantiated along with the required IOBs. In addition, the example design implements an optimized clocking scheme.

- 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 shows the block diagram for the example design and the test bench provided with the Ethernet MAC wrapper. The design has been tested with the Xilinx® ISE® 12.1 software, Cadence Incisive Enterprise Simulator (IES) v9.2, and Mentor Graphics ModelSim v6.5c.
Generating the Ethernet MAC Wrapper

To start using the Ethernet MAC wrapper and example design:

1. Start the CORE Generator software.
   For help starting and using the CORE Generator tool, see the documentation supplied with the ISE tools, including the CORE Generator Guide at www.xilinx.com/support/software_manuals.htm.

2. Choose File > New Project.

3. Do the following to set project options:
   ♦ From Target Architecture, select Virtex-4.
     
     *Note:* If an unsupported silicon family is selected (a family other than Virtex-4 FX), the Ethernet MAC wrapper does not appear in the taxonomy tree.

   ♦ For Design Entry, select either VHDL or Verilog; for Vendor, select Other.

4. After creating the project, locate the directory containing the Ethernet MAC wrapper in the taxonomy tree. The project appears under one of the following:
   ♦ Communications & Networking /Ethernet
   ♦ Communications & Networking /Networking
   ♦ Communications & Networking/Telecommunications

5. Double-click the Embedded Tri-Mode Ethernet MAC wrapper icon. The Configuration Options dialog box appears.

6. In the Component Name field, enter a name for the core instance.
Chapter 3: Quick Start Example Design

7. Accept the remaining defaults; then click Finish to generate the core.

8. The wrapper and its supporting files, including the example design, are generated in your project directory. For a detailed description of the design example files and directories, see Chapter 5, “Detailed Example Design.”
Implementing the Example Design

The HDL example design can be processed through 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 tool project directory and `<component_name>` is the name entered in the Component Name field on the first GUI screen.

**Note:** Example design implementation is not supported when the DCR bus is used.

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

**Linux**

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

**Windows**

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

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

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

Running the Simulation

**Functional Simulation**

To run the functional simulation you must have the Xilinx Simulation Libraries compiled for your system. For more information on compiling libraries, 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)

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

**Virtex-4 Devices**

Virtex-4 device designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator.

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

- ModelSim v6.5c
- Cadence Incisive Enterprise Simulator (IES) v9.2

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

To run a VHDL functional simulation:

- Launch the simulator and set the current directory to:
  `<project_dir>/component_name/simulation/functional`
- For ModelSim, map the UniSim and SecureIP library:
  ```
  ModelSim> vmap unisim <path to compiled libraries>/unisim
  ModelSim> vmap secureip <path to compiled libraries>/secureip
  ```
- Launch the simulation script:
  ```
  ModelSim> do simulate_mti.do
  IES> ./simulate_ncsim.sh
  ```

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

Verilog Simulation

To run a Verilog functional simulation:

- Launch the simulator and set the current directory to:
  `<project_dir>/component_name/simulation/functional`
- For ModelSim, map the UniSim and SecureIP library:
  ```
  ModelSim> vmap unisim_ver <path to compiled libraries>/unisim_ver
  ModelSim> vmap secureip <path to compiled libraries>/secureip
  ```
- Launch the simulation script in one of the following ways:
  ```
  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 MACs.
What’s Next?

For detailed information about the example design, including guidelines for modifying the design and extending the test bench, see Chapter 5, “Detailed Example Design.”
Chapter 4

Customizing the Core

This chapter describes the Virtex®-4 FPGA Embedded Tri-Mode Ethernet MAC Wrapper Graphical User Interface (GUI), used to customize the core.

Ethernet MAC Wrapper GUI Screens

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

- "Core Configuration Options–Screen 1" — Used to name the core, select the desired software configuration interface, and enable the number of EMACs.

- "EMAC Configuration Options–Screen 2" — Used to select the PHY interface, speed, data width, global buffer usage options (for example, Byte PHY and Clock Enable), management data (MDIO) bus enable, and flow control configuration for the specified EMAC. If both EMACs are enabled, this screen is displayed twice—once for each enabled EMAC.

- "EMAC Configuration–Screen 3" — Used to set transmitter, receiver, and address filter configuration. If both EMACs are enabled, this screen is displayed twice—once for each enabled EMAC.

- "MDIO/EMAC Configuration–Screen 4" — This screen is only displayed if the Enable Management Data (MDIO) option is selected on the first "EMAC Configuration Options–Screen 2". For each enabled EMAC with the Enable Management Data (MDIO) option selected, this screen is displayed.
Core Configuration Options—Screen 1

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

Figure 4-1: Core Configuration Options

Component Name

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

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

- **Device Control Registers (DCR).** Accesses the configuration registers through DCR using the PowerPC® processor. When the DCR bus is used to access the internal registers of the Ethernet MAC, the DCR bus bridge in the host interface translates commands carried over the DCR bus into Ethernet MAC host bus signals. The resulting signals are input into one of the Ethernet MACs.

- **Host.** Accesses the Host Interface through the fabric. When the generic host bus is used, the \texttt{HOSTEMAC1SEL} signal selects either the host access of EMAC0 or EMAC1. When \texttt{HOSTEMAC1SEL} is asserted, the host accesses EMAC1. \texttt{HOSTEMAC1SEL} acts as the host address bit 10. If only one Ethernet MAC is used, this signal can be tied off to use either one of the Ethernet MACs during the power-up FPGA configuration.

- **None.** The Ethernet MACs are configured using tie-off pins—80 tie-off pins (\texttt{TIEEMAC#CONFIGVEC[79:0]}) are available to configure the Virtex-4 FPGA Ethernet MAC. The values of these tie-off pins are loaded into the Ethernet MAC at power-up or when the Ethernet MAC is reset. If the None option is selected, the transmit and receive engines must be enabled to ensure proper operation of the Ethernet MAC. The \texttt{TIEEMAC#CONFIGVEC[57]} and \texttt{TIEEMAC#CONFIGVEC[50]} tie-off pins are automatically set to high.

Enable EMACs

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

EMAC Configuration Options—Screen 2

This EMAC configuration screen lets you determine the Physical (PHY) interface, speed, data width, global buffer usage, management data (MDIO) bus enable, and flow control configuration for the specified EMAC. Some options on this screen are disabled depending on the PHY Interface selected; not all options are available with all PHY interface types. Some of these configurations will be overwritten when running simulation using the demonstration test bench. See “Demonstration Test Bench Tasks” in Chapter 5 for more information.

**PHY Interface**

Select the PHY interface type from the drop-down menu:

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

![EMAC Configuration Options](image)

**Figure 4-2: EMAC Configuration Options**
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 Mbps** Configures the core to run at a single rate.
- **10/100 Mbps** Configures the core to run at 10 or 100 Mbps.

Client Side Data Width

- **8-bit** -- An 8-bit data width is available for all interface types.
- **16-bit** -- A 16-bit client interface enables the over clocking mode for EMAC. It is available for the 1000 Base-X PCS/PMA interface; this enables the EMAC to operate at 250 MHz while the logic in the FPGA fabric is clocked at 125 MHz. The 16-bit option yields a 2.5 Gbps line rate.

Global Buffer Usage

Use these options to reduce the clock buffer usage.

- **Clock Enable** -- In MII mode, selecting Clock Enable reduces the number of BUFGs by requiring the user logic to use a separate clock-enable signal. See the Virtex-4 FPGA Tri-Mode Ethernet MAC User Guide for more information about determining the clock-enable signal setup.
- **Byte PHY** -- In Tri-Speed GMII mode, selecting Byte PHY reduces the number of BUFGs by adding the Byte PHY to the physical side logic.

Management Data

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

SGMII Capabilities

Select the SGMII Capabilities options:

- **10/100/1000 Mbps (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 RocketIO™ transceiver, subsequently consuming extra logic resources. However, this default mode provides reliable SGMII operation under all conditions.
- **10/100/1000 Mbps (restricted tolerance for clocks) OR 100/1000 Mbps** Uses the receiver elastic buffer present in the RocketIO transceivers. This is half the size and can potentially under- or overflow during SGMII jumbo frame reception at 10 Mbps operation. However, there are logical implementations where this can be proven reliable; if so, it is favored because of its lower logic utilization.

For detailed information about SGMII capabilities, see Appendix C, “SGMII / Dynamic Standards Switching.”
Flow Control Configuration

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

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

EMAC Configuration—Screen 3

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

Some of these configurations will be overwritten when running simulation using the demonstration test bench. See “Demonstration Test Bench Tasks” in Chapter 5 for more information.

*Figure 4-3: EMAC Configuration Options*
Transmitter Configuration

Transmitter configuration refers to the Ethernet MAC configuration registers located at 0x280. Initial values for several bits of this register can be set using the GUI. Changes to the register bits can be written using one of the host interfaces, if enabled. For more information, see “Configuration Registers,” in the *Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC User Guide*.

- **TX Reset** -- Initial value of this bit cannot be changed unless the Host or DCR bus is selected.
- **Jumbo Frame Enable** -- When selected, the transmitter sends frames greater than the maximum length specified in the *IEEE Std 802.3-2002*. When unselected, the transmitter sends only frames up to 1518 bytes (1522 bytes for VLAN).
- **In-band FCS Enable** -- When selected, this bit sets the Ethernet MAC transmitter to be ready for the FCS field from the client.
- **TX Enable 0** -- Initial value of this bit cannot be changed unless the Host or DCR bus is selected.
- **VLAN Enable** -- When selected, the VLAN transmitter allows transmission of the VLAN-tagged frames.
- **IFG Adjust Enable** -- When selected, the transmitter reads the value of `CLIENTEMAC#TXIFGDELAY` at the start of frame transmission and adjusts the IFG.

Receiver Configuration

Receiver configuration refers to the Ethernet MAC configuration registers located at 0x240. Initial values for several bits of this register can be set using the GUI. Changes to the register bits may be written using one of the host interfaces, if enabled. For more information, see “Configuration Registers,” in the *Virtex-4 Embedded Tri-Mode Ethernet MAC User Guide*.

- **RX Reset** -- Initial value of this bit cannot be changed unless the Host or DCR bus is selected.
- **Jumbo Frame Enable** -- When selected, the Ethernet MAC receiver accepts frames over the maximum length specified in the *IEEE Std 802.3-2002* specification. When unselected, the receiver accepts only frames up to the specified maximum.
- **In-band FCS Enable** -- When selected, the receiver passes the FCS field up to the client. When unselected, the FCS field is not passed to the client. In either case, the FCS is verified on the frame.
- **RX Enable** -- Initial value of this bit cannot be changed unless the Host or DCR bus is selected.
- **VLAN Enable** -- When selected, the receiver accepts VLAN tagged frames. The maximum allowed frame length increases by four bytes.
- **RX Disable Length** -- When selected, disables the length/type field check on the frame.

Duplex Settings

When Half-Duplex Enable is selected, the receiver and transmitter operate in half-duplex mode (applicable only for 10 and 100 Mbps). When unselected, the EMAC operates in full-duplex mode.
Address Filter Configuration

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

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

MDIO/EMAC Configuration—Screen 4

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

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

Note that some of these configurations will be overwritten when running simulation using the demonstration test bench. See “Demonstration Test Bench Tasks” in Chapter 5 for more information.
MDIO Configuration

- **PHY AN Enable** -- If selected, auto-negotiation is enabled.
- **PHY Isolate** -- If selected, the PHY is electrically isolated.
- **PHY Loopback MSB** -- If selected, the PHY loopback is enabled.

Several default MDIO configurations, for example PHY Reset and PHY Powerdown, need to be set manually in the EMAC wrapper and cannot be configured via the GUI.
Chapter 5

Detailed Example Design

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

- `<project directory>`
  Top-level project directory; name is user-defined.
  - `<project directory>/<component name>`
    Core release notes file
    - `<component name>/drivers`
      Product documentation
    - `<component name>/doc`
      Product documentation
    - `<component name>/example design`
      Verilog or VHDL design files
      - `example_design/client`
        Example client loopback logic
        - `client/fifo`
          FIFO files for the example client loopback logic
      - `example_design/physical`
        Physical interface description files
    - `<component name>/implement`
      Implementation script files
      - `implement/results`
        Results directory, created after implementation scripts are run, and contains implement script results
    - `<component name>/simulation`
      Simulation scripts
      - `simulation/functional`
        Functional simulation files
      - `simulation/timing`
        Timing simulation files
Directory and File Contents

The core directories and their associated files are defined in the following sections.

<project directory>

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

Table 5-1:  Project Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;project_dir&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;component_name&gt;.xco</td>
<td>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 tool for each core that it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator tool.</td>
</tr>
<tr>
<td>&lt;component_name&gt;_flist.txt</td>
<td>Text file listing all of the output files produced when the wrapper and example design files were generated in the CORE Generator software.</td>
</tr>
</tbody>
</table>

Back to Top

<project directory>/<component name>

The <component name> directory contains the release notes file provided with the core, detailing the changes and enhancements to the core.

Table 5-2:  Component Name Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;project_dir&gt;/&lt;component_name&gt;</td>
<td></td>
</tr>
<tr>
<td>v4_emac_readme.txt</td>
<td>Core release notes file</td>
</tr>
</tbody>
</table>

Back to Top
<component name>/drivers

The file in this directory is only generated when the DCR Bus is selected.

Table 5-3: Drivers Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;project_dir&gt;/&lt;component_name&gt;/drivers</code></td>
<td></td>
</tr>
<tr>
<td>v4_emac_l.h</td>
<td>Header file providing low-level driver functions for accessing the Ethernet MAC configuration information over the DCR bus. This file is only generated when the DCR Bus is selected.</td>
</tr>
</tbody>
</table>

<component name>/doc


Table 5-4: Doc Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;project_dir&gt;/&lt;component_name&gt;/doc</code></td>
<td></td>
</tr>
<tr>
<td>v4_emac_ds307.pdf</td>
<td>Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC Wrapper Data Sheet</td>
</tr>
<tr>
<td>v4_emac_gsg240.pdf</td>
<td>Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC Wrapper Getting Started Guide</td>
</tr>
</tbody>
</table>
<component name>/example design

This directory and sub-directories contain the support files necessary for a VHDL or Verilog implementation of the example design. See “Example Design,” page 46 for more information. This directory contains all the design files required for the example design.

Table 5-5: Example Design Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;project_dir&gt;/&lt;component_name&gt;/example_design</td>
<td></td>
</tr>
<tr>
<td>&lt;component_name&gt;.v[hd]</td>
<td>Ethernet MAC wrapper file</td>
</tr>
<tr>
<td>&lt;component_name&gt;_block.v[hd]</td>
<td>This lowest level of the hierarchy instantiates specific clock and reset circuitry required by the core. It also instantiates the appropriate physical interface specified when creating the core using the CORE Generator software. This layer provides a direct interface to the Client Side Interface of the Ethernet MAC primitive.</td>
</tr>
<tr>
<td>&lt;component_name&gt;_block.ucf</td>
<td>User constraints file (UCF) for the core and the example design. See Appendix B, “Constraining the Example Design” for additional information.</td>
</tr>
<tr>
<td>&lt;component_name&gt;_example_design.v[hd]</td>
<td>Top-level of the example design containing clocking and reset logic. In addition, the top-level design contains an address swap module; this module receives incoming Ethernet packets, swaps the source and destination address, and sends the Ethernet packet back. All received Ethernet packets are returned to the sending address.</td>
</tr>
<tr>
<td>&lt;component_name&gt;_locallink.v[hd]</td>
<td>This hierarchy level instantiates LocalLink FIFOs that translate between the Xilinx standard LocalLink Interface and the Client Side Interface of the Ethernet MAC primitive. This hierarchy level is useful for designers who want to interface to a Xilinx standard LocalLink interface.</td>
</tr>
<tr>
<td>dcm_reset.v[hd]</td>
<td>Only present when a DCM is used. This is a self-contained module for resetting a DCM following Power-on-Reset or loss of lock.</td>
</tr>
<tr>
<td>sync_block.v[hd]</td>
<td>Synchronizer module used for passing signals across a clock domain.</td>
</tr>
</tbody>
</table>

1. Note that although there are numerous ways to create the clocking and reset logic, only the implementation provided in the example has been tested and successfully verified to work in the example design.

Back to Top
example_design/client

This directory contains the support files necessary for the example client loopback logic connected to the Ethernet MAC client interfaces.

The 8-bit versions of the following files are only present when an 8-bit client interface has been selected. Similarly the 16-bit versions are only present when a 16-bit client interface has been selected.

Table 5-6: Client Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;project_dir&gt;/&lt;component_name&gt;/example_design/client</td>
<td></td>
</tr>
<tr>
<td>address_swap_module_[8</td>
<td>16].v[hd]</td>
</tr>
</tbody>
</table>

client/fifo

This directory contains the files for the FIFO that is instanced in the client loopback example design.

Table 5-7: FIFO Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;project_dir&gt;/&lt;component_name&gt;/example_design/client/fifo</td>
<td></td>
</tr>
<tr>
<td>eth_fifo_[8</td>
<td>16].v[hd]</td>
</tr>
<tr>
<td>tx_client_fifo_[8</td>
<td>16].v[hd]</td>
</tr>
<tr>
<td>rx_client_fifo_[8</td>
<td>16].v[hd]</td>
</tr>
</tbody>
</table>
example_design/physical

This directory contains the files that describe the physical interfaces of the Ethernet MAC. Appropriate files, from the following list, are delivered by the CORE Generator software depending on the physical interface selected.

Table 5-8: Physical Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;project_dir&gt;/&lt;component_name&gt;/example_design/physical</code></td>
<td>Generated if the MII or tri-speed GMII physical interface is selected for the Ethernet MAC. Assures correct FCS transmission.</td>
</tr>
<tr>
<td>fcs_blk_mii.v[hd]</td>
<td>Delivered if GMII is selected on one or both Ethernet MACs without the Advanced Clocking option (Byte PHY).</td>
</tr>
<tr>
<td>gmii_if.v[hd]</td>
<td>Delivered if GMII is selected on one or both Ethernet MACs with the Byte PHY Advanced Clocking option.</td>
</tr>
<tr>
<td>gmii_byte_phy_if.v[hd]</td>
<td>Delivered if MII is selected on one or both of the Ethernet MACs.</td>
</tr>
<tr>
<td>mii_if.v[hd]</td>
<td>Delivered if RGMII version 1.3 is selected on one or both of the Ethernet MACs.</td>
</tr>
<tr>
<td>rgmii_if.v[hd]</td>
<td>Delivered if RGMII version 2.0 is selected on one or both of the Ethernet MACs.</td>
</tr>
<tr>
<td>gt11_dual_1000X.v[hd]</td>
<td>If SGMII or 1000Base-X PCS/PMA or SGMII interface is selected for one or both Ethernet MACs, these files collectively interface MGTs to the physical interface.</td>
</tr>
</tbody>
</table>

<component name>/implement

The implement directory contains the core implementation script files.

Table 5-9: Implement Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;project_dir&gt;/&lt;component_name&gt;/implement</code></td>
<td>Generated if the MII or tri-speed GMII physical interface is selected for the Ethernet MAC. Assures correct FCS transmission.</td>
</tr>
<tr>
<td>implement.bat</td>
<td>Windows batch file that processes the example design through the Xilinx tool flow.</td>
</tr>
<tr>
<td>implement.sh</td>
<td>Linux shell script that processes the example design through the Xilinx tool flow.</td>
</tr>
<tr>
<td>xst.scr</td>
<td>XST script file for the example design.</td>
</tr>
<tr>
<td>xst.prj</td>
<td>XST project file for the example design; it enumerates all the HDL files that need to be synthesised.</td>
</tr>
</tbody>
</table>
**implement/results**

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

**Table 5-10: Results Directory**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;project_dir&gt;/&lt;component_name&gt;/implement/results</td>
<td></td>
</tr>
<tr>
<td>routed.sdf</td>
<td>Timing information for simulation</td>
</tr>
</tbody>
</table>

**<component name>/simulation**

The simulation directory and the sub-directories below it provide the files necessary to test a VHDL or Verilog implementation of the example design.

**Table 5-11: Simulation Directory**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;project_dir&gt;/&lt;component_name&gt;/simulation</td>
<td></td>
</tr>
<tr>
<td>demo_tb.v[hd]</td>
<td>VHDL or Verilog demonstration test bench for the Ethernet MAC wrapper</td>
</tr>
<tr>
<td>configuration_tb.v[hd]</td>
<td>Configuration test bench is instantiated in demo_tb.v[hd]. It provides stimuli to configure the Ethernet MACs via the selected management interface.</td>
</tr>
<tr>
<td>emac0_phy_tb.v[hd]</td>
<td>Physical interface test bench for EMAC0. This stimulates the receiver ports and monitors the transmitter ports of the EMAC0 physical interface. This is instantiated in demo_tb.v[hd] and is only present when EMAC0 has been selected.</td>
</tr>
<tr>
<td>emac1_phy_tb.v[hd]</td>
<td>Physical interface test bench for EMAC1. This stimulates the receiver ports and monitors the transmitter ports of the EMAC1 physical interface. This is instantiated in demo_tb.v[hd] and is only present when EMAC1 has been selected.</td>
</tr>
</tbody>
</table>
simulation/functional

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

Table 5-12:  Functional Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;project_dir&gt;/&lt;component_name&gt;/simulation/functional</code></td>
<td></td>
</tr>
<tr>
<td>simulate_mti.do</td>
<td>ModelSim macro file that compiles the example design sources and the structural simulation model then runs the functional simulation to completion.</td>
</tr>
<tr>
<td>wave_mti.do</td>
<td>ModelSim macro file that opens a wave window and adds interesting signals to it. It is called by the simulate_mti.do file.</td>
</tr>
<tr>
<td>simulate_ncsim.sh</td>
<td>IES script file that compiles the example design sources and the structural simulation model and then runs the functional simulation to completion.</td>
</tr>
<tr>
<td>wave_ncsim.sv</td>
<td>IES macro file that opens a wave window and adds interesting signals to it.</td>
</tr>
</tbody>
</table>

Back to Top

simulation/timing

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

Table 5-13:  Timing Directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;project_dir&gt;/&lt;component_name&gt;/simulation/timing</code></td>
<td></td>
</tr>
<tr>
<td>simulate_mti.do</td>
<td>ModelSim macro file that compiles the VHDL or Verilog timing model and demo test bench then runs the timing simulation to completion.</td>
</tr>
<tr>
<td>wave_mti.do</td>
<td>ModelSim macro file that opens a wave window and adds interesting signals to it. It is called by the simulate_mti.do macro file.</td>
</tr>
<tr>
<td>simulate_ncsim.sh</td>
<td>IES script file that compiles the VHDL or Verilog timing model and demo test bench and then runs the timing simulation to completion.</td>
</tr>
<tr>
<td>wave_ncsim.sv</td>
<td>IES macro file that opens a wave window and adds interesting signals to it.</td>
</tr>
</tbody>
</table>

Back to Top
Implementation and Test Scripts

Setting up for simulation

The Xilinx UniSim and SecureIP libraries must be mapped into simulator. If the UniSim and SecureIP libraries are not setup for your environment, go to Answer Record 15338 for assistance compiling Xilinx simulation models and for setting up the simulator environment.

Implementation Scripts for Timing Simulation

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

Note: The implementation scripts do not support the DCR bus.

Linux

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

Windows

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

The implement script performs the following steps:

- The HDL example design is synthesised using XST.
- Ngdbuild is run to consolidate the Ethernet MAC wrapper netlist and the HDL example netlist into the 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. For more information on the constraints file see Appendix B, “Constraining the Example Design.”
- The design is place-and-routed on the target device.
- Static timing analysis is performed on the routed design using trce.
- A bitstream is generated.
- Netgen runs on the routed design to generate VHDL and Verilog netlists and timing information in the form of SDF files.

The Xilinx tool flow generates several output and report files. These are saved in the following directory, 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.

For ModelSim

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

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

For IES

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

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

The test scripts perform the following tasks:

- Compiles the gate level netlist
- Compiles the demonstration test bench
- Starts a simulation of the test bench (with timing information if a Full-system Evaluation License or Full License is in use)
- Opens a Wave window and adds some signals of interest (wave_mti.do, wave_ncsim.sv)
- Runs the simulation to completion
Test Scripts For Functional Simulation

The test script that automates the functional simulation of the test bench.

For ModelSim

VHDL

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

Verilog

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

For IES

VHDL

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

Verilog

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

The test scripts perform the following tasks:

- Compiles the Ethernet MAC wrapper
- Compiles the example design files
- Compiles the demonstration test bench
- Starts a simulation of the test bench with no timing information
- Opens a Wave window and adds some signals of interest (wave_mti.do, wave_ncsim.sv)
- Runs the simulation to completion
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:

**VHDL**

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

**Verilog**

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

The HDL example design contains the following:

- An instance of the Ethernet MAC wrapper
- A client loopback module, includes an address swapping module and a receiver and transmitter FIFO
- Clock management logic, including DCM and Global Clock Buffer instances, where required
- GMII/MII, RGMII, SGMII or 1000 Base-X PCS/PMA interface logic, including MGTs, IOB and DDR registers instances, where required

The HDL example design connects the client side of the Ethernet MAC to the LocalLink FIFOs (which are in turn put in loop back through the address swap module) and the selected physical interface to external IOBs. 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 Mbps, 100 Mbps, 1 Gbps Ethernet FIFO

The 10 Mbps, 100 Mbps, 1 Gbps Ethernet FIFO is defined in the following files:

**VHDL**

```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
```

**Verilog**

```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
```
Chapter 5: Detailed Example Design

The 10 Mbps, 100 Mbps, 1 Gbps Ethernet FIFO contains an instance of the `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/written.

Figure 5-2 illustrates a straightforward frame transfer across the LocalLink interface. For more information about the FIFO, see Appendix A, “Using the Client Side FIFO.”

![Figure 5-2: Frame Transfer across LocalLink Interface](image)

**rx_client_fifo**

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

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

- The FIFO may 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 will not be able to 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 may overflow and data will be lost. It is therefore recommended that the example design is not used with the Ethernet MAC in jumbo frame mode for frames larger than 4000 bytes.
tx_client_fifo

The tx_client_fifo is built around 2 Dual Port Block RAMs, giving a total memory capacity of 4096 bytes of frame data.

When a full frame has been written into the transmit FIFO, the FIFO will present data to the MAC transmitter. On receiving the acknowledge signal from the Ethernet MAC the rest of the frame shall be transmitted, providing there is no retransmit request output by the Ethernet MAC. If a retransmission request is received, the frame will be queued for retransmission.

If the FIFO memory fills up, the dst_rdy_out_n signal will be used to halt the LocalLink interface writing in data, until space becomes available in the FIFO. If the FIFO memory fills up but no full frames are available for transmission (for example, if a frame larger than 4000 bytes is written into the FIFO), the FIFO will assert the tx_overflow signal and continue to accept the rest of the frame from the user. The overflow frame will be dropped by the FIFO. This ensures that the LocalLink interface does not lock up.

Address Swap Module

The address swap module is described in the following files:

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

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

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

An appropriate Physical Interface block is provided for each selected EMAC. This block connects the physical interface of the EMAC block to the I/O of the FPGA. Depending on the physical interface selected, the block contains the following components:

- For GMII/MII, this component will contain Input/Output block (IOB) buffers and IOB flip-flops.
- For RGMII, this component will contain IOB buffers and IOB Double-Data Rate flip-flops.
- For 1000BASE-X PCS/PMA or SGMII, this component will instantiate and connect MGTs. Calibration blocks are included and are connected to the instantiated MGTs. See the Calibration Block User Guide for detailed information. See Answer Record 22477 for more information about downloading the design files which include the Calibration Block User Guide.
Demonstration Test Bench

Test Bench Functionality

The demonstration test bench is defined in the following files:

**VHDL**

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

**Verilog**

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

The demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself.
The top-level test bench (demo_tb.vhd, demo_tb.v) consists of:

- 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

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

The physical layer test benches (emac0/1_phy_tb.vhd, emac0/1_phy_tb.v) consist of

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

Demonstration Test Bench Tasks

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The selected Ethernet MACs are configured through the management or configuration interface. The settings are:
  - Define the MDC clock frequency
  - Disable auto-negotiation in SGMII and 1000Base-X PCS/PMA modes (overwriting GUI configurations)
  - Disable flow control (overwriting GUI configurations)
  - Disable receiver and transmit reset configuration registers (overwriting GUI configurations)
  - Disable receiver and transmit half duplex mode (overwriting GUI configurations)
  - Disable Phy isolate feature for SGMII and 1000Base-X PCS/PMA modes (overwriting GUI configurations)
  - Disable the transceiver and EMAC loopback feature for SGMII and 1000Base-X PCS/PMA modes (overwriting GUI configurations)
  - Disable the unidirectional enable bit in PCS/PMA management register for SGMII and 1000Base-X PCS/PMA modes (overwriting GUI configurations)
  - Enable transmitter and receiver
- The configuration test bench then sets the speed of the selected Ethernet MACs.
• If EMAC0 has been selected to run at 1000 Mbps or in Tri-Speed mode, the following four frames are pushed into the EMAC0 receiver interface at 1 Gbps:
  ♦ The first frame is a minimum length frame.
  ♦ The second frame is a type frame.
  ♦ The third frame is an errored frame.
  ♦ The fourth frame is a padded frame.
• If EMAC1 has been selected to run at 1000 Mbps or in Tri-Speed mode, then the same four frames are applied to the EMAC1 receiver interface simultaneously.
• The frames received at the transmitter of each Ethernet MAC interface are checked against the stimulus frames to ensure data is the same.
• If applicable, the selected Ethernet MACs are configured through the management interface to run at 100 Mbps. The same four frames are then sent to the receiver interface and checked against the stimulus frames.
• If applicable, the selected Ethernet MACs are then configured through the management interface to run at 10 Mbps. The same four frames are then sent to the receiver interface and checked against the stimulus frames.

Changing the Test Bench

Changing Frame Data

It is possible to change the contents of the frame data passed into the Ethernet MAC receivers. This can be done by changing the data fields for each frame defined in the test bench. Further frames can be added by defining a new frame of data.

Changing Frame Error Status

Errors can be inserted into any of the pre-defined frames by changing the error field to ‘1’ in any column of that frame.

When an error is introduced into a frame, the bad_frame field for that frame must be set in order to disable the monitor checking for that frame.

The error currently written into the third frame can be removed by setting all error fields for the frame to ‘0’ and unsetting the bad_frame field.

Changing the Tri-Mode Ethernet MAC Configuration

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

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

The Ethernet MACs can be reconfigured by adding further 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 Mbps/100 Mbps/1 Gbps Ethernet FIFO consists of independent transmit and receive FIFOs embedded in a top-level wrapper. Figure A-1 shows how the FIFO fits into a typical implementation. Each FIFO is built around 2 Dual Port Block RAMs giving a memory capacity of 4096 bytes in each FIFO. This chapter describes the operation of the FIFO.

Figure A-1: Typical 10M/100M/1G Ethernet FIFO Implementation
Overview of LocalLink Interface

Data is transferred on the LocalLink interface from source to destination, with the flow governed by the four active low control signals sof_n, eof_n, src_rdy_n and dst_rdy_n. The flow of data is controlled by the src_rdy_n and dst_rdy_n signals. Only when these signals are asserted simultaneously is data transferred from source to destination. The individual packet boundaries are marked by the sof_n and eof_n signals. Figure A-2 shows the transfer of an 8-byte frame.

![Figure A-2: Frame Transfer across LocalLink Interface](image)

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](image)
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-4 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 will then be presented on the LocalLink interface for reading by the user. If the frame is marked as bad, that frame will be dropped by the FIFO.

LocalLink Interface

Table A-1 describes the receive FIFO LocalLink interface.

Table A-1: Receive FIFO LocalLink Interface

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Clock Domain</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_ll_clock</td>
<td>Input</td>
<td>N/A</td>
<td>Read clock for LocalLink interface</td>
</tr>
<tr>
<td>rx_ll_reset</td>
<td>Input</td>
<td>rx_ll_clock</td>
<td>Synchronous reset</td>
</tr>
<tr>
<td>rx_ll_data_out[7:0]</td>
<td>Output</td>
<td>rx_ll_clock</td>
<td>Data read from FIFO</td>
</tr>
<tr>
<td>rx_ll_sof_out_n</td>
<td>Output</td>
<td>rx_ll_clock</td>
<td>Start of frame indicator</td>
</tr>
<tr>
<td>rx_ll_eof_out_n</td>
<td>Output</td>
<td>rx_ll_clock</td>
<td>End of frame indicator</td>
</tr>
<tr>
<td>rx_ll_src_rdy_out_n</td>
<td>Output</td>
<td>rx_ll_clock</td>
<td>Source ready indicator</td>
</tr>
<tr>
<td>rx_ll_dst_rdy_in_n</td>
<td>Input</td>
<td>rx_ll_clock</td>
<td>Destination ready indicator</td>
</tr>
<tr>
<td>rx_fifo_status[3:0]</td>
<td>Output</td>
<td>rx_ll_clock</td>
<td>FIFO memory status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FIFO occupancy indication in units of 256 bytes (rounded down).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A value of 1 indicates FIFO space between 256-511 bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A value of 2 indicates FIFO space between 512-767 bytes and so on.</td>
</tr>
</tbody>
</table>

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

The FIFO status signal indicates the occupancy of the FIFO.
Appendix A: Using the Client Side FIFO

Transmit FIFO Operation

The transmit FIFO accepts frames over the LocalLink interface and stores them in block RAM for transmission via the EMAC. When a full frame is written into the transmit FIFO, the FIFO will present 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 Virtex-4 Embedded Tri-Mode Ethernet MAC User Guide.

LocalLink Interface

Table A-2 shows the transmit FIFO LocalLink interface signals.

Table A-2: Transmit FIFO LocalLink Interface

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Clock Domain</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx_ll_clock</td>
<td>Input</td>
<td>N/A</td>
<td>Write clock for LocalLink interface</td>
</tr>
<tr>
<td>tx_ll_reset</td>
<td>Input</td>
<td>tx_ll_clock</td>
<td>Synchronous reset</td>
</tr>
<tr>
<td>tx_ll_data_in[7:0]</td>
<td>Input</td>
<td>tx_ll_clock</td>
<td>Write data to be sent to transmitter</td>
</tr>
<tr>
<td>tx_ll_sof_in_n</td>
<td>Input</td>
<td>tx_ll_clock</td>
<td>Start of frame indicator</td>
</tr>
<tr>
<td>tx_ll_eof_in_n</td>
<td>Input</td>
<td>tx_ll_clock</td>
<td>End of frame indicator</td>
</tr>
<tr>
<td>tx_ll Src_rdy_in_n</td>
<td>Input</td>
<td>tx_ll_clock</td>
<td>Source ready indicator</td>
</tr>
<tr>
<td>tx_ll_dst_rdy_out_n</td>
<td>Output</td>
<td>tx_ll_clock</td>
<td>Destination ready indicator</td>
</tr>
<tr>
<td>tx_fifo_status[3:0]</td>
<td>Output</td>
<td>tx_ll_clock</td>
<td>FIFO occupancy indication in units of 256 bytes (rounded down).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A value of 1 indicates FIFO space between 256-511 bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A value of 2 indicates FIFO space between 512-767 bytes and so on.</td>
</tr>
</tbody>
</table>

In half-duplex operation, if the client interface collision signal is asserted by the EMAC, the current frame transmission will be terminated. If the retransmit signal is also asserted, the FIFO re-queues the frame for transmission.

If the FIFO memory fills up, the dst_rdy_out_n signal will be used to halt the LocalLink interface writing in data, until space becomes available in the FIFO. If the FIFO memory fills up but no frames are available for transmission (for example, if a frame larger than 4000 bytes is written into the FIFO), the FIFO will assert the tx_overflow signal and continue to accept the rest of the frame from the user. The overflow frame will be dropped by the FIFO. This ensures that the LocalLink interface does not lock up.

User Interface Data Width Conversion

Conversion of the user interface 8-bit data path to a 16, 32, 64 or 128-bit data path can be made by connecting the LocalLink interface directly to the Parameterizable LocalLink FIFO (see Xilinx Application Note XAPP691, “Parameterizable LocalLink FIFO”).
Appendix B

Constraining the Example Design

An example UCF is provided with the HDL example design, which provides examples of constraint requirements for the design.

Device, Package, and Speedgrade Selection

The Ethernet MAC UCF sets the part to a 4VFX60-FF672-10 device. This should be changed to the desired Virtex®-4 device.

I/O Location Constraints

Example placement is provided for the GMII, RGMII v1.3 and RGMII v2.0 physical interfaces. This should be changed to the desired pinout.

Timing Constraints

For more information on the clocking schemes used for each physical interface, see the Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC User Guide. Example constraints are given in the UCF delivered with the HDL example design for the core. These should be studied in conjunction with the HDL source code.

In the following examples, # refers to the Ethernet MAC number (EMAC0 or EMAC1).

GMII Constraints

IOB Constraints

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

# Place flip flops in IOBs
INST "gmii#?RXD_TO_MAC" IOB = true;
INST "gmii#?RX_DV_TO_MAC" IOB = true;
INST "gmii#?RX_ER_TO_MAC" IOB = true;
The GMII is a 3.3 volt signal-level interface. The following constraints set these IOs to use the LVTTL standard.

```
INST "gmii_txd_#<?>"     IOSTANDARD = LVTTL;
INST "gmii_tx_en_#"     IOSTANDARD = LVTTL;
INST "gmii_tx_er_#"     IOSTANDARD = LVTTL;
INST "gmii_rxd_#<?>"     IOSTANDARD = LVTTL;
INST "gmii_rx_dv_#"     IOSTANDARD = LVTTL;
INST "gmii_rx_er_#"     IOSTANDARD = LVTTL;
INST "gmii_tx_clk_#"     IOSTANDARD = LVTTL;
INST "gmii_rx_clk_#"     IOSTANDARD = LVTTL;
```

**Input Setup/Hold Timing**

The following GMII I/O constraints have been derived from the IEEE GMII timing specification. GMII requires a 2 ns setup and 0 ns hold time on the incoming data. '#' is 0 or 1 corresponding to the appropriate EMAC.

```
# Identify GMII Rx Pads
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 ns BEFORE "GMII_RX_CLK_#";
```

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

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

```
INST *gmii_rx_clk_#_dcm        CLKOUT_PHASE_SHIFT = FIXED;
INST *gmii_rx_clk_#_dcm        PHASE_SHIFT = -72;
```

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

In case of tri-speed operation, additional constraints for 10/100 Mbps are recommended. These two constraints put a bound on the incoming and outgoing data to ensure that a large skew between the data and clock is not present. These constraints are not in the IEEE specification and are merely recommendations.

```
INST "gmii_txd_#<?>"     TNM = "sig_gmii_tx_#";
INST "gmii_tx_en_#"     TNM = "sig_gmii_tx_#";
INST "gmii_tx_er_#"     TNM = "sig_gmii_tx_#";
INST "gmii_rxd_#<?>"     TNM = "sig_gmii_rx_#";
INST "gmii_rx_dv_#"      TNM = "sig_gmii_rx_#";
INST "gmii_rx_er_#"      TNM = "sig_gmii_rx_#";

# Put a 10 ns window on receive data, and a 15ns window from clock to out on the transmit side
TIMEGRP "sig_gmii_rx_#" OFFSET = IN 10 ns VALID 20 ns BEFORE "gmii_rx_clk_#";
TIMEGRP "sig_gmii_tx_#" OFFSET = OUT 15 ns AFTER "mii_tx_clk_#";
```
Clock Constraints

If an external GMII interface is implemented, the following constraint is always applied. Additionally, other constraints that vary according to the Ethernet MAC configuration are also required—see the appropriate following section.

PHYEMAC#GTXCLK Clock

This signal is routed to the PHYEMAC#GTXCLK port of the Ethernet MAC and is constrained to 125 MHz for 1 Gbps and Tri-Speed operation. The clock should be supplied by the user from a high quality source. This clock is not placed onto global clock routing. If both EMACs use a GMII, this clock is shared between them.

```
NET "gtx_clk_ibufg_#_i" TNM_NET = "clk_gtx_clk";
TIMESPEC "TS_gtx_clk" = PERIOD "clk_gtx_clk" 7200 ps HIGH 50 %;
```

1 Gbps Operation Only - EMAC0 or EMAC1

Transmitter Clock

At 1 Gbps speed only, the transmitter clock can be shared between client and PHY. The tx_gmii_mii_clk_in_#_i clock should be constrained to 125 MHz.

```
NET "tx_gmii_mii_clk_in_#_i" TNM_NET = "clk_phy_tx_clk#";
TIMESPEC "TS_phy_tx_clk#" = PERIOD "clk_phy_tx_clk#" 7200 ps HIGH 50 %;
```

Receiver Clock

At 1 Gbps speed only, the receiver clock can be shared between client and PHY. The gmii_rx_clk_0_i clock should be constrained to 125 MHz.

```
NET "gmii_rx_clk_0_i" TNM_NET = "clk_phy_rx_clk#";
TIMESPEC "TS_phy_rx_clk#" = PERIOD "clk_phy_rx_clk#" 7200 ps HIGH 50 %;
```

1 Gbps Operation Only - EMAC0 and EMAC1 Clock Optimizations

Transmitter Clock

At 1 Gbps speed only, the transmitter clock can be shared between client and PHY of both EMACs. The tx_gmii_mii_clk_in_0_i clock should be constrained to 125 MHz.

```
NET "tx_gmii_mii_clk_in_0_i" TNM_NET = "clk_phy_tx_clk0";
TIMESPEC "TS_phy_tx_clk0" = PERIOD "clk_phy_tx_clk0" 7200 ps HIGH 50 %;
```

Receiver Clocks

At 1 Gbps speed only, the receiver clocks can be shared between client and PHY of the same Ethernet MAC. However, each Ethernet MAC requires a separate receiver clock domain. The following clocks should both be constrained to 125 MHz.

```
NET "gmii_rx_clk_0_i" TNM_NET = "clk_phy_rx_clk0";
TIMESPEC "TS_phy_rx_clk0" = PERIOD "clk_phy_rx_clk0" 7200 ps HIGH 50 %;

NET "gmii_rx_clk_1_i" TNM_NET = "clk_phy_rx_clk1";
TIMESPEC "TS_phy_rx_clk1" = PERIOD "clk_phy_rx_clk1" 7200 ps HIGH 50 %;
```
Appendix B: Constraining the Example Design

Tri-Speed or 10/100 Mbps Operation

If an external GMII interface is implemented with standard clocking (no use of the Byte PHY mode), the following constraints should be applied. There are no clock optimizations that can be performed when both EMACs use this interface: the following constraints must be applied to both EMACs separately.

Transmitter Client Clock

The \texttt{tx\_client\_clk\_in\_#\_i} signal is connected to the \texttt{CLIENTEMAC\#TXCLIENTCLKIN} input of the Ethernet MAC and to the users client side transmitter logic. The clock should be constrained to 125 MHz for 1 Gbps operation.

\begin{verbatim}
NET "tx_client_clk_in_#_i" TNM_NET = "clk_client_tx_clk#";
TIMESPEC "TS_client_tx_clk#" = PERIOD "clk_client_tx_clk#" 7200 ps HIGH 50 %;
\end{verbatim}

Receiver Client Clock

The \texttt{rx\_client\_clk\_in\_#\_i} signal is connected to the \texttt{CLIENTEMAC\#RXCLIENTCLKIN} input of the Ethernet MAC and to the users client receive side logic. The clock should be constrained to 125 MHz for 1 Gbps operation.

\begin{verbatim}
NET "rx_client_clk_in_#_i" TNM_NET = "clk_client_rx_clk#";
TIMESPEC "TS_client_rx_clk#" = PERIOD "clk_client_rx_clk#" 7200 ps HIGH 50 %;
\end{verbatim}

Transmitter PHY Clock

The \texttt{tx\_gmii\_mii\_clk\_in\_#\_i} signal is routed to the \texttt{CLIENTEMAC\#TXGMIIMIIICLKIN} port of the Ethernet MAC along with the RGMII transmitter interface logic. The clock should be constrained to 125 MHz.

\begin{verbatim}
NET "tx_gmii_mii_clk_in_#_i" TNM_NET = "clk_phy_tx_clk#";
TIMESPEC "TS_phy_tx_clk#" = PERIOD "clk_phy_tx_clk#" 7200 ps HIGH 50 %;
\end{verbatim}

Receiver PHY Clock

The \texttt{rgmi_rxc_#_i} signal is routed to the \texttt{PHYEMAC\#RXCLK} port of the Ethernet MAC along with the RGMII receiver interface logic. The clock should be constrained to 125 MHz.

\begin{verbatim}
NET "gmii_rxc_#_i" TNM_NET = "clk_phy_rx_clk#";
TIMESPEC "TS_phy_rx_clk#" = PERIOD "clk_phy_rx_clk#" 7200 ps HIGH 50 %;
\end{verbatim}
GMII with Byte PHY Constraints

If an external GMII interface is implemented with the Byte PHY logic (Full Duplex operation only is supported), the following constraints should be applied. There are no clock optimizations that can be performed when both EMACs use this interface; the following constraints must be applied to both EMACs separately.

Input Setup/Hold Timing

GMII with Byte PHY follows the same I/O constraints from the IEEE GMII timing specification as the regular GMII. In Byte PHY configuration, dual data rate input registers are used on the Rx data, and the following timing constraints are used to analyze both the rising and falling edges.

# Define data valid window with respect to the clock rising edge.
# The spec states that, worst case, the data is valid 2 ns before the clock edge.
# The worst case is to provide zero hold time (a 2ns window in total)
TIMEGRP "DDR_RISING_#" = FFS;
TIMEGRP "DDR_FALLING_#" = FALLING FFS;
TIMEGRP "gmii_rx_#" OFFSET = IN 2 ns VALID 2 ns BEFORE "GMII_RX_CLK_#" TIMEGRP "DDR_RISING_#";
# Define data valid window with respect to the clock falling edge.
# The phase offset is advanced 2 ns after the rising clock edge (IN -2 ns), therefore putting it 2 ns before the falling edge
TIMEGRP "gmii_rx_#" OFFSET = IN -2 ns VALID 2 ns BEFORE "GMII_RX_CLK_#" TIMEGRP "DDR_FALLING_#";

Receiver Clock

The Byte PHY option allows the receiver clock to be shared between client and PHY. The mii_rx_clk_#_i clock should be constrained to 12.5 MHz for 10/100 Mbps operation.

NET "gmii_rx_clk_#_i" TNM_NET = "clk_phy_rx_clk#";
TIMESPEC "TS_phy_rx_clk#" = PERIOD "clk_phy_rx_clk#" 7200 ps HIGH 50 %;

Transmitter Clock

The Byte PHY option allows the transmitter clock to be shared between client and PHY. The tx_gmii_mii_clk_in_#_i clock should be constrained to 12.5 MHz for 10/100 Mbps operation.

NET "tx_gmii_mii_clk_in_0_i" TNM_NET = "clk_phy_tx_clk0";
TIMESPEC "TS_phy-txt_clk0" = PERIOD "clk_phy_tx_clk0" 7200 ps HIGH 50 %;
Appendix B: Constraining the Example Design

MII Constraints

I/O Constraints

The following MII I/O constraints are recommended in the MII specifications:

<table>
<thead>
<tr>
<th>Constraint</th>
<th>TNM</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST &quot;mii_txd_#&lt;?&gt;&quot;</td>
<td>TNM = &quot;sig_mii_tx_#&quot;;</td>
</tr>
<tr>
<td>INST &quot;mii_tx_en_#&quot;</td>
<td>TNM = &quot;sig_mii_tx_#&quot;;</td>
</tr>
<tr>
<td>INST &quot;mii_tx_er_#&quot;</td>
<td>TNM = &quot;sig_mii_tx_#&quot;;</td>
</tr>
<tr>
<td>INST &quot;mii_rxd_#&lt;?&gt;&quot;</td>
<td>TNM = &quot;sig_mii_rx_#&quot;;</td>
</tr>
<tr>
<td>INST &quot;mii_rx_dv_#&quot;</td>
<td>TNM = &quot;sig_mii_rx_#&quot;;</td>
</tr>
<tr>
<td>INST &quot;mii_rx_er_#&quot;</td>
<td>TNM = &quot;sig_mii_rx_#&quot;;</td>
</tr>
</tbody>
</table>

# using recommended budget from the clause 22
TIMEGRP "sig_mii_rx_#" OFFSET = IN 10 ns VALID 20 ns BEFORE "mii_rx_clk_#";
TIMEGRP "sig_mii_tx_#" OFFSET = OUT 15 ns AFTER "miiTx_clk_#";

The first constraint provides a setup/hold window of 10 ns on the receive data. The second constraint ensures the data is present on the output pins 15 ns after the clock. These are both only recommended specifications.

Clock Constraints

If an external MII interface is implemented with standard clocking (no use of the Clock Enables), the following constraints should be applied. There are no clock optimizations that can be performed when both EMACs use this interface; the following constraints must be applied to both EMACs separately.

Receiver Client Clock

The rx_client_clk_in_#_i signal is connected to the CLIENTEMAC#RXCLIENTCLKIN input of the Ethernet MAC and to the users client side receiver logic. The clock should be constrained to 12.5 MHz for 10/100 Mbps operation.

NET "rx_client_clk_in_#_i" TNM_NET = "clk_client_rx_clk#";
TIMESPEC "TS_client_rx_clk#" = PERIOD "clk_client_rx_clk#" 7200 ps HIGH 50 %;

Transmitter Client Clock

The tx_client_clk_in_#_i signal is connected to the CLIENTEMAC#TXCLIENTCLKIN input of the Ethernet MAC and to the users client side transmitter logic. The clock should be constrained to 12.5 MHz for 10/100 Mbps operation.

NET "tx_client_clk_in_#_i" TNM_NET = "clk_client_tx_clk#";
TIMESPEC "TS_client_tx_clk#" = PERIOD "clk_client_tx_clk#" 7200 ps HIGH 50 %;

Receiver PHY Clock

The mii_rx_clk_#_i signal is routed to the PHYEMAC#RXCLK port of the Ethernet MAC along with the MII transmitter interface logic. The clock should be constrained to 25 MHz.

NET "mii rx_clk_#_i" TNM_NET = "clk_phy_rx_clk#";
TIMESPEC "TS_phy_rx_clk#" = PERIOD "clk_phy_rx_clk#" 7200 ps HIGH 50 %;
Transmitter PHY Clock

The tx_gmi_mii_clk_in_#_i signal is routed to the CLIENTEMAC#TXGMIIMIICLKIN port of the Ethernet MAC along with the MII receiver interface logic. The clock should be constrained to 25 MHz.

```
NET "tx_gmi_mii_clk_in_#_i" TNM_NET = "clk_phy_tx_clk#";
TIMESPEC "TS_phy_tx_clk#" = PERIOD "clk_phy_tx_clk#" 7200 ps HIGH 50 %;
```

MII with Clock Enable Constraints

If an external MII interface is implemented with the Clock Enables, the following constraints should be applied. No clock optimization can be performed when both EMACs use this interface; the following constraints must be applied to both EMACs separately.

Receiver Clock

The clock enable option allows the receiver clock to be shared between client and PHY. The mii_rx_clk_#_i clock should be constrained to 12.5 MHz for 10/100 Mbps operation.

```
NET "mii_rx_clk_#_i" TNM_NET = "clk_phy_rx_clk#";
TIMESPEC "TS_phy_rx_clk#" = PERIOD "clk_phy_rx_clk#" 7200 ps HIGH 50 %;
```

Transmitter Clock

The clock enable option allows the transmitter clock to be shared between client and PHY. The tx_gmi_mii_clk_in_#_i clock should be constrained to 12.5 MHz for 10/100 Mbps operation.

```
NET "tx_gmi_mii_clk_in_0_i" TNM_NET = "clk_phy_tx_clk0";
TIMESPEC "TS_phy_tx_clk0" = PERIOD "clk_phy_tx_clk0" 7200 ps HIGH 50 %;
```

Additional Constraints

MII with clock enable requires the following additional constraints at the TEMAC boundary. These ensure that the data will cross the clock domains properly.

```
NET "*tx_mii_to_client_clk_?_r" MAXDELAY = 4000 ps;
NET "*rx_mii_to_client_clk_?_r" MAXDELAY = 4000 ps;
NET "*v4_emac_ll*rx_bad_frame_*_i" MAXDELAY = 6000 ps;
NET "*v4_emac_ll*rx_data_valid_*_i" MAXDELAY = 6000 ps;
NET "*v4_emac_ll*rx_good_frame_*_i" MAXDELAY = 6000 ps;
NET "*v4_emac_ll*rx_data_*_i*" MAXDELAY = 6000 ps;
NET "*v4_emac_ll*tx_ack_*_i" MAXDELAY = 6000 ps;
NET "*v4_emac_ll*tx_collision_*_i" MAXDELAY = 6000 ps;
NET "*v4_emac_ll*tx_retransmit_*_i" MAXDELAY = 6000 ps;
NET "*v4_emac_ll*tx_data_*_i*" MAXDELAY = 6000 ps;
NET "*v4_emac_ll*tx_data_valid_*_i" MAXDELAY = 6000 ps;
```
Appendix B: Constraining the Example Design

RGMII (v1.3 and v2.0) Constraints

IOB Constraints (v2.0 only)

The RGMII v2.0 is a 1.5 volt signal-level interface. The 1.5 volt HSTL Class I SelectIO™ interface standard is used for RGMII interface pins. Use the following constraints with the device IO Banking rules.

```
INST "rgmii_iob_0"            IOSTANDARD = HSTL_I;
INST "rgmii_txd_0<?>"         IOSTANDARD = HSTL_I;
INST "rgmii_tx_ctl_0"         IOSTANDARD = HSTL_I;
INST "rgmii_rxd_0<?>"         IOSTANDARD = HSTL_I;
INST "rgmii_rx_ctl_0"         IOSTANDARD = HSTL_I;
INST "rgmii_txc_0"            IOSTANDARD = HSTL_I;
INST "rgmii_rxc_0"            IOSTANDARD = HSTL_I;
```

Input Setup/Hold Timing

Tri-speed or 1Gbps Operation Only

The following RGMII I/O constraints have been derived from the Hewlett Packard RGMII timing specifications. RGMII needs a setup time of 1 ns, and a hold time of 1 ns on the receive data. ‘#’ is 0 or 1 corresponding to the appropriate EMAC.

```
INST "rgmii_rxd_#<?>" TNM = "rgmii_rx_#";
INST "rgmii_rx_ctl_#" TNM = "rgmii_rx_#";
TIMEGRP "DDR_RISING_#"             = FFS;
TIMEGRP "DDR_FALLING_#"           = FALLING FFS

# Define data valid window with respect to the clock rising edge.
# The spec states that, worst case, the data is valid 1 ns before the clock edge.
# The worst case is to provide 1 ns hold time (a 2ns window in total)
TIMEGRP "rgmii_rx_#" OFFSET = IN 1 ns VALID 2 ns BEFORE "RGMII_RXC_#"
TIMEGRP "DDR_RISING_#";

# Define data valid window with respect to the clock falling edge.
# The phase offset is advanced 3 ns after the rising clock edge (IN -3 ns), therefore putting it 1 ns before the falling edge
TIMEGRP "rgmii_rx_#" OFFSET = IN -3 ns VALID 2 ns BEFORE "RGMII_RXC_#"
TIMEGRP "DDR_FALLING_#";
```

The RGMII design uses a DCM on the receiver clock domain for all devices. Phase-shifting is then applied to the DCM to align the resultant clock so that it will correctly sample the 2 ns RGMII data valid window at the input flip-flops. The fixed phase shift is applied to the DCM using the following UCF syntax.

```
INST *rgmii_rxc_#_dcm                   CLKOUT_PHASE_SHIFT = FIXED;
INST *rgmii_rxc_#_dcm                   PHASE_SHIFT = -37;
```

The value of PHASE_SHIFT is preconfigured in the example designs to meet the setup and hold constraints for the example RGMII pinout in the particular device. The setup/hold timing which is achieved after place-and-route is reported in the data sheet section of the TRCE report (created by the implement script).
10/100 Mbps Operation Only

In 10/100Mbps operation, DCM is not used on the receiver clock domain due to slower clock frequencies. Therefore, the input hold constraints are relaxed from the Hewlett Packard RGMII timing specifications to meet timing for 100Mb/s.

# Define data valid window with respect to the clock rising edge.
# The spec states that, worst case, the data is valid 1 ns before the clock edge.
# The hold constraint is relaxed to 10ns for 10/100 Mbps to meet timing.
TIMEGRP "rgmii_rx_*" OFFSET = IN 1 ns VALID 11 ns BEFORE "RGMII_RXC_*"
TIMEGRP "DDR_RISING_*";

# Define data valid window with respect to the clock falling edge.
# The phase offset is advanced 19 ns after the rising clock edge (IN - 19 ns), therefore putting it 1 ns before the falling edge
TIMEGRP "rgmii_rx_*" OFFSET = IN -19 ns VALID 11 ns BEFORE "RGMII_RXC_*"
TIMEGRP "DDR_FALLING_*";

Finally, an IDELAY is used to create the 2 ns delay on the clock with respect to the data called for in the RGMII v2.0 specification. This is one of several possible implementations of this delay. See the Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC User Guide for more information on this mode.

# Set IDELAY value for 2ns delay on transmit clock w.r.t. the data
INST "*rgmii#!?rgmii_tx_clk_delay" IOBDELAY_VALUE = 0;

When IDELAY or IODELAY primitives are instantiated with a fixed delay attribute, an IDELAYCTRL component must be also instantiated to continuously calibrate the individual input delay elements. The IDELAYCTRL module requires a reference clock, which is assumed to be an input to the example design delivered by the CORE Generator™ tool. The most efficient way to use the IDELAYCTRL module is to lock the placement of the instance to the clock region of the device where the IDELAY/IODELAY components are placed. An example LOC constraint for the IDELAYCTRL module is shown in the following code. If your pins are in more than 1 clock region, you need an IDELAYCTRL for each region. See the Virtex-4 FPGA User Guide and code comments for more information.

# IDELAYCTRL location - dlyctrl_tx0 needs to be in same clock region as transmitter rgmii_txc_0 signal
INST "*dlyctrl_tx0" LOC = "IDELAYCTRL_X1Y3";
Appendix B: Constraining the Example Design

Clock Constraints

If an external RGMII interface is implemented, the following two constraints are always applied. An appropriate location for the BUFG related to RGMII inputs and outputs must be locked to meet the RGMII IO timing specifications. Additionally, other constraints that vary according to the Ethernet MAC configuration are also required—see the appropriate following section.

PHYEMAC#GTXCLK Clock

This signal is routed to the PHYEMAC#GTXCLK port of the Ethernet MAC and is constrained to 125 MHz for 1 Gbps and Tri-Speed operation. The clock should be supplied by the user from a high quality source. This clock is not placed onto global clock routing. If both EMACs use an RGMII, this clock is shared between them.

```
NET "gtx_clk_ibufg_#_i" TNM_NET = "clk_gtx_clk";
TIMESPEC "TS_gtx_clk" = PERIOD "clk_gtx_clk" 7200 ps HIGH 50 %;
```

IDELAY Reference Clock

The ref_clk_bufg_i signal is used to control the Virtex-4 FPGA IDELAY elements. This should be constrained to run at 200 MHz. This clock can be used globally by all IDELAY logic and for this reason can be shared by both EMACs. See the Virtex-4 FPGA User Guide for more information.

```
NET "refclk_bufg_i" TNM_NET = "clk_ref_clk";
TIMESPEC "TS_ref_clk" = PERIOD "clk_ref_clk" 5000 ps HIGH 50 %;
```

1 Gbps Operation Only - EMAC0 or EMAC1

Transmitter Clock

At 1 Gbps speed only, the transmitter clock can be shared between client and PHY. The tx_gmii_mii_clk_in_#_i clock should be constrained to 125 MHz.

```
NET "tx_gmii_mii_clk_in_#_i" TNM_NET = "clk_phy_tx_clk#";
TIMESPEC "TS_phy_tx_clk#" = PERIOD "clk_phy_tx_clk#" 7200 ps HIGH 50 %;
```

Receiver Clock

At 1 Gbps speed only, the receiver clock can be shared between client and PHY. The rgmii_rxc_#_i clock should be constrained to 125 MHz.

```
NET "rgmii_rxc_#_i" TNM_NET = "clk_phy_rx_clk#";
TIMESPEC "TS_phy_rx_clk#" = PERIOD "clk_phy_rx_clk#" 7200 ps HIGH 50 %;
```

1 Gbps Operation Only - EMAC0 and EMAC1 Clock Optimizations

Transmitter Clock

At 1 Gbps speed only, the transmitter clock can be shared between client and PHY of both EMACs. The tx_gmii_mii_clk_in_0_i clock should be constrained to 125 MHz.

```
NET "tx_gmii_mii_clk_in_0_i" TNM_NET = "clk_phy_tx_clk0";
TIMESPEC "TS_phy_tx_clk0" = PERIOD "clk_phy_tx_clk0" 7200 ps HIGH 50 %;
```
Receiver Clocks

At 1 Gbps speed only, the receiver clocks can be shared between client and PHY of the same Ethernet MAC. However, each Ethernet MAC requires a separate receiver clock domain. The following clocks should both be constrained to 125 MHz.

\[
\begin{align*}
\text{NET} & \quad \text{TNM_NET} = \text{clk\_phy\_rx\_clk0}; \\
\text{TIMESPEC} & \quad \text{TS\_phy\_rx\_clk0} = \text{PERIOD} \text{clk\_phy\_rx\_clk0} \ 7200 \ \text{ps} \ \text{HIGH} \ 50 \%; \\
\text{NET} & \quad \text{TNM_NET} = \text{clk\_phy\_rx\_clk1}; \\
\text{TIMESPEC} & \quad \text{TS\_phy\_rx\_clk1} = \text{PERIOD} \text{clk\_phy\_rx\_clk1} \ 7200 \ \text{ps} \ \text{HIGH} \ 50 \%; 
\end{align*}
\]

Tri-Speed or 10/100 Mbps Operation

There are no clock optimizations that can be performed when both EMACs use this interface; the following constraints must be applied to both EMACs separately.

Transmitter Client Clock

The \text{tx\_client\_clk\_in\_#\_i} signal is connected to the \text{CLIENTEMAC}\#\text{TXCLIENTCLKIN} input of the Ethernet MAC and to the users client side transmitter logic. The clock should be constrained to 125 MHz for 1 Gbps operation.

\[
\begin{align*}
\text{NET} & \quad \text{TNM_NET} = \text{clk\_client\_tx\_clk#}; \\
\text{TIMESPEC} & \quad \text{TS\_client\_tx\_clk#} = \text{PERIOD} \text{clk\_client\_tx\_clk#} \ 7200 \ \text{ps} \ \text{HIGH} \ 50 \%; 
\end{align*}
\]

Receiver Client Clock

The \text{rx\_client\_clk\_in\_#\_i} signal is connected to the \text{CLIENTEMAC}\#\text{RXCLIENTCLKIN} input of the Ethernet MAC and to the users client receive side logic. The clock should be constrained to 125 MHz.

\[
\begin{align*}
\text{NET} & \quad \text{TNM_NET} = \text{clk\_client\_rx\_clk#}; \\
\text{TIMESPEC} & \quad \text{TS\_client\_rx\_clk#} = \text{PERIOD} \text{clk\_client\_rx\_clk#} \ 7200 \ \text{ps} \ \text{HIGH} \ 50 \%; 
\end{align*}
\]

Transmitter PHY Clock

The \text{tx\_gmii\_mii\_clk\_in\_#\_i} signal is routed to the \text{CLIENTEMAC}\#\text{TXGMIIMIICLKIN} port of the Ethernet MAC along with the RGMII transmitter interface logic. The clock should be constrained to 125 MHz.

\[
\begin{align*}
\text{NET} & \quad \text{TNM_NET} = \text{clk\_phy\_tx\_clk#}; \\
\text{TIMESPEC} & \quad \text{TS\_phy\_tx\_clk#} = \text{PERIOD} \text{clk\_phy\_tx\_clk#} \ 7200 \ \text{ps} \ \text{HIGH} \ 50 \%; 
\end{align*}
\]

Receiver PHY Clock

The \text{rgmii\_rxc\_#\_i} signal is routed to the \text{PHYEMAC}\#\text{RXCLK} port of the Ethernet MAC along with the RGMII receiver interface logic. The clock should be constrained to 125 MHz.

\[
\begin{align*}
\text{NET} & \quad \text{TNM_NET} = \text{clk\_phy\_rx\_clk#}; \\
\text{TIMESPEC} & \quad \text{TS\_phy\_rx\_clk#} = \text{PERIOD} \text{clk\_phy\_rx\_clk#} \ 7200 \ \text{ps} \ \text{HIGH} \ 50 \%; 
\end{align*}
\]
Appendix B: Constraining the Example Design

1000Base-X PCS/PMA (8-bit Client Interface) Constraints

If an external 1000BASE-X PCS/PMA interface is implemented, the following constraints should be applied to the MGT clock circuitry.

**txoutclk1**

The **txoutclk1** signal is connected to the TXUSRCLK2 and RXUSRCLK2 clocks of any connected MGT. This clock should be constrained to 125 MHz.

```
NET "txoutclk1" TNM_NET = "clk_pcs_clk1";
TIMESPEC "TS_pcs_clk1" = PERIOD "clk_pcs_clk1" 7200 ps HIGH 50 %;
```

If an external PCS/PMA interface is implemented with an 8-bit client interface, **txoutclk1** can be shared across both transmitter and receiver client interfaces. Additionally, this clock can be shared across both EMACs.

1000Base-X PCS/PMA (16-bit Client Interface) Constraints

If an external PCS/PMA interface is implemented with a 16-bit client interface, the following constraints should also be applied, in addition to the MGT constraints described in “1000Base-X PCS/PMA (8-bit Client Interface) Constraints.”

**EMAC0** or **EMAC1**

If Ethernet MAC# is used with the 1000BASE-X PCS/PMA 16-bit client interface, the following constraint should be used to constrain the client interface clocks:

```
NET "tx_client_clk_in_#_i" TNM_NET = "clk_client_tx_clk#";
TIMESPEC "TS_client_tx_clk#" = PERIOD "clk_client_tx_clk#" 7200 ps HIGH 50 %;
```

The PCS clock for the EMAC running at 16 bit-client mode will now be 250 MHz; so the constraint needs to be modified (**clk_pcs_clk1** is for EMAC 0 and **clk_pcs_clk1_1** is for EMAC1). In the following example, EMAC0 is in 16-bit client mode and EMAC1 is in 8-bit client mode.

```
NET "*txoutclk1" TNM_NET = "clk_pcs_clk1";
NET "*txoutclk1_1" TNM_NET = "clk_pcs_clk1_1";
TIMESPEC "TS_pcs_clk1" = PERIOD "clk_pcs_clk1" 4000 ps HIGH 50 %;
TIMESPEC "TS_pcs_clk1_1" = PERIOD "clk_pcs_clk1_1" 7200 ps HIGH 50 %;
```

**EMAC0 and EMAC1 Clock Optimizations**

If EMAC0 and EMAC1 are both used with the 1000BASE-X PCS/PMA 16-bit client interfaces, the client clocks can be shared across both EMACs and only the following constraints need be applied:

```
NET "tx_client_clk_in_0_i" TNM_NET = "clk_client_tx_clk0";
TIMESPEC "TS_client_tx_clk0" = PERIOD "clk_client_tx_clk0" 7200 ps HIGH 50 %;
```

The PCS clock for both EMACs running in 16 bit-client mode will now be 250 MHz; so the constraint needs to be modified as follows:

```
NET "*txoutclk1" TNM_NET = "clk_pcs_clk1";
TIMESPEC "TS_pcs_clk1" = PERIOD "clk_pcs_clk1" 4000 ps HIGH 50 %;
```
SGMII Constraints

If an SGMII is implemented, the MGT constraints described in “1000Base-X PCS/PMA (8-bit Client Interface) Constraints” should be applied. Additional constraints may be required:

**EMAC0**

For an SGMII that operates at only 1 Gbps, no further constraints are necessary. For an SGMII that operates at multiple speeds or a speed other than 1 Gbps, the following constraint should be used to constrain the client interface clocks:

\[
\text{NET } ^*\text{tx_client_clk_in_0_i}^* \text{ TNM_NET } = ^*\text{clk_client_tx_clk0}^*; \\
\text{TIMESPEC } ^*\text{TS_client_tx_clk0}^* = \text{PERIOD } ^*\text{clk_client_tx_clk0}^* \ 7200 \ \text{ps HIGH 50 \%};
\]

**EMAC1**

For an SGMII which operates at only 1 Gbps, no further constraints need to be applied. For an SGMII which operates at multiple speeds or a speed other than 1 Gbps, the following constraint should be used to constrain the client interface clocks:

\[
\text{NET } ^*\text{tx_client_clk_in_1_i}^* \text{ TNM_NET } = ^*\text{clk_client_tx_clk1}^*; \\
\text{TIMESPEC } ^*\text{TS_client_tx_clk1}^* = \text{PERIOD } ^*\text{clk_client_tx_clk1}^* \ 7200 \ \text{ps HIGH 50 \%};
\]

Additional constraints are required for SGMII when the external fabric buffer is present, which constrain the recovered clock and the elastic buffer.

**EMAC0**

\[
\text{NET } ^*\text{RXRECCLK1_0}^* \text{ TNM_NET } = ^*\text{clk_rec_clk0}^*; \\
\text{TIMESPEC } ^*\text{TS_rec_clk0}^* = \text{PERIOD } ^*\text{clk_client_rec_clk0}^* \ 7200 \ \text{ps HIGH 50 \%}; \\
\text{NET } ^*\text{clock_correction_A/wr_addr_gray<*>^* MAXDELAY } = 7 \ \text{ns}; \\
\text{INST } ^*\text{clock_correction_A/rd_wr_addr_gray}^* \text{ TNM } = ^*\text{rx_graycode_A}^*; \\
\text{INST } ^*\text{clock_correction_A/rd_occupancy}^* \text{ TNM } = ^*\text{rx_binary_A}^*; \\
\text{TIMESPEC } ^*\text{ts_rx_meta_protect_A}^* = \text{FROM } ^*\text{rx_graycode_A}^* \text{ TO } ^*\text{rx_binary_A}^* \ 5 \ \text{ns};
\]

**EMAC1**

\[
\text{NET } ^*\text{RXRECCLK1_1}^* \text{ TNM_NET } = ^*\text{clk_rec_clk1}^*; \\
\text{TIMESPEC } ^*\text{TS_rec_clk1}^* = \text{PERIOD } ^*\text{clk_client_rec_clk1}^* \ 7200 \ \text{ps HIGH 50 \%}; \\
\text{NET } ^*\text{clock_correction_B/wr_addr_gray<*>^* MAXDELAY } = 7 \ \text{ns}; \\
\text{INST } ^*\text{clock_correction_B/rd_wr_addr_gray}^* \text{ TNM } = ^*\text{rx_graycode_B}^*; \\
\text{INST } ^*\text{clock_correction_B/rd_occupancy}^* \text{ TNM } = ^*\text{rx_binary_B}^*; \\
\text{TIMESPEC } ^*\text{ts_rx_meta_protect_B}^* = \text{FROM } ^*\text{rx_graycode_B}^* \text{ TO } ^*\text{rx_binary_B}^* \ 5 \ \text{ns};
\]

**SGMII and 1000Base-X PCS/PMA Constraints**

See the Solution Record 21605 for information about silicon-stepping levels.

\[
\text{CONFIG STEPPING } = ^*\text{SCD1}^*;
\]
Management Clock Constraints

host_clk_i

The host_clk_i signal must be constrained to run at the desired frequency. This is shared between the 2 EMACs. The clock can be connected to PHYEMACnGTXCLK and constrained to operate at 125 MHz to improve clock resource usage.

```plaintext
NET "host_clk_i" TNM_NET = "host_clock";
TIMESPEC "TS_clk_host" = PERIOD "host_clk" 10000 ps HIGH 50 %;
```

Example Design Constraints

The following additional constraints are required for the LocalLink FIFOs provided as part of example design.

16-bit Client Interface

```plaintext
# Tx client FIFO:

INST "*tx_fifo_i?wr_tran_frame_tog" TNM = "tx_metastable";
INST "*tx_fifo_i?frame_in_fifo_sync" TNM = "tx_metastable";
INST "*tx_fifo_i?wr_txfer_tog" TNM = "tx_metastable";
INST "*tx_fifo_i?wr_rd_addr*" TNM = "tx_metastable";

INST "*tx_fifo_i?wr_tran_frame_sync" TNM = "tx_stable";
INST "*tx_fifo_i?frame_in_fifo" TNM = "tx_stable";
INST "*tx_fifo_i?wr_txfer_tog_sync" TNM = "tx_stable";
INST "*tx_fifo_i?wr_addr_diff*" TNM = "tx_stable";

TIMESPEC "TS_tx_meta_protect" = FROM "tx_metastable" TO "tx_stable" 5 ns;

# Rx client FIFO:

INST "*rx_fifo_i?rd_store_frame_tog" TNM = "rx_metastable";
INST "*rx_fifo_i?wr_rd_addr_gray_sync*" TNM = "rx_metastable";

INST "*rx_fifo_i?rd_store_frame_sync" TNM = "rx_stable";
INST "*rx_fifo_i?wr_rd_addr_gray*" TNM = "rx_stable";

TIMESPEC "TS_rx_meta_protect" = FROM "rx_metastable" TO "rx_stable" 5 ns;
```

8-bit Client Interface

The following constraints are added in if the 8-bit client interface is used. Instantiate these constraints for each EMAC that you are using.

```plaintext
# Replace '#' with EMAC number.
INST "*client_side_FIFO_emac#?tx_fifo_i?wr_col_window_pipe_0" TNM = "tx_metastable";
INST "*client_side_FIFO_emac#?tx_fifo_i?wr_retran_frame_tog" TNM = "tx_metastable";
INST "*client_side_FIFO_emac#?tx_fifo_i?wr_col_window_pipe_1" TNM = "tx_stable";
INST "*client_side_FIFO_emac#?tx_fifo_i?wr_retran_frame_sync" TNM = "tx_stable";
```
SGMII / Dynamic Standards Switching

SGMII Capabilities

The Virtex®-4 FPGA Embedded Tri-Mode Ethernet MAC wrapper GUI provides two SGMII Capabilities options:

- **10/100/1000 Mbps (clock tolerance compliant with Ethernet specification)** Default setting that provides the implementation using the Receiver Elastic Buffer in FPGA fabric. This alternative Receiver Elastic Buffer uses a single block RAM to create a buffer twice as large as the one present in the RocketIO™ transceiver, subsequently consuming extra logic resources. However, this default mode provides reliable SGMII operation under all conditions.

- **10/100/1000 Mbps (restricted tolerance for clocks) OR 100/1000 Mbps** Uses the receiver elastic buffer present in the RocketIO transceivers. This is half the size and can potentially under- or overflow during SGMII frame reception at 10 Mbps 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 C-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 C-1, the clock source for the PHY is slightly faster than the clock source to the FPGA. Therefore, during frame reception, the receiver elastic buffer (shown here as implemented in the RocketIO transceiver) will start to fill up.

Following frame reception, in the interframe gap period, idles will be 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 RocketIO Transceiver User Guides).
### 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 Gbps operation, this translates into 1522 clock cycles
- At 100 Mbps operation, this translates into 15220 clock cycles (because each byte is repeated 10 times
- At 10 Mbps operation, this translates into 152200 clock cycles (because each byte is repeated 100 times).

Considering the 10 Mbps case, we would need \( \frac{152200}{5000} = 31 \) FIFO entries in the Elastic Buffer above and below the half way point to guarantee that the buffer will not under or overflow during frame reception. This assumes that frame reception begins when the buffer is exactly half full.

The size of the Rx Elastic Buffer in the RocketIO transceivers is of size 64 entries. However, we cannot assume that the buffer is exactly half full at the start of frame reception. Additionally, the underflow and overflow thresholds are not exact (please see the RocketIO Transceiver User Guides).

So to guarantee reliable SGMII operation at 10 Mbps (non-jumbo frames), the RocketIO 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 RocketIO Transceiver Rx Elastic Buffer

The Elastic Buffer in the RocketIO transceiver can be used reliably when:

- 10 Mbps operation is not required (for example, when connecting the core to the 1-Gigabit Ethernet MAC to provide only 1 Gbps operation). Please note that both 1 Gbps and 100 Mbps operation can be guaranteed.
- When the clocks are closely related (see the following section).

If there is any doubt, please select the FPGA fabric Rx Elastic Buffer Implementation.

Closely Related Clock Sources

Case 1

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

![Diagram of SGMII Implementation: Shared Clock Sources](image)

**Figure C-2: SGMII Implementation: Shared Clock Sources**

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

In this situation, the receiver elastic buffer will not under or overflow, and the elastic buffer implementation in the RocketIO transceiver should be used to save logic resources.
Case 2

Now consider again the case illustrated by Figure C-1. However, this time, assume that the clock sources used are both 50 ppm. Now the maximum frequency difference between the two devices will be 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 shown that this will provide reliable operation with the RocketIO transceiver Rx Elastic Buffers. Again, review 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 802.3 specification. Jumbo frames require special consideration to reliably receive frames. Table C-1 defines the maximum-size jumbo frames that can be received reliably when using the Receiver Elastic Buffer.

<table>
<thead>
<tr>
<th>Standard/Speed</th>
<th>Maximum Frame Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000BASE-X (1 Gbps only)</td>
<td>280000</td>
</tr>
<tr>
<td>SGMII (1 Gbps)</td>
<td>280000</td>
</tr>
<tr>
<td>SGMII (100 Mbps)</td>
<td>28000</td>
</tr>
<tr>
<td>SGMII (10 Mbps)</td>
<td>2800</td>
</tr>
</tbody>
</table>