# Virtex-4 Embedded Tri-Mode Ethernet MAC Wrapper v4.5

**Getting Started Guide** 

UG240 August 8, 2007





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

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

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

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

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

© 2006-2007 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

# **Revision History**

The following table shows the revision history for this document.

| Date    | Version | Revision                                                     |
|---------|---------|--------------------------------------------------------------|
| 1/18/06 | 1.1     | Initial Xilinx release; core version 4.1, Xilinx tools 8.1i. |
| 7/13/06 | 2.0     | Core version 4.2, Xilinx tools 8.2i.                         |
| 9/21/06 | 3.0     | Updated core to version 4.3.                                 |
| 2/15/07 | 4.0     | Updated core to version 4.4, Xilinx tools 9.1i.              |
| 8/8/07  | 5.0     | Updated core to version 4.5; Xilinx tools 9.2i.              |

# Table of Contents

# Preface: About This Guide

| Contents             | 9    |
|----------------------|------|
| Additional Resources | . 10 |
| 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                      |
| Technical Support                         |
| Feedback                                  |
| Embedded Tri-Mode Ethernet MAC Wrapper 14 |
| Document                                  |

## **Chapter 2: Quick Start Example Design**

| Overview                            | 15 |
|-------------------------------------|----|
| Generating the Ethernet MAC Wrapper | 17 |
| Implementing the Example Design     | 18 |
| Running the Simulation              | 19 |
| Functional Simulation               | 19 |
| Timing Simulation                   | 20 |
| What's Next?                        | 21 |

# **Chapter 3: Customizing the Core**

| Ethernet MAC Wrapper GUI Screens    | 23 |
|-------------------------------------|----|
| Core Configuration Options–Screen 1 | 24 |
| EMAC Configuration Options–Screen 2 | 25 |
| EMAC Configuration–Screen 3         | 28 |
| MDIO/EMAC Configuration–Screen 4    | 30 |

# **Chapter 4: Detailed Example Design**

| Directory and File Contents                                                 | <br>. 32 |
|-----------------------------------------------------------------------------|----------|
| <pre><pre>cproject directory&gt;</pre></pre>                                | <br>. 32 |
| <pre><project directory="">/<component name=""></component></project></pre> | <br>. 32 |
| <pre><component name="">/drivers</component></pre>                          | <br>. 33 |
| <component name="">/doc</component>                                         | <br>. 33 |
| <component name="">/example design</component>                              | <br>. 33 |
| example_design/client                                                       | <br>. 35 |
| client/fifo                                                                 | <br>. 35 |
| example_design/physical                                                     | <br>. 36 |

| <component name="">/implement</component>    | 36 |
|----------------------------------------------|----|
| implement/results                            | 37 |
| <component name="">/simulation</component>   | 37 |
| simulation/functional                        | 38 |
| simulation/timing                            | 38 |
| Implementation and Test Scripts              | 39 |
| Implementation Scripts for Timing Simulation | 39 |
| Test Scripts For Timing Simulation           | 39 |
| Test Scripts For Functional Simulation       | 41 |
| Example Design                               | 42 |
| HDL Example Design                           | 42 |
| 10 Mbps, 100 Mbps, 1 Gbps Ethernet FIFO      | 43 |
| Address Swap Module                          | 45 |
| Physical Interface                           | 45 |
| Demonstration Test Bench                     | 46 |
| Test Bench Functionality                     | 46 |
| Changing the Test Bench                      | 48 |

## Appendix A: Using the Client Side FIFO

| Overview of LocalLink Interface      | . 49 |
|--------------------------------------|------|
| Receive FIFO Operation               | . 50 |
| LocalLink Interface                  | . 50 |
| Transmit FIFO Operation              | 51   |
| LocalLink Interface                  | 51   |
| User Interface Data Width Conversion | . 52 |

## Appendix B: Constraining the Example Design

| Device, Package, and Speedgrade Selection                | <br>53   |
|----------------------------------------------------------|----------|
| I/O Location Constraints                                 | <br>53   |
| Timing Constraints                                       | <br>53   |
| GMII Constraints                                         | <br>53   |
| GMII with Byte PHY Constraints                           | <br>. 56 |
| MII Constraints                                          | <br>. 57 |
| MII with Clock Enable Constraints                        | <br>. 58 |
| RGMII (v1.3 and v2.0) Constraints                        | <br>. 59 |
| 1000Base-X PCS/PMA (8-bit Client Interface) Constraints  | <br>62   |
| 1000Base-X PCS/PMA (16-bit Client Interface) Constraints | <br>. 62 |
| SGMII Constraints                                        | <br>62   |
| SGMII and 1000Base-X PCS/PMA Constraints                 | <br>63   |
| Management Clock Constraints                             | <br>63   |
| Example Design Constraints                               | <br>63   |

## Appendix C: SGMII / Dynamic Standards Switching

| FPGA Fabric Rx Elastic Buffer Requirement | 67 |
|-------------------------------------------|----|
| The RocketIO Rx Elastic Buffer            | 69 |
| Jumbo Frame Reception                     | 70 |

# Schedule of Figures

# Chapter 2: Quick Start Example Design

| Figure 2-1: Default Example Design and Test Bench | 16 |
|---------------------------------------------------|----|
| Figure 2-2: Core Configuration Options            | 18 |

# **Chapter 3: Customizing the Core**

| Figure 3-1: Core Configuration Options | 24 |
|----------------------------------------|----|
| Figure 3-2: EMAC Configuration Options | 26 |
| Figure 3-3: EMAC Configuration Options | 28 |
| Figure 3-4: MDIO Configuration         | 30 |

# Chapter 4: Detailed Example Design

| Figure 4-1: HDL Example Design                                      | 42 |
|---------------------------------------------------------------------|----|
| Figure 4-2:    Frame Transfer across LocalLink Interface            | 44 |
| Figure 4-3:       Modification of Frame Data by Address Swap Module | 45 |
| Figure 4-4:    Demonstration Test Bench                             | 46 |

## Appendix A: Using the Client Side FIFO

| Figure A-1: | Typical 10M/100M/1G Ethernet FIFO Implementation | 49 |
|-------------|--------------------------------------------------|----|
| Figure A-2: | Frame Transfer across LocalLink Interface        | 50 |
| Figure A-3: | Frame Transfer with Flow Control                 | 50 |

## Appendix C: SGMII / Dynamic Standards Switching

| Figure C-1: | SGMII Implementation: Separate Clock Sources | 68 |
|-------------|----------------------------------------------|----|
| Figure C-2: | SGMII Implementation: Shared Clock Sources   | 69 |

# 



# Preface

# About This Guide

The Virtex-4<sup>TM</sup> Embedded Tri-Mode Ethernet MAC Wrapper v4.5 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.

## 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 and a list of additional resources.
- Chapter 1, "Introduction" describes the wrapper and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Quick Start Example Design," describes how to quickly generate the example design using the default parameters.
- Chapter 3, "Customizing the Core," defines the Graphical User Interface options.
- Chapter 4, "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.

# **Additional Resources**

For additional information, visit <u>www.xilinx.com/support</u>. The following table lists some of the resources you can select from this website. You can also directly access these resources using the provided URLs.

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

# Conventions

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

## **Typographical**

The following typographical conventions are used in this document:

| Convention Meaning or Use |                                                                     | Example              |
|---------------------------|---------------------------------------------------------------------|----------------------|
| Courier font              | Messages, prompts, and<br>program files that the system<br>displays | speed grade: - 100   |
| Courier bold              | Literal commands that you enter in a syntactical statement          | ngdbuild design_name |



| Convention                      | Meaning or Use                                                                                                        | Example                                                                                                  |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
|                                 | Variables in a syntax<br>statement for which you must<br>supply values                                                | ngdbuild design_name                                                                                     |
| Italic font                     | References to other manuals                                                                                           | See the <i>Development System</i><br><i>Reference Guide</i> for more<br>information.                     |
|                                 | Emphasis in text                                                                                                      | If a wire is drawn so that it<br>overlaps the pin of a symbol, the<br>two nets are <i>not</i> connected. |
| <text brackets="" in=""></text> | User-defined variable for directory names.                                                                            | <component_name></component_name>                                                                        |
| Square brackets []              | An optional entry or<br>parameter. However, in bus<br>specifications, such as<br><b>bus[7:0]</b> , they are required. | <b>ngdbuild</b> [option_name]<br>design_name                                                             |
|                                 | Also used with pipe symbol to indicate either one or the other.                                                       | eth_fifo_[8   16].v                                                                                      |
| Braces { }                      | A list of items from which you must choose one or more                                                                | lowpwr ={on off}                                                                                         |
| Vertical bar                    | Separates items in a list of choices                                                                                  | lowpwr ={on off}                                                                                         |
| Vertical ellipsis               | Repetitive material that has been omitted                                                                             | IOB #1: Name = QOUT'<br>IOB #2: Name = CLKIN'                                                            |
| Horizontal ellipsis             | Repetitive material that has been omitted                                                                             | <b>allow block</b> block_name<br>loc1 loc2 locn;                                                         |

# **Online Document**

The following conventions are used in this document:

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



# Chapter 1

# Introduction

The Virtex-4 Embedded Tri-Mode Ethernet MAC (MAC) 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® 2000 Professional with Service Pack 2-4
- Windows XP Professional with Service Pack 1

#### Solaris/Linux

- Sun Solaris®9/10
- Red Hat® Enterprise Linux 4.0 (32-bit and 64-bit)

#### Software

■ ISE<sup>TM</sup> 9.2i

Check the release notes for the required Service Pack; ISE Service Packs can be downloaded from <a href="https://www.xilinx.com/xlnx/xil\_sw\_updates\_home.jsp?update=sp">www.xilinx.com/xlnx/xil\_sw\_updates\_home.jsp?update=sp</a>.

# About the Ethernet MAC Wrapper

The Virtex-4 Embedded Tri-Mode Ethernet MAC wrapper is included in the latest IP Update on the Xilinx IP Center. For detailed information, visit: www.xilinx.com/xlnx/xebiz/designResources/ip\_product\_details.jsp?key=Embedded\_TEM AC\_Wrapper

The Virtex-4 Embedded Tri-Mode Ethernet MAC wrapper is provided to all licensed Xilinx ISE customers at no cost and can be generated using the Xilinx CORE Generator<sup>™</sup> v9.2i or higher.

# **Recommended Design Experience**

Although the Virtex-4 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.

**XILINX**<sup>®</sup>

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

## **Additional Resources**

For additional details and updates, see the *Virtex-4 Embedded Tri-Mode Ethernet MAC User Guide*, accessible from <u>www.xilinx.com/bvdocs/userguides/ug074.pdf</u>.

# **Technical Support**

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

Xilinx will provide technical support for use of this product as described in the *Virtex-4 Embedded Tri-Mode Ethernet MAC Data Sheet, Virtex-4 Embedded Tri-Mode Ethernet MAC Getting Started Guide,* and the *Virtex-4 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 <u>www.xilinx.com/support</u>. 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 <u>www.xilinx.com/support</u>. 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

# Quick Start Example Design

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

## **Overview**

The Virtex-4 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 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 2-1 shows the block diagram for the example design and the test bench provided with the Ethernet MAC wrapper. The design has been tested with Xilinx ISE 9.2i, Cadence<sup>™</sup> IUS v5.8 and Mentor Graphics<sup>®</sup> ModelSim<sup>®</sup> PE/SE 6.1e.



Figure 2-1: Default Example Design and Test Bench



# **Generating the Ethernet MAC Wrapper**

To start using the Ethernet MAC wrapper and example design:

1. Start the CORE Generator.

For help starting and using the CORE Generator, see the documentation supplied with ISE, 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.

| 🐐 Embedded T                                                                 | ri-mode Ethernet NAC Wrapper                  | ×    |
|------------------------------------------------------------------------------|-----------------------------------------------|------|
| logi <mark>C</mark> ŘRE                                                      | Embedded Tri-mode Ethernet MAC<br>Wrapper     | v4.5 |
| Component Nam                                                                | ne v4 emac v4 f                               |      |
| -Shared Interfa                                                              |                                               |      |
| Host interface                                                               | configuration.                                |      |
| -Host Type-                                                                  | -                                             |      |
| The host int                                                                 | terlace type is shared between the two EMACs. |      |
| 🔿 DCR                                                                        |                                               |      |
| 🔿 Host                                                                       |                                               |      |
| <ul> <li>None</li> </ul>                                                     | 3                                             |      |
| Enable EMAC:<br>Enable EMAC:<br>Enable EMAC:<br>Enable EMAC:<br>Enable EMAC: | s<br>MAC 0<br>MAC 1                           |      |
|                                                                              |                                               | _    |
|                                                                              |                                               |      |
|                                                                              |                                               |      |
|                                                                              |                                               |      |
|                                                                              |                                               |      |
|                                                                              |                                               |      |
| View Data Sheet                                                              | Page 1 of 5 < Back Next > Finish Cancel       |      |

Figure 2-2: Core Configuration Options

- 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 4, "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 the user in running the Xilinx software.

In the following examples, <project\_dir> is the CORE Generator 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:

#### UNIX

```
unix-shell% cd <component_name>/implement
unix-shell% ./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 <u>www.xilinx.com/support/software\_manuals.htm</u>.

In addition, the simulator used must provide SWIFT model support to simulate the Ethernet MAC and the Virtex-4 RocketIO Multi-Gigabit Transceivers (MGT).

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

#### VHDL Simulation

To run a VHDL functional simulation:

- Launch the simulator and set the current directory to: <project\_dir>/<component\_name>/simulation/functional
- For ModelSim, map the UniSim library:

ModelSim> vmap unisim <path to compiled libraries>/unisim

• Launch the simulation script:

ModelSim> do simulate\_mti.do

IUS> ./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 library:

ModelSim> vmap unisims\_ver cpath to compiled libraries>/unisims\_ver

• Launch the simulation script in one of the following ways:

ModelSim> do simulate\_mti.do

IUS> ./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.

### Timing Simulation

To run the gate-level 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 <u>www.xilinx.com/support/software\_manuals.htm</u>.

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

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

#### **VHDL Simulation**

To run a VHDL timing simulation:

- Launch the simulator and set the current directory to: <project\_dir>/<component\_name>/simulation/timing
- For ModelSim, map the SimPrim library:

ModelSim> vmap simprim <path to compiled libraries>/simprim

• Launch the simulation script in one of the following ways:

ModelSim> do simulate\_mti.do

IUS> ./simulate\_ncsim.sh

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

#### Verilog Simulation

To run a Verilog timing simulation:

- Launch the ModelSim simulator and set the current directory to: <project\_dir>/<component\_name>/simulation/timing
- For ModelSim map the SimPrim library:

ModelSim> vmap simprims\_ver <path to compiled\_libraries>/simprims\_ver



• Launch the simulation script:

ModelSim> do simulate\_mti.do

IUS> ./simulate\_ncsim.sh

The scripts compile the gate-level netlist 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 4, "Detailed Example Design."



# Chapter 3

# **Customizing the Core**

This chapter describes the Virtex-4 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.

| 👯 Embedded Tri-mode Ethernet MAC Wrapper                 | X    |
|----------------------------------------------------------|------|
| Logic Embedded Tri-mode Ethernet MAC<br>Wrapper          | √4.5 |
| Component Name v4 emac. v4 5                             |      |
| - Shared Interfaces                                      |      |
| Host interface configuration.                            |      |
| Host Type                                                |      |
| The host interface type is shared between the two EMACs. |      |
| O DCB                                                    |      |
|                                                          |      |
|                                                          |      |
| None                                                     |      |
|                                                          |      |
| CEnable EMACs                                            |      |
| Enable EMAC 0                                            |      |
| Enable EMAC 1                                            |      |
|                                                          |      |
|                                                          |      |
|                                                          |      |
|                                                          |      |
|                                                          |      |
|                                                          |      |
|                                                          |      |
|                                                          |      |
|                                                          |      |
|                                                          |      |
| New Date Chard                                           |      |
|                                                          |      |

Figure 3-1: 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<sup>TM</sup> 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 HOSTEMAC1SEL signal selects either the host access of EMAC0 or EMAC1. When HOSTEMAC1SEL is asserted, the host accesses EMAC1. HOSTEMAC1SEL acts as

the host address bit 10. If only one Ethernet MAC is used, this signal can be tied off to use either one of the Ethernet MACs during the power-up FPGA configuration.

• None. The Ethernet MACs are configured using tie-off pins—80 tie-off pins (TIEEMAC#CONFIGVEC[79:0]) are available to configure the Virtex-4 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 TIEEMAC#CONFIGVEC[57] and 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.

## 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.

| 🕴 Embedded Tri-mode Ethernet MAC Wrapper 🛛 🔀 |                                                     |                             |  |
|----------------------------------------------|-----------------------------------------------------|-----------------------------|--|
| logi <del>CKRE</del>                         | E Embedded Tri-mode Ethernet MAC<br>Wrapper v4.5    |                             |  |
|                                              |                                                     | EMAC 0 Configuration        |  |
| Phy Interface:                               | GMII                                                |                             |  |
| Speed                                        |                                                     | Client Side Data Width      |  |
| 🔿 Tri-spe                                    | ed                                                  | 8-bit                       |  |
| ⊙ 1000 M                                     | Ibps                                                | 0 16-bit                    |  |
| 0 10/100                                     | ) Mbps                                              |                             |  |
| -Global Buffer (                             | Jsage                                               | Management Data             |  |
| Options to rec                               | luce clock buffer usage                             | Enable Management Data      |  |
| Clock En                                     | able                                                | MDID                        |  |
| Byte PHY                                     | ,                                                   |                             |  |
| SGMII Capabi                                 | lities                                              |                             |  |
| 10/100     10/100     10/100                 | 0/1000 Mb/s (clock tolerance compliant with ethern  | et specification)           |  |
| 0 10/100                                     | )/1000 Mb/s (restricted tolerance for clocks) OR 10 | 0/1000 Mb/s                 |  |
|                                              |                                                     |                             |  |
| -Flow Control C                              | Configuration                                       |                             |  |
| TX Flow (                                    | Control Enable                                      |                             |  |
| RX Flow                                      | Control Enable                                      |                             |  |
|                                              |                                                     |                             |  |
| View Data Sheet                              | Page 2 of 5                                         | < Back Next > Finish Cancel |  |

Figure 3-2: EMAC Configuration Options

### **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

## 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 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, 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 RocketIOs. 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.

| 💐 Embedded Tri-mode Ethernet MAC Wrapper 🛛 🛛 🔀 |                                  |  |
|------------------------------------------------|----------------------------------|--|
| Embedded Tri-mode Ethernet MAC<br>Wrapper ب4.5 |                                  |  |
|                                                | EMAC 0 Configuration             |  |
| Transmitter Configuration                      | Receiver Configuration           |  |
| Initial transmitter settings                   | Initial receiver settings        |  |
| TX Reset                                       | RX Reset                         |  |
| 🔲 Jumbo Frame Enable                           | Jumbo Frame Enable               |  |
| In-band FCS Enable                             | In-band FCS Enable               |  |
| ✓ T× Enable 0                                  | RX Enable                        |  |
| 🔲 VLAN Enable                                  | ULAN Enable                      |  |
| 🔲 IFG Adjust Enable                            | RX Disable Length                |  |
| - Duplex Settings                              |                                  |  |
| Half Duplex Enable                             |                                  |  |
| Address Filter Configuration                   |                                  |  |
| Initial Address Filter settings                |                                  |  |
| Address Filter Enable                          |                                  |  |
| Unicast Pause MAC Address AA BB CC DD          | EE FF                            |  |
|                                                |                                  |  |
|                                                |                                  |  |
|                                                |                                  |  |
|                                                |                                  |  |
|                                                |                                  |  |
|                                                |                                  |  |
| View Data Sheet Page 3                         | of 5 < Back Next > Finish Cancel |  |

Figure 3-3: 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 Embedded Tri-Mode Ethernet 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 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.

| Embedded Tri-mode Etherr     | <sup>tet MAC Wrapper</sup><br>Embedded Tri-mode Etl | hernet MAC           |
|------------------------------|-----------------------------------------------------|----------------------|
|                              | Wrapper                                             | v4.5                 |
|                              |                                                     | EMAC 0 Configuration |
| MDIO Configuration           |                                                     |                      |
| Initial 1000BASE-X PCS/PMA a | nd SGMII settings                                   |                      |
| PHY Reset                    |                                                     |                      |
| PHY AN Enable                |                                                     |                      |
| PHY Isolate                  |                                                     |                      |
| PHY Powerdown                |                                                     |                      |
| PHY Loopback MSB             |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
|                              |                                                     |                      |
| ( D ( 0) )                   | Page 4 of C ( Page )                                | Nauka Guide Council  |

Figure 3-4: MDIO Configuration

#### **MDIO Configuration**

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



# Chapter 4

# **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, 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 below.

#### interface i

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

| ò | <project directory="">/<component name=""><br/>Core release notes file</component></project> |                                                                                                                                |  |
|---|----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|--|
|   |                                                                                              | <component name="">/drivers<br/>Product documentation</component>                                                              |  |
|   |                                                                                              | <component name="">/doc<br/>Product documentation</component>                                                                  |  |
|   |                                                                                              | <component name="">/example design<br/>Verilog or VHDL design files</component>                                                |  |
|   |                                                                                              | <ul> <li>example_design/client</li> <li>Example client loopback logic</li> <li>client/fifo</li> </ul>                          |  |
|   |                                                                                              |                                                                                                                                |  |
|   |                                                                                              | Physical interface description files                                                                                           |  |
|   | ò                                                                                            | <component name="">/implement</component>                                                                                      |  |
|   |                                                                                              | Implementation script files                                                                                                    |  |
|   |                                                                                              | implement/results<br>Results directory, created after implementation scripts are run, and<br>contains implement script results |  |
|   |                                                                                              | <component name="">/simulation<br/>Simulation scripts</component>                                                              |  |
|   |                                                                                              | isimulation/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 project files.

#### Table 4-1: Project Directory

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

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 4-2: Component Name Directory

| Name                                                          | Description             |  |
|---------------------------------------------------------------|-------------------------|--|
| <project_dir>/<component_name></component_name></project_dir> |                         |  |
| v4_emac_release_notes.txt                                     | Core release notes file |  |
| Back to Top                                                   |                         |  |



### <component name>/drivers

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

#### Table 4-3: Drivers Directory

| Name                                                                           | Description                                                                                                                                                                                       |
|--------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <project_dir>/<co< td=""><td>mponent_name&gt;/drivers</td></co<></project_dir> | mponent_name>/drivers                                                                                                                                                                             |
| v4_emac_l.h                                                                    | Header file providing low-level driver<br>functions for accessing the Ethernet MAC<br>configuration information over the DCR<br>bus. This file is only generated when the<br>DCR Bus is selected. |

Back to Top

## <component name>/doc

This directory contains Ethernet MAC documentation. For details on the Virtex-4 Embedded Tri-Mode Ethernet MAC, see the *Virtex-4 Embedded Tri-Mode Ethernet MAC User Guide*.

#### Table 4-4: Doc Directory

| Name                                                              | Description                                                              |
|-------------------------------------------------------------------|--------------------------------------------------------------------------|
| <project_dir>/<component_name>/doc</component_name></project_dir> |                                                                          |
| v4_emac_ds307.pdf                                                 | Virtex-4 Embedded Tri-Mode Ethernet MAC<br>Wrapper Data Sheet            |
| v4_emac_gsg240.pdf                                                | Virtex-4 Embedded Tri-Mode Ethernet MAC<br>Wrapper Getting Started Guide |

Back to Top

## <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 40 for more information. This directory contains all the design files required for the example design.

#### Table 4-5: Example Design Directory

| Name                                                                         | Description               |  |
|------------------------------------------------------------------------------|---------------------------|--|
| <project_dir>/<component_name>/example_design</component_name></project_dir> |                           |  |
| <component_name>.v[hd]</component_name>                                      | Ethernet MAC wrapper file |  |

| Name                                                        | Description                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <component_name>_block.v[hd]</component_name>               | This lowest level of the hierarchy<br>instantiates specific clock and reset<br>circuitry required by the core. It also<br>instantiates the appropriate physical<br>interface specified when creating the core<br>using the CORE Generator. This layer<br>provides a direct interface to the Client<br>Side Interface of the Ethernet MAC<br>primitive.                        |
| <component_name>_block.ucf</component_name>                 | User constraints file (UCF) for the core and<br>the example design. See Appendix B,<br>"Constraining the Example Design" for<br>additional information.                                                                                                                                                                                                                       |
| <component_name>_example_<br/>design.v[hd]</component_name> | Top-level of the example design containing<br>clocking and reset logic. <sup>1</sup> In addition, the<br>top-level design contains an address swap<br>module; this module receives incoming<br>Ethernet packets, swaps the source and<br>destination address, and sends the<br>Ethernet packet back. All received Ethernet<br>packets are returned to the sending<br>address. |
| <component_name>_<br/>locallink.v[hd]</component_name>      | This hierarchy level instantiates LocalLink<br>FIFOs that translate between the Xilinx<br>standard LocalLink Interface and the<br>Client Side Interface of the Ethernet MAC<br>primitive. This hierarchy level is useful for<br>designers who want to interface to a Xilinx<br>standard LocalLink interface.                                                                  |

Table 4-5: Example Design Directory (Continued)

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.



## 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 4-6: Client Directory

| Name                                                                                                  | Description                                                                                                   |
|-------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| <project_dir>/<component_n< td=""><td>ame&gt;/example_design/client</td></component_n<></project_dir> | ame>/example_design/client                                                                                    |
| address_swap_module_[8   16].v[hd]                                                                    | The client loopback instances this to swap<br>the source and destination addresses of the<br>incoming frames. |

Back to Top

## client/fifo

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

| Table 4-7: | <b>FIFO D</b> | irectory |
|------------|---------------|----------|
|------------|---------------|----------|

| Name                                                                                     | Description                                                                                                                                                                                                                                                                                                  |
|------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <project_dir>/<component_name>/example_design/client/fifo</component_name></project_dir> |                                                                                                                                                                                                                                                                                                              |
| eth_fifo_[8   16].v[hd]                                                                  | FIFO top level. This instantiates the<br>transmit and receive client FIFOs and ties<br>the LocalLink client interface signals<br>together to provide a loopback path from<br>the receiver to the transmitter. For more<br>information on the FIFO see "10 Mbps, 100<br>Mbps, 1 Gbps Ethernet FIFO," page 41. |
| <pre>tx_client_fifo_[8   16].v[hd]</pre>                                                 | Transmit client FIFO. The FIFO receives<br>packets from the client over the LocalLink<br>interface, stores the packet in entirety, and<br>forwards the packet to the MAC.                                                                                                                                    |
| <pre>rx_client_fifo_[8   16].v[hd]</pre>                                                 | Receive client FIFO. The receive FIFO<br>stores the received frame in entirety and<br>forwards the frame to the client after<br>completely verifying the frame's integrity.                                                                                                                                  |

## 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 depending on the physical interface selected.

Table 4-8: Physical Directory

| Name                                                                                                     | Description                                                                                                                                                              |
|----------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <project_dir>/<component_na< td=""><td>me&gt;/example_design/physical</td></component_na<></project_dir> | me>/example_design/physical                                                                                                                                              |
| gmii_if.v[hd]                                                                                            | Delivered if GMII is selected on one or<br>both Ethernet MACs without the<br>Advanced Clocking option (Byte PHY).                                                        |
| gmii_byte_phy_if.v[hd]                                                                                   | Delivered if GMII is selected on one or<br>both Ethernet MACs with the Byte PHY<br>Advanced Clocking option.                                                             |
| <pre>mii_if.v[hd]</pre>                                                                                  | Delivered if MII is selected on one or both of the Ethernet MACs.                                                                                                        |
| rgmii_if.v[hd]                                                                                           | Delivered if RGMII version 1.3 is selected<br>on one or both of the Ethernet MACs.                                                                                       |
| rgmii_v2_0_if.v[hd]                                                                                      | Delivered if RGMII version 2.0 is selected<br>on one or both of the Ethernet MACs.                                                                                       |
| gt11_dual_1000X.v[hd]<br>gt11_to_gt_rxclkcorcnt_shim.v[hd]<br>cal_block_v1_4_1.v[hd]                     | If SGMII or 1000Base-X PCS/PMA or<br>SGMII interface is selected for one or both<br>Ethernet MACs, these files collectively<br>interface MGTs to the physical interface. |

Back to Top

### <component name>/implement

The implement directory contains the core implementation script files.

Table 4-9: Implement Directory

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

## 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 4-10: Results Directory

| Name                                                                                            | Description                                                                            |
|-------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| <project_dir>/<component< td=""><td>_name&gt;/implement/results</td></component<></project_dir> | _name>/implement/results                                                               |
| routed.v[hd]                                                                                    | Back-annotated SimPrim-based VHDL or<br>Verilog design. Used for timing<br>simulation. |
| routed.sdf                                                                                      | Timing information for simulation                                                      |
| Dealers Tex                                                                                     |                                                                                        |

Back to Top

## <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.

| Name                                                                                   | Description                                                                                                                                                                                                                                            |  |  |
|----------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
| <project_dir>/<compos< td=""><td>nent_name&gt;/simulation</td></compos<></project_dir> | nent_name>/simulation                                                                                                                                                                                                                                  |  |  |
| demo_tb.v[hd]                                                                          | VHDL or Verilog demonstration test bench for the Ethernet MAC wrapper                                                                                                                                                                                  |  |  |
| configuration_tb.v[hd]                                                                 | Configuration test bench is instantiated i<br>demo_tb.v[hd]. It provides stimuli to<br>configure the Ethernet MACs via the<br>selected management interface.                                                                                           |  |  |
| emac0_phy_tb.v[hd]                                                                     | Physical interface test bench for EMAC0.<br>This stimulates the receiver ports and<br>monitors the transmitter ports of the<br>EMAC0 physical interface. This is<br>instantiated in demo_tb.v[hd] and is only<br>present when EMAC0 has been selected. |  |  |
| emac1_phy_tb.v[hd]                                                                     | Physical interface test bench for EMAC1.<br>This stimulates the receiver ports and<br>monitors the transmitter ports of the<br>EMAC1 physical interface. This is<br>instantiated in demo_tb.v[hd] and is only<br>present when EMAC1 has been selected. |  |  |

Table 4-11: Simulation Directory

## simulation/functional

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

Table 4-12: Functional Directory

| Name                                                                                                  | Description                                                                                                                                                                                                                                                                                                |  |  |
|-------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
| <project_dir>/<component_n< td=""><td>ame&gt;/simulation/functional</td></component_n<></project_dir> | ame>/simulation/functional                                                                                                                                                                                                                                                                                 |  |  |
| simulate_mti.do                                                                                       | <ul> <li>ModelSim macro file that compiles the example design sources and the structural simulation model then runs the functional simulation to completion.</li> <li>ModelSim macro file that opens a wave window and adds interesting signals to it. It is called by the simulate_mti.do file</li> </ul> |  |  |
| wave_mti.do                                                                                           |                                                                                                                                                                                                                                                                                                            |  |  |
| simulate_ncsim.sh                                                                                     | IUS script file that compiles the example<br>design sources and the structural<br>simulation model and then runs the<br>functional simulation to completion.                                                                                                                                               |  |  |
| wave_ncsim.sv                                                                                         | IUS macro file that opens a wave window and adds interesting signals to it.                                                                                                                                                                                                                                |  |  |

Back to Top

## simulation/timing

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

Table 4-13: TIming Directory

| Name                                                                                           | Description                                                                                                                                    |  |  |
|------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
| <project_dir>/<component< td=""><td>name&gt;/simulation/timing</td></component<></project_dir> | name>/simulation/timing                                                                                                                        |  |  |
| simulate_mti.do                                                                                | ModelSim macro file that compiles the<br>VHDL or Verilog timing model and demo<br>test bench then runs the timing simulation<br>to completion. |  |  |
| wave_mti.do                                                                                    | ModelSim macro file that opens a wave<br>window and adds interesting signals to it.<br>It is called by the simulate_mti.do macro<br>file.      |  |  |
| simulate_ncsim.sh                                                                              | IUS script file that compiles the VHDL or<br>Verilog timing model and demo test bench<br>and then runs the timing simulation to<br>completion. |  |  |
| wave_ncsim.sv                                                                                  | IUS macro file that opens a wave window and adds interesting signals to it.                                                                    |  |  |



# **Implementation and Test Scripts**

## 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.

#### UNIX

<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 IUS

#### 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 IUS

#### 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 4-1: HDL Example Design

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

#### VHDL

```
<project_dir>/<component_name>/example_design/
```



#### Verilog

```
<project_dir>/<component_name>/example_design/
```

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

```
<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

```
<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

The 10 Mbps, 100 Mbps, 1 Gbps Ethernet FIFO contains an instance of tx\_client\_fifo to connect to the Ethernet MAC client side transmitter interface, and an instance of the rx\_client\_fifo to connect to the Ethernet MAC client receiver interface. Both transmit and receive FIFO components implement a LocalLink user interface, through which the frame data can be read/written.

Figure 4-2 illustrates a straightforward frame transfer across the LocalLink interface. For more information about the FIFO, see Appendix A, "Using the Client Side FIFO."



Figure 4-2: Frame Transfer across LocalLink Interface

#### 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



Figure 4-3: Modification of Frame Data by 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 4-3 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 flipflops.
- 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 <u>Answer Record 22477</u> for information about downloading the design files which include the *Calibration Block User Guide*.

# **Demonstration Test Bench**

# **Test Bench Functionality**



Figure 4-4: Demonstration Test Bench

The demonstration test bench is defined in the following files:



#### 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

```
<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, setting up the MDC clock frequency, disabling auto-negotiation in SGMII and 1000Base-X PCS/PMA modes, and disabling flow control.
- 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. The user 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.



# Appendix A

# 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.





## **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

Figure A-3 illustrates frame transfer of a 5-byte frame, where both the src\_rdy\_n and dst\_rdy\_n signals are used to control the flow of data across the interface.



Figure A-3: Frame Transfer with Flow Control

# **Receive FIFO Operation**

The receive FIFO takes data from the client interface of the Ethernet MAC core and converts it into LocalLink format. See the *Virtex-4 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 | Э |
|---------------------------------------------|---|
|---------------------------------------------|---|

| Signal      | Direction | Clock<br>Domain | Description                        |
|-------------|-----------|-----------------|------------------------------------|
| rx_ll_clock | Input     | N/A             | Read clock for LocalLink interface |
| rx_ll_reset | Input     | rx_ll_clock     | Synchronous reset                  |

| Signal              | Direction | Clock<br>Domain | Description                                                           |  |
|---------------------|-----------|-----------------|-----------------------------------------------------------------------|--|
| rx_ll_data_out[7:0] | Output    | rx_ll_clock     | Data read from FIFO                                                   |  |
| rx_ll_sof_out_n     | Output    | rx_ll_clock     | Start of frame indicator                                              |  |
| rx_ll_eof_out_n     | Output    | rx_ll_clock     | End of frame indicator                                                |  |
| rx_ll_src_rdy_out_n | Output    | rx_ll_clock     | Source ready indicator                                                |  |
| rx_ll_dst_rdy_in_n  | Input     | rx_ll_clock     | Destination ready indicator                                           |  |
| rx_fifo_status[3:0] | Output    | rx_ll_clock     | FIFO memory status                                                    |  |
|                     |           |                 | FIFO occupancy indication in<br>units of 256 bytes (rounded<br>down). |  |
|                     |           |                 | A value of 1 indicates FIFO space between 256-511 bytes.              |  |
|                     |           |                 | A value of 2 indicates FIFO space between 512-767 bytes and so on.    |  |

Table A-1: Receive FIFO LocalLink Interface

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

The FIFO status signal indicates the occupancy of the FIFO.

## **Transmit FIFO Operation**

The transmit FIFO accepts frames 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.

| Signal             | Signal Direction |             | Signal Direction Clock Domain        |  | Description |  |  |
|--------------------|------------------|-------------|--------------------------------------|--|-------------|--|--|
| tx_ll_clock        | Input            | N/A         | Write clock for LocalLink interface  |  |             |  |  |
| tx_ll_reset        | Input            | tx_ll_clock | Synchronous reset                    |  |             |  |  |
| tx_ll_data_in[7:0] | Input            | tx_ll_clock | Write data to be sent to transmitter |  |             |  |  |
| tx_ll_sof_in_n     | Input            | tx_ll_clock | Start of frame indicator             |  |             |  |  |

Table A-2: Transmit FIFO LocalLink Interface

| Signal              | Direction | Clock<br>Domain | Description                                                        |  |
|---------------------|-----------|-----------------|--------------------------------------------------------------------|--|
| tx_ll_eof_in_n      | Input     | tx_ll_clock     | End of frame indicator                                             |  |
| tx_ll_src_rdy_in_n  | Input     | tx_ll_clock     | Source ready indicator                                             |  |
| tx_ll_dst_rdy_out_n | Output    | tx_ll_clock     | Destination ready indicator                                        |  |
| tx_fifo_status[3:0] | Output    | tx_ll_clock     | FIFO occupancy indication in units of 256 bytes (rounded down).    |  |
|                     |           |                 | A value of 1 indicates FIFO space between 256-511 bytes.           |  |
|                     |           |                 | A value of 2 indicates FIFO space between 512-767 bytes and so on. |  |

| Table A-2: Transmit FIFO LocalLink Interface | Table A-2: | Transmit FIFO LocalLink Interface |
|----------------------------------------------|------------|-----------------------------------|
|----------------------------------------------|------------|-----------------------------------|

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 4vfx60ff672-10 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 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.

```
# GMII spec: 2ns setup time, 0ns hold time
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 -6 ns VALID 2 ns BEFORE "GMII\_RX\_CLK\_#";

The GMII design uses IDELAY components on the receiver clock. A fixed tap delay is applied to delay the clock so the data is correctly sampled by the gmii\_rx\_clk clock at the IOB flip-flop, meeting GMII setup and hold timing.

The clock tree has intrinsically greater delay than the data. By delaying the clock more with an IDELAY, the data is aligned with the previous clock edge, causing a 1 period (8 ns) shift in the timing window. The new setup requirement becomes 2 ns setup - 8 ns period shift = -6 ns as the new OFFSET requirement. The timing tool compares the data to the correct clock edge.

IDELAY components are also supplied on the data and control signals. This allows delaying either the data or the clock to meet setup and hold time. In the example design, only the clock is delayed, and the IOBDELAY\_VALUE is set to 0 on everything but the clock.

The following constraints show an example of setting the delay value for one of these IDELAY components. All bits can be adjusted individually, if desired, to compensate for any PCB routing skew. The IOBDELAY\_TYPE specifies this delay is constant and will not be changed during operation.

```
# IDELAY on clock path to align it with the data
INST "*gmii_rx_clk_?_delay" IOBDELAY_TYPE = FIXED;
INST "*gmii_rx_clk_?_delay" IOBDELAY_VALUE = 23;
```

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

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

```
# IDELAYCTRL location - must be same clock region as receiver IOs
# See Virtex 4 Users Guide for more information
INST "*dlyctrl_#" LOC = "IDELAYCTRL_X0Y6";
```



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 10ns 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_clk#" 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 %;
```

#### 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 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 125 MHz for 1 Gbps 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 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 receive side logic. The clock should be constrained to 125 MHz for 1 Gbps 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 PHY Clock

The tx\_gmii\_mii\_clk\_in\_#\_i signal is routed to the CLIENTEMAC #TXGMIIMIICLKIN port of the Ethernet MAC along with the RGMII transmitter interface logic. The 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 PHY Clock

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

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

#### 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.



#### **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_tx_clk0" = PERIOD "clk_phy_tx_clk0" 7200 ps HIGH 50 %;
```

## MII Constraints

#### I/O Constraints

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

| INST  | "mii_txd_# "         | TNM = "sig_mii_tx_#";                                           |
|-------|----------------------|-----------------------------------------------------------------|
| INST  | "mii_tx_en_#"        | TNM = "sig_mii_tx_#";                                           |
| INST  | "mii_tx_er_#"        | TNM = "sig_mii_tx_#";                                           |
|       |                      |                                                                 |
| INST  | "mii_rxd_# "         | TNM = "sig_mii_rx_#";                                           |
| INST  | "mii_rx_dv_#"        | TNM = "sig_mii_rx_#";                                           |
| INST  | "mii_rx_er_#"        | TNM = "sig_mii_rx_#";                                           |
|       |                      |                                                                 |
| # usi | ng recommended budg  | get from the clause 22                                          |
| TIME  | GRP "sig_mii_rx_#" ( | <pre>DFFSET = IN 10 ns VALID 20 ns BEFORE "mii_rx_clk_#";</pre> |
| TIME  | GRP "sig_mii_tx_#" ( | <pre>DFFSET = OUT 15 ns AFTER "mii_tx_clk_#";</pre>             |

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\_gmii\_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_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 %;
```

#### 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\_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_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;
                                              MAXDELAY = 6000 ps;
NET "*v4_emac_ll*rx_data_valid_*_i"
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 \text{ ps};
NET "*v4_emac_ll*tx_data_*_i"
                                              MAXDELAY = 6000 ps;
NET "*v4_emac_ll*tx_data_valid_*_i"
                                              MAXDELAY = 6000 ps;
```



## 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 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

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  | "rg | ymii_r: | xd_#    | "      |   | T  | NM | =  | "rgmii_ | _r2 | x_#' | ';     |         |         |      |
|-------|-----|---------|---------|--------|---|----|----|----|---------|-----|------|--------|---------|---------|------|
| INST  | "rg | gmii_r: | x_ctl_# | n      |   | T  | NM | =  | "rgmii_ | _rz | x_#' | ';     |         |         |      |
|       |     |         |         |        |   |    |    |    |         |     |      |        |         |         |      |
| TIMEC | BRP | "rgmi   | i_rx_#" | OFFSET | = | IN | -7 | ns | VALID   | 2   | ns   | BEFORE | "RGMII_ | _RXC_#" | LOW; |

The RGMII design uses IDELAY components on the clock. A fixed tap delay is applied to delay the clock so that the data is correctly sampled by the rgmii\_rxc clock at the IOB IDDR registers, meeting RGMII setup and hold timing.

The clock tree has intrinsically greater delay than the data. By delaying the clock more with an IDELAY, the data is aligned with the previous clock edge, causing a 1 period (8 ns) shift in the timing window. The new setup requirement becomes 1 ns setup - 8 ns period shift = -7 ns as the new OFFSET requirement. The timing tool compares the data to the correct clock edge.

IDELAY components are also supplied on the data and control signals. This allows delaying either the data or the clock to meet setup and hold time. In the example design, only the clock is delayed, and the IOBDELAY\_VALUE is set to 0 on everything but the clock.

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

```
# IDELAY on data to align it with receive clock
INST "*rgmii_rx_clk_?_delay" IOBDELAY_TYPE = FIXED;
```

```
INST "*rgmii_rx_clk_?_delay" IOBDELAY_VALUE = 36;
```

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

When IDELAY or IODELAY primitives are instantiated with a fixed delay attribute, an IDELAYCTRL component must be also instantiated to continuously calibrate the individual input delay elements. The IDELAYCTRL module requires a reference clock, which is assumed to be an input to the example design delivered by CORE Generator. The most efficient way to use the IDELAYCTRL module is to lock the placement of the instance to the clock region of the device where the IDELAY/IODELAY components are placed. An

example LOC constraint for the IDELAYCTRL module is shown below. If your pins are in more than 1 clock region, you need an IDELAYCTRL for each region. See the *Virtex-4 User Guide* and code comments for more information.

```
# IDELAYCTRL locations - One must be in the same clock region as receiver IOs
# dlyctrl_tx needs to be in same clock region as transmitter rgmii_txc_0 signal
INST "*dlyctrl_#" LOC = "IDELAYCTRL_X0Y6";
INST "*dlyctrl_tx" LOC = "IDELAYCTRL_X0Y0";
```

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 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 = 8;
```

#### **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 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 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_clk#" 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 "rgmii_rxc_0_i" TNM_NET = "clk_phy_rx_clk0";
TIMESPEC "TS_phy_rx_clk0" = PERIOD "clk_phy_rx_clk0" 7200 ps HIGH 50 %;
NET "rgmii_rxc_1_i" TNM_NET = "clk_phy_rx_clk1";
TIMESPEC "TS_phy_rx_clk1" = PERIOD "clk_phy_rx_clk1" 7200 ps HIGH 50 %;
```

#### 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 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 125 MHz for 1 Gbps 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 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 receive side logic. The clock should be constrained to 125 MHz for 1 Gbps 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 PHY Clock

The tx\_gmii\_mii\_clk\_in\_#\_i signal is routed to the CLIENTEMAC #TXGMIIMIICLKIN port of the Ethernet MAC along with the RGMII transmitter interface logic. The 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 PHY Clock

The rgmii\_rxc\_#\_i signal is routed to the PHYEMAC #RXCLK port of the Ethernet MAC along with the RGMII receiver interface logic. The 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 %;
```

## 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 %;
```

#### 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 %;
```

### SGMII Constraints

If an SGMII is implemented, the MGT constraints described in "1000Base-X PCS/PMA (8bit 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 Gpbs, the following constraint should be used to constrain the client interface clocks:

```
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 %;
```

#### 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 Gpbs, the following constraint should be used to constrain the client interface clocks:



```
NET "tx_client_clk_in_1_i" TNM_NET = "clk_client_tx_clk1";
TIMESPEC "TS_client_tx_clk1" = PERIOD "clk_client_tx_clk1" 7200 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

```
NET "*RXRECCLK1_0" TNM_NET = "clk_rec_clk0";
TIMESPEC "TS_rec_clk0" = PERIOD "clk_client_rec_clk0" 7200 ps HIGH 50 %;
NET "*clock_correction_A/wr_addr_gray<?>" MAXDELAY = 7 ns;
INST "*clock_correction_A/rd_wr_addr_gray*" TNM = "rx_graycode_A";
INST "*clock_correction_A/rd_occupancy*" TNM = "rx_binary_A";
TIMESPEC "ts_rx_meta_protect_A" = FROM "rx_graycode_A" TO "rx_binary_A" 5 ns;
```

#### EMAC1

```
NET "*RXRECCLK1_1" TNM_NET = "clk_rec_clk1";
TIMESPEC "TS_rec_clk1" = PERIOD "clk_client_rec_clk1" 7200 ps HIGH 50 %;
NET "*clock_correction_B/wr_addr_gray<?>" MAXDELAY = 7 ns;
INST "*clock_correction_B/rd_wr_addr_gray*" TNM = "rx_graycode_B";
INST "*clock_correction_B/rd_occupancy*" TNM = "rx_binary_B";
TIMESPEC "ts_rx_meta_protect_B" = FROM "rx_graycode_B" TO "rx_binary_B" 5 ns;
```

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

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

```
CONFIG STEPPING = "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 PHYEMAC*n*GTXCLK and constrained to operate at 125 MHz to improve clock resource usage.

```
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

```
# Tx client FIF0:
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_txfer_tog_sync" TNM = "tx_stable";
INST "*tx_fifo_i?wr_addr_diff*" TNM = "tx_stable";
INST "*tx_fifo_i?wr_addr_diff*" TNM = "tx_stable";
```

# 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.

# 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";
```



# Appendix C

# SGMII / Dynamic Standards Switching

#### SGMII Capabilities

The Virtex-4 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, 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 RocketIOs. 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) 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 User Guides).



Figure C-1: SGMII Implementation: Separate Clock Sources

#### Analysis

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

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

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

- At 1 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 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 RocketIOs 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 refer to the RocketIO User Guides).

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



## The RocketIO Rx Elastic Buffer

The Elastic Buffer in the RocketIO 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.



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

| Standard/Speed           | Maximum Frame Size |
|--------------------------|--------------------|
| 1000BASE-X (1 Gbps only) | 280000             |
| SGMII (1 Gbps)           | 280000             |
| SGMII (100 Mbps)         | 28000              |
| SGMII (10 Mbps)          | 2800               |