

LogiCORE™ IP Ethernet 1000BASE-X PCS/PMA or SGMII v9.1

Getting Started Guide

UG145 March 24, 2008





Xilinx is disclosing this Document and Intellectual Property (hereinafter “the Design”) to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the Design 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 the Design 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 Design; 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 Design. Xilinx reserves the right to make changes, at any time, to the Design 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 Design.

THE DESIGN 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 DESIGN, 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 DESIGN, 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 DESIGN, 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 DESIGN. 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 DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design 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 Design in such High-Risk Applications is fully at your risk.

© 2004-2008 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
09/30/04	1.0	Initial Xilinx release.
04/28/05	2.0	Updated to version 6.0 of the core, and Xilinx tools v7.1i SP2.
01/18/06	3.0	Updated to version 7.0 of the core, Xilinx tools v8.1i; updated Licensing chapter.
07/13/06	4.0	Updated to version 7.1 of the core, Xilinx tools v8.2i.
10/23/06	5.0	Updated to core version 8.0, support for Virtex-5 LXT and Spartan-3 A device families.
02/15/07	6.0	Updated to core version 8.1, Xilinx tools v9.1i.

Date	Version	Revision
08/08/07	7.0	Updated to core version 9.0, Xilinx tools v9.2i, Cadence IUS to v5.8.
03/24/08	8.0	Updated to core version 9.1, Xilinx tools v10.1.

Table of Contents

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

Preface: About This Guide

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

Chapter 1: Introduction

System Requirements	11
About the Core	11
Designs Using RocketIO Transceivers	11
Recommended Design Experience	12
Additional Core Resources	12
Technical Support	12
Feedback	12
Ethernet 1000BASE-X PCS/PMA or SGMII Core	12
Document	13

Chapter 2: Licensing the Core

Before you Begin	15
License Options	15
Simulation Only	15
Full	15
Obtaining Your License	16
Installing the License File	16

Chapter 3: Quick Start Example Design

Overview	17
Generating the Core	18
Implementing the Example Design	19
Simulating the Example Design	19
Setting up for Simulation	19
Functional Simulation	20
Timing Simulation	20
What's Next?	20

Chapter 4: Detailed Example Design

Directory and File Contents	24
<project directory>	24
<project directory>/<component name>	24
<component name>/doc	25
<component name>/example design	25
<component name>/implement	26
implement/results	26
<component name>/simulation	27
simulation/functional	27
simulation/timing	28
Implementation Scripts	29
Simulation Scripts	29
Functional Simulation	29
Timing Simulation	30
Example Design for 1000BASE-X Using RocketIO Transceivers	30
Top-Level Example Design HDL	31
Block Level HDL	31
Transceiver	32
Transmitter Elastic Buffer	34
Demonstration Test Bench	35
Customizing the Test Bench	37
Core Example Design for 1000BASE-X with Ten-Bit Interface	38
Top-Level Example Design HDL	38
Block Level HDL	39
Transmitter Elastic Buffer	39
Demonstration Test Bench	40
Customizing the Test Bench	42
SGMII Example Design / Dynamic Switching Example Design	
Using a RocketIO Transceiver	43
Top-Level Example Design HDL	43
Block Level HDL	44
Transceiver	45
Receiver Elastic Buffer	47
SGMII Adaptation Module	47
Demonstration Test Bench	48
Customizing the Test Bench	49
SGMII Example Design / Dynamic Switching Example Design with Ten-Bit Interface	51
Top-Level Example Design HDL	51
Block Level HDL	52
SGMII Adaptation Module	52
Demonstration Test Bench	53
Customizing the Test Bench	54

Appendix: SGMII Adaptation Module

Introduction	57
File Structure and Functional Description	58
SGMII Adaptation Module Top Level	58
Clock Generation	58

Johnson Counter.....	60
Transmitter Rate Adaptation Module.....	61
Receiver Rate Adaptation Module	62

Schedule of Figures

Chapter 3: Quick Start Example Design

<i>Figure 3-1: Ethernet 1000BASE-X PCS/PMA Example Design and Test Bench</i>	17
<i>Figure 3-2: Core Customization Screen</i>	18

Chapter 4: Detailed Example Design

<i>Figure 4-1: Example Design HDL for the Ethernet 1000BASE-X PCS/PMA using a RocketIO Transceiver</i>	30
<i>Figure 4-2: Demonstration Test Bench Using RocketIO</i>	35
<i>Figure 4-3: Example Design HDL for the Ethernet 1000BASE-X PCS with TBI</i>	38
<i>Figure 4-4: Demonstration Test Bench for the Ethernet 1000BASE-X PCS with TBI</i>	40
<i>Figure 4-5: Example Design HDL for the Ethernet 1000BASE-X PCS/PMA or SGMII Core in SGMII mode using a RocketIO Transceiver</i>	43
<i>Figure 4-6: Demonstration Test Bench for the Ethernet 1000BASE-X PCS/PMA or SGMII Core in SGMII Mode using RocketIO Transceivers</i>	48
<i>Figure 4-7: Example Design HDL for the Ethernet 1000BASE-X PCS/PMA or SGMII Core in SGMII mode with TBI</i>	51
<i>Figure 4-8: Demonstration Test Bench for the Ethernet 1000BASE-X PCS/PMA or SGMII Core in SGMII Mode with TBI</i>	53

Appendix: SGMII Adaptation Module

<i>Figure 5-1: SGMII Adaptation Module</i>	58
<i>Figure 5-2: Clock Generator Output Clocks and Clock Enable</i>	60
<i>Figure 5-3: Transmitter Rate Adaptation Module Data Sampling</i>	62
<i>Figure 5-4: Receiver Rate Adaptation Module Data Sampling</i>	63

About This Guide

The *LogiCORE™ IP Ethernet 1000Base-X PCS/PMA or SGMII Getting Started Guide* provides information about generating an Ethernet 1000BASE-X PCS/PMA core, customizing and simulating the core using the provided example designs, and running the design files through implementation using the Xilinx tools.

Guide Contents

The following chapters are included in this guide:

- [Preface, “About this Guide”](#) introduces the organization and purpose of the Getting Started Guide and the conventions used in the guide.
- [Chapter 1, “Introduction”](#) describes the core and related information, including system requirements, recommended design experience, technical support, and submitting feedback to Xilinx.
- [Chapter 2, “Licensing the Core”](#) describes the available license options for the core and how to obtain them.
- [Chapter 3, “Quick Start Example Design”](#) provides instructions to quickly generate the core and run the example design through implementation and simulation using the default settings.
- [Chapter 4, “Detailed Example Design”](#) describes the demonstration test bench in detail and provides directions for how to customize the demonstration test bench for use in an application.

Conventions

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

Typographical

The following typographical conventions are used in this document:

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

Convention	Meaning or Use	Example
<i>Italics</i>	References to other manuals	See the <i>User Guide</i> for details.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are <i>not</i> connected.
Shading	Unsupported or reserved items	This feature is not supported
Brackets < >	User-defined variable.	<project directory>
Square brackets []	Optional entry or parameter, with the exception of bus specifications. For bus specifications, brackets are required, for example bus [7:0] .	ngdbuild [option_name] design_name
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 .	Omitted repetitive material	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Omitted repetitive material	allow block block_name loc1 loc2 ... locn;
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returns 45524943h
	An '_n' means the signal is active low	usr_teof_n is active low

Online Document

The following conventions are used in this document:

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

Introduction

The Ethernet 1000BASE-X PCS/PMA or SGMII core is a fully verified solution that supports Verilog-HDL and VHDL. In addition, the example design in this guide is provided in both Verilog and VHDL formats.

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

System Requirements

Windows

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

Linux

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

Software

- ISE™ 10.1

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

About the Core

The Ethernet 1000BASE-X PCS/PMA or SGMII core is a Xilinx CORE Generator™ system core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see the Ethernet 100BASE-X PCS/PMA [product page](#). For information about licensing options, see [Chapter 2, “Licensing the Core.”](#)

Designs Using RocketIO Transceivers

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

- For Virtex-II Pro and Virtex-4 devices, RocketIO Multi-Gigabit Transceivers (MGT)
- For Virtex-5 LXT and SXT devices, RocketIO GTP transceivers; Virtex-5 FXT devices, RocketIO GTX transceivers

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

Recommended Design Experience

Although the Ethernet 1000BASE-X PCS/PMA or SGMII core is a fully-verified solution, the challenge associated with implementing a complete design varies, depending on the configuration and functionality of the application. For best results, previous experience building high-performance, pipelined FPGA designs using Xilinx implementation software and user constraint files (UCFs) is recommended.

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

Additional Core Resources

For detailed information and updates about the Ethernet 1000BASE-X PCS/PMA or SGMII core, see the following documents, available from the [product page](#).

- *LogiCORE IP 1000BASE-X PCS/PMA or SGMII Data Sheet*
- *LogiCORE IP Ethernet 1000BASE-X PCS/PMA or SGMII Getting Started Guide*

From the document directory after generating the core:

- *LogiCORE IP 1000BASE-X PCS/PMA or SGMII Release Notes*
- *LogiCORE IP Ethernet 1000BASE-X PCS/PMA or SGMII User Guide*

Technical Support

For technical support, see www.support.xilinx.com/. Questions are routed to a team of engineers with expertise using the Ethernet 1000BASE-X PCS/PMA or SGMII core.

Xilinx provides technical support for use of this product as described in the *LogiCORE IP Ethernet 1000BASE-X PCS/PMA or SGMII Getting Started Guide* or *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 Ethernet 1000BASE-X PCS/PMA or SGMII core and the documentation supplied with the core.

Ethernet 1000BASE-X PCS/PMA or SGMII Core

For comments or suggestions about the Ethernet 1000BASE-X PCS/PMA or SGMII core, please submit a WebCase from www.xilinx.com/support/clearxpress/websupport.htm/

Be sure to include the following information:

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

Document

For comments or suggestions about this document, please submit a WebCase from www.xilinx.com/support/clearpress/websupport.htm/

Be sure to include the following information:

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

Licensing the Core

This chapter provides instructions for obtaining a license for the Ethernet 1000BASE-X PCS/PMA or SGMII core, which you must do before using the core in your designs. The Ethernet 1000BASE-X PCS/PMA or SGMII core is provided under the terms of the [Xilinx LogiCORE IP Site License Agreement](#), which conforms to the terms of the [SignOnce](#) IP License standard defined by the Common License Consortium.

Before you Begin

This chapter assumes you have installed the core using either the CORE Generator IP Software Update installer or by performing a manual installation after downloading the core from the web. For information about installing the core, see the Ethernet 100BASE-X PCS/PMA [product page](#).

License Options

The Ethernet 1000BASE-X PCS/PMA or SGMII core provides two licensing options, both free of charge.

Simulation Only

The Simulation Only Evaluation license is provided by default with the Xilinx CORE Generator and requires no electronic license key. This license lets you assess the core functionality with either the provided example design or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a structural model generated by the CORE Generator.) When you open the core from the CORE Generator, a message appears regarding the limitations of the Simulation Only Evaluation license.

Full

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

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

Obtaining Your License

Note: No action is required to obtain the Simulation Only Evaluation license; it is provided by default with the CORE Generator.

Obtaining a Full License

To obtain a Full license, you must register for access to the *lounge*, a secured area of the Ethernet 100BASE-X PCS/PMA [product page](#).

- From the product page, click Register to register and request access to the lounge. Access to the lounge is automatic and granted immediately.
- After you receive confirmation of lounge access, click Access Lounge on the Ethernet 100BASE-X PCS/PMA or SGMII product page and log in.
- Click Access Lounge on the product lounge page and fill out the license request form linked from this location; then click Submit to automatically generate the license. An email containing the license and installation instructions will be sent immediately to the email address you specified.

Installing the License File

The Simulation Only Evaluation license does not require a license file; it is provided with the CORE Generator. If you select the Full license option, an email will be sent to you containing instructions for installing your license file. In addition, the email provides information about advanced licensing options and technical support.

Quick Start Example Design

The quick start steps provided in this chapter let you quickly generate an Ethernet 1000BASE-X PCS/PMA or SGMII core, run the design through implementation with the Xilinx tools, and simulate the design using the provided demonstration test bench. For detailed information about the example design, see [Chapter 4, “Detailed Example Design.”](#)

Overview

The Ethernet 1000BASE-X PCS/PMA or SGMII example design consists of the following:

- Ethernet 1000BASE-X PCS/PMA core netlist
- Example design HDL top-level and associated HDL files
- Demonstration test bench to exercise the example design

The Ethernet 1000BASE-X PCS/PMA or SGMII example design has been tested using Xilinx ISE 10.1, Cadence® IUS v6.1, ModelSim® PE/SE 6.3c and Synopsys® vcs_mxY-2006.06-SP1.

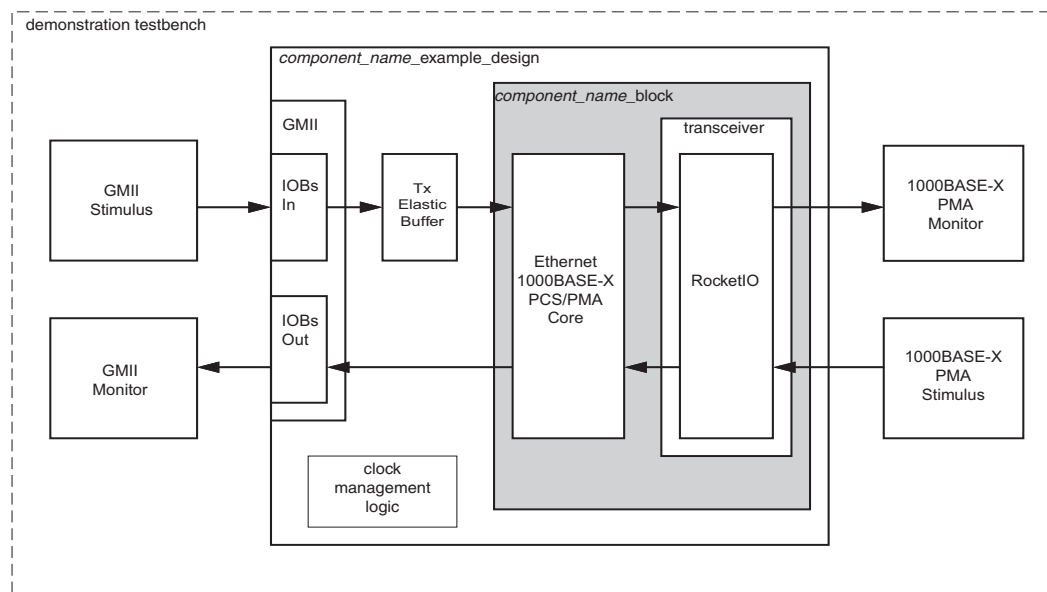


Figure 3-1: Ethernet 1000BASE-X PCS/PMA or SGMII Example Design and Test Bench

Generating the Core

This section provides detailed instructions for generating the Ethernet 1000BASE-X PCS/PMA or SGMII example design core.

To generate the core:

1. Start the CORE Generator tool.
For general help with starting and using CORE Generator on your system, see the documentation supplied with ISE, including the *Core Generator Guide*. These documents can be downloaded from:
www.xilinx.com/support/software_manuals.htm.
2. Create a new project.
3. For project options, select the following:
 - A Virtex™-5 device to generate the default Ethernet 1000BASE-X PCS/PMA or SGMII core.
 - In the Design Entry section, select VHDL or Verilog; then select Other for Vendor.
4. Locate the Ethernet 1000BASE-X PCS/PMA or SGMII core in the taxonomy tree, listed under one of the following:
 - Communications & Networking/Ethernet
 - Communications & Networking/Networking
 - Communications & Networking/Telecommunications
5. Double-click the core. A message may appear to indicate the limitations of the Simulation Only Evaluation license.
6. Click OK; the core customization screen appears.

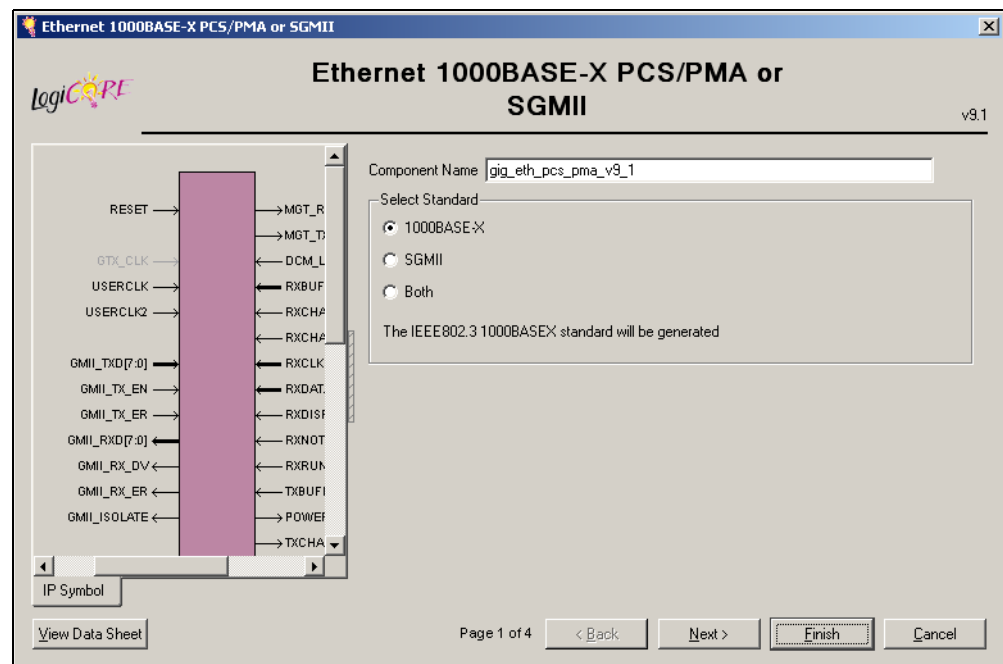


Figure 3-2: Core Customization Screen

7. Enter a core instance name in the Component Name field.

8. Click Finish to generate the core using the default options.

The default core and its supporting files, including the example design, are generated in your project directly. For a detailed description of the design example files and directories, refer to [“Directory and File Contents” in Chapter 4](#).

Implementing the Example Design

Note: Available only with a Full license.

After the core is generated, the netlists and example design can be processed by the Xilinx implementation tools. The generated output files include several scripts to assist you in running the Xilinx software.

To implement the Ethernet 1000BASE-X PCS/PMA or SGMII sample design core:

From the CORE Generator project directory window, type the following:

Linux

```
unix-shell> cd <project_dir>/<component_name>/implement
unix-shell> ./implement.sh
```

Windows

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

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

Simulating the Example Design

Setting up for Simulation

To run the gate-level simulation you must have the Xilinx Simulation Libraries compiled for your system. See the Compiling Xilinx Simulation Libraries (COMPXLIB) in the *Xilinx ISE Synthesis and Verification Design Guide*, and the *Xilinx ISE Software Manuals and Help*.

You can download these documents from:

www.xilinx.com/support/software_manuals.htm.

In addition, use the following guidelines to determine the simulator type required for your design:

Virtex-5 Devices

Virtex-5 device designs incorporating a RocketIO transceiver require either a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator or a SWIFT-compliant simulator.

- For a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator, ModelSim v6.3c is currently supported.
- For a SWIFT-compliant simulator, Cadence IUS v6.1 and Synopsys VCS 2006.06-SP1 are currently supported.

Virtex-4 and Virtex-II Pro Devices

Virtex-4 and Virtex-II Pro device designs incorporating a RocketIO transceiver require a SWIFT-compliant simulator. ModelSim, Cadence IUS, and Synopsys are currently supported using the versions defined above.

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

Functional Simulation

Note: Available for both license types.

This section provides instructions for running a functional simulation of the Ethernet 1000BASE-X PCS/PMA or SGMII core using either VHDL or Verilog. The functional simulation model is provided when the core generated; implementing the core before simulation is not required.

To run a VHDL or Verilog functional simulation of the example design:

1. Open a command prompt or shell, then set the current directory to:
`<project_dir>/<component_name>/simulation/functional/`
2. Launch the simulation script:

```
ModelSim: vsim -do simulate_mti.do
IUS: ./simulate_ncsim.sh
VCS: ./simulate_vcs.sh (Verilog only)
```

The simulation script compiles the functional simulation model, the example design files, the demonstration test bench, and adds relevant signals to a wave window. It then runs the simulation to completion. After completion, you can inspect the simulation transcript and waveform to observe the operation of the core.

Timing Simulation

Note: Available only with a Full license.

This section contains instructions for running a timing simulation of the Ethernet 1000BASE-X PCS/PMA or SGMII core using either VHDL or Verilog. A timing simulation model is generated when run through the Xilinx tools using the implementation script. You must implement the core before attempting to run timing simulation.

To run a VHDL or Verilog timing simulation of the example design:

1. Run the implementation script (see [“Implementing the Example Design,” page 21](#)).
2. Open a command prompt or shell, then set the current directory to:
`<project_dir>/<component_name>/simulation/timing/`
3. Launch the simulation script:

```
ModelSim: vsim -do simulate_mti.do
IUS: ./simulate_ncsim.sh
VCS: ./simulate_vcs.sh (Verilog only)
```

The simulator script compiles the gate-level model and the demonstration test bench, adds relevant signals to a wave window, and then runs the simulation to completion. You can then inspect the simulation transcript and waveform to observe the operation of the core.

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."](#)

To begin using the Ethernet 1000BASE-X PCS/PMA or SGMII core in your own designs, see the *Xilinx Ethernet 1000BASE-X PCS/PMA or SGMII User Guide*.

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.

<project directory>

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

<project directory>/<component name>

Core release notes file

<component name>/doc

Product documentation

<component name>/example design

Verilog and VHDL design files

<component name>/implement

Implementation script files

implement/results

Results directory, created after implementation scripts are run, and contains implement script results

<component name>/simulation

Simulation scripts

simulation/functional

Functional simulation files

simulation/timing

Timing simulation files

Directory and File Contents

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

<project directory>

The project directory contains all the CORE Generator project files.

Table 4-1: Project Directory

Name	Description
<project_dir>	
<component_name>.ngc	Top-level netlist. This is instantiated by the Verilog or VHDL example design.
<component_name>.v[hd]	Verilog or VHDL simulation model; UniSim-based
<component_name>.v{ho eo}	Verilog or VHDL instantiation template for the core
<component_name>.xco	Log file that records the settings used to generate a core. An XCO file is generated by the CORE Generator for each core that it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator.
<component_name>.xcp	Similar to the XCO file except that it does not specify project-specific settings, such as target architecture and output products
<component_name>_flist.txt	List of files delivered with the core

[Back to Top](#)

<project directory>/<component name>

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

Table 4-2: Component Name Directory

Name	Description
<project_dir>/<component_name>	
gig_eth_pcs_pma_readme.txt	Core release notes file

[Back to Top](#)

<component name>/doc

The doc directory contains the PDF documentation provided with the core.

Table 4-3: Doc Directory

Name	Description
<project_dir>/<component_name>/doc	
gig_eth_pcs_pma_ds_264.pdf	<i>Ethernet 1000BASE-X PCS/PMA or SGMII Data Sheet</i>
gig_eth_pcs_pma_gsg145.pdf	<i>Ethernet 1000BASE-X PCS/PMA or SGMII Getting Started Guide</i>
gig_eth_pcs_pma_ug155.pdf	<i>Ethernet 1000BASE-X PCS/PMA or SGMII User Guide</i>

[Back to Top](#)

<component name>/example design

The example design directory contains the example design files provided with the core, and may contain files other than those defined in [Table 4-4](#). For more information, see

- “Example Design for 1000BASE-X Using RocketIO Transceivers,” page 32
- “Core Example Design for 1000BASE-X with Ten-Bit Interface,” page 40
- “SGMII Example Design / Dynamic Switching Example Design Using a RocketIO Transceiver,” page 45
- “SGMII Example Design / Dynamic Switching Example Design with Ten-Bit Interface,” page 53

Table 4-4: Example Design Directory

Name	Description
<project_dir>/<component_name>/example_design	
_example_design.ucf	Example User Constraints File (UCF) provided for the example design
_example_design.v[hd]	Top-level file that allows example design to be implemented in a device as a standalone design.
_block.vhd	A block-level file that is a useful part of example design and should be instantiated in all customer designs.

[Back to Top](#)

<component name>/implement

The implement directory contains the core implementation script files.

Note: This directory is only present with the Full license.

Table 4-5: Implement Directory

Name	Description
<project_dir>/<component_name>/implement	
implement.sh	UNIX shell script that processes the example design through the Xilinx tool flow. See “Implementation Scripts,” page 31 for more information.
implement.bat	Windows batch file that processes the example design through the Xilinx tool flow. See “Implementation Scripts,” page 31 for more information.
xst.prj	XST project file for the example design (VHDL only); it enumerates all of the VHDL files that need to be synthesized.
xst.scr	XST script file for the example design

[Back to Top](#)

implement/results

The results directory is created by the implement script, after which the implement script results are placed in the results directory.

Table 4-6: Results Directory

Name	Description
<project_dir>/<component_name>/implement/results	
routed.v[hd]	Back-annotated SimPrim-based model used for timing simulation
routed.sdf	Timing information for simulation

[Back to Top](#)

<component name>/simulation

The simulation directory and subdirectories that provide the files necessary to test a Verilog or VHDL implementation of the example design. For more information, see:

- “Example Design for 1000BASE-X Using RocketIO Transceivers,” page 32
- “Core Example Design for 1000BASE-X with Ten-Bit Interface,” page 40
- “SGMII Example Design / Dynamic Switching Example Design Using a RocketIO Transceiver,” page 45
- “SGMII Example Design / Dynamic Switching Example Design with Ten-Bit Interface,” page 53

Table 4-7: Simulation Directory

Name	Description
<project_dir>/<component_name>/simulation	
demo_tb.v[hd]	Top-level file of the demonstration test bench for the example design. Instantiates the example design (the Device Under Test (DUT)), generates clocks, resets, and test bench control semaphores.
stimulus_tb.v[hd]	Creates test bench stimulus in the form of four Ethernet frames, which are injected into the DUT. The output from the DUT is also monitored for errors.

[Back to Top](#)

simulation/functional

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

Table 4-8: Functional Directory

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_mti.do	ModelSim macro file that compiles Verilog or VHDL sources and runs the functional simulation to completion.
wave_mti.do	ModelSim macro file that opens a wave window and adds signals of interest to it. It is called by the simulate_mti.do macro file.
simulate_ncsim.sh	IUS script file that compiles the Verilog or VHDL sources and runs the functional simulation to completion.
wave_ncsim.sv	IUS macro file that opens a wave window and adds signals of interest to it. It is called by the simulate_ncsim.sh script file.

Table 4-8: Functional Directory (Continued)

Name	Description
<code>simulate_vcs.sh</code>	VCS script file that compiles the Verilog sources and runs the functional simulation to completion.
<code>vcs_commands.key</code>	This file is sourced by VCS at the start of simulation: it configures the simulator.
<code>vcs_session.tcl</code>	VCS macro file that opens a wave window and adds signals of interest to it. It is called by the <code>simulate_vcs.sh</code> script file

[Back to Top](#)

simulation/timing

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

Note: This directory is only present with the Full license.

Table 4-9: Timing Directory

Name	Description
<project_dir>/<component_name>/simulation/timing	
<code>simulate_mti.do</code>	ModelSim macro file that compiles Verilog or VHDL sources and runs the timing simulation to completion.
<code>wave_mti.do</code>	ModelSim macro file that opens a wave window and adds signals of interest to it. It is called by the <code>simulate_mti.do</code> macro file.
<code>simulate_ncsim.sh</code>	IUS script file that compiles the Verilog or VHDL sources and runs the timing simulation to completion.
<code>wave_ncsim.sv</code>	IUS macro file that opens a wave window and adds signals of interest to it. It is called by the <code>simulate_ncsim.sh</code> script file.
<code>simulate_vcs.sh</code>	VCS script file that compiles the Verilog sources and runs the functional simulation to completion.
<code>vcs_commands.key</code>	This file is sourced by VCS at the start of simulation: it configures the simulator.
<code>vcs_session.tcl</code>	VCS macro file that opens a wave window and adds signals of interest to it. It is called by the <code>simulate_vcs.sh</code> script file

[Back to Top](#)

Implementation Scripts

Note: These scripts are only present with the Full license.

The implementation script is either a shell script or batch file that processes the example design through the Xilinx tool flow. It is located at:

UNIX

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

Windows

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

The implement script performs the following steps:

1. The HDL example design files are synthesized using XST.
2. Ngdbuild is run to consolidate the core netlist and the example design netlist into the NGD file containing the entire design.
3. The design is mapped to the target technology.
4. The design is placed-and-routed on the target device.
5. Static timing analysis is performed on the routed design using `trce`.
6. A bitstream is generated.
7. Netgen runs on the routed design to generate a VHDL or Verilog netlist (as appropriate for the Design Entry project setting) 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, which is created by the implement script:

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

Simulation Scripts

Functional Simulation

The test script is a ModelSim, IUS or VCS macro that automates the simulation of the test bench and is in the following location:

```
<project_dir>/<component_name>/simulation/functional/
```

The test script performs the following tasks:

- Compiles the structural UniSim simulation model
- Compiles HDL example design source code
- Compiles the demonstration test bench
- Starts a simulation of the test bench
- Opens a Wave window and adds signals of interest (`wave_mti.do/wave_ncsim.sv`)
- Runs the simulation to completion

Timing Simulation

Note: This script is only present with the Full license.

The test script is a ModelSim, IUS or VCS macro that automates the simulation of the test bench and is in the following location:

```
<project_dir>/<component_name>/simulation/timing/
```

The test script performs the following tasks:

- Compiles the SimPrim-based gate level netlist simulation model
- Compiles the demonstration test bench
- Starts a simulation of the test bench
- Opens a Wave window and adds signals of interest (`wave_mti.do/wave_ncsim.sv`)
- Runs the simulation to completion

Example Design for 1000BASE-X Using RocketIO Transceivers

Figure 4-1 illustrates the complete example design for the Ethernet 1000BASE-X PCS/PMA using a RocketIO transceiver specific to the target device (Virtex-4, Virtex-II Pro, or Virtex-5).

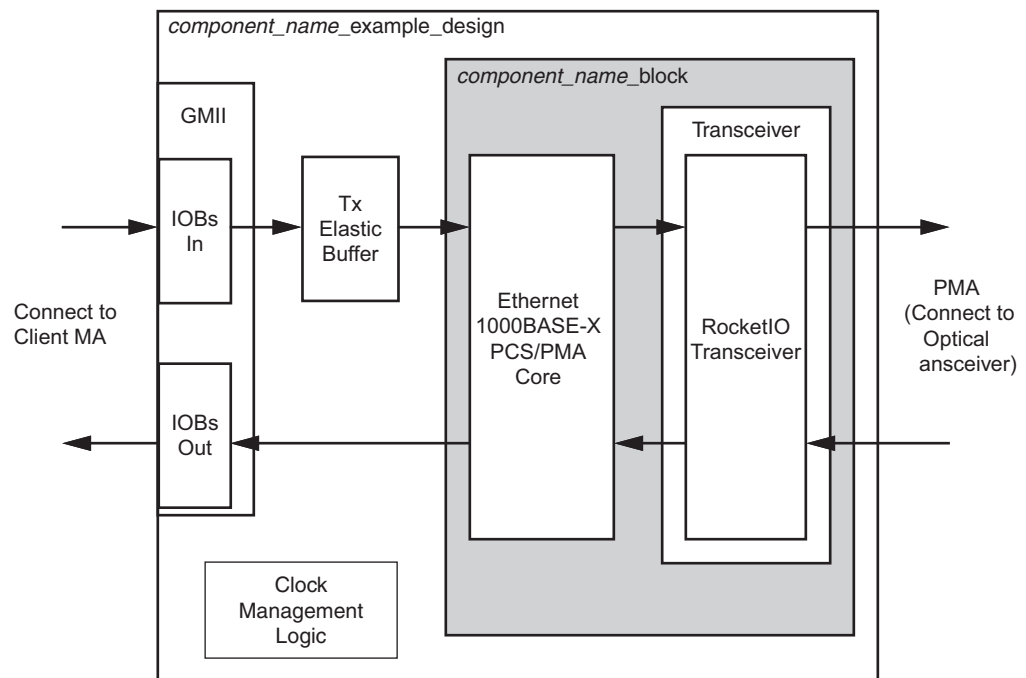


Figure 4-1: Example Design HDL for the Ethernet 1000BASE-X PCS/PMA using a RocketIO Transceiver

Top-Level Example Design HDL

The following files describe the top-level example design for the Ethernet 1000BASE-X PCS/PMA core using a RocketIO transceiver specific to the desired device (Virtex-4 or Virtex-5 FPGA).

VHDL

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

Verilog

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

The example design HDL top level contains the following:

- An instance of the Ethernet 1000BASE-X PCS/PMA block level
- Clock management logic for the core and the RocketIO transceiver, including DCM (if required) and Global Clock Buffer instances
- A transmitter elastic buffer
- GMII interface logic, including IOB instances

The example design HDL top-level connects the GMII of the block level to external IOBs. This configuration allows the functionality of the core to be demonstrated using a simulation package as discussed in this guide. The example design can also be synthesized and, if required, placed on a suitable board and demonstrated in hardware.

Note: In the Virtex-4 and Virtex-5 families, RocketIO transceivers are provided in pairs. When generated with the appropriate options, the example design is capable of connecting two instances of the core to the RocketIO transceiver pair.

Block Level HDL

The following files describe the block-level design for the Ethernet 1000BASE-X PCS/PMA core using a RocketIO transceiver specific to the target device (Virtex-II Pro, Virtex-4, or Virtex-5 FPGA).

VHDL

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

Verilog

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

The block-level HDL contains the following:

- An instance(s) of the Ethernet 1000BASE-X PCS/PMA core
- An instance(s) of a RocketIO transceiver specific to a Virtex-II Pro, Virtex-4, or Virtex-5 device

The block-level HDL connects the PHY side interface of the core to a RocketIO transceiver, as illustrated in [Figure 4-1](#). This is the most useful part of the example design and should be instantiated in all customer designs that use the core.

Note: For Virtex-4 and Virtex-5 families, RocketIO transceivers are provided in pairs. When generated with the respective RocketIO transceiver pair.

Transceiver

A wrapper file for the RocketIO transceiver specific to the target device (Virtex-II Pro, Virtex-4, or Virtex-5 FPGA) is described in the following files:

VHDL

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

Verilog

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

This file instances a RocketIO transceiver and applies Gigabit Ethernet 1000BASE-X attributes to it. This RocketIO transceiver wrapper is instantiated from the block-level HDL file of the example design.

Note: In the Virtex-4 and Virtex-5 families, RocketIO transceivers are provided in pairs. When generated with the appropriate options, the block level is capable of connecting two instances of the core to the RocketIO transceiver pair. When only a single instance of the core is requested, the unused RocketIO transceiver from the pair is still instantiated from within this transceiver wrapper but left unconnected.

Virtex-5 Specific Transceiver Files

RocketIO GTP Transceiver

For Virtex-5 LXT and SXT devices, the transceiver wrapper file directly instantiates RocketIO GTP transceiver wrapper files created from the Virtex-5 RocketIO GTP Wizard. These files tie off (or leaves unconnected) unused I/O for the GTP pair, and apply the 1000BASE-X attributes. The files can be edited /tailored by rerunning the RocketIO GTP Wizard and swapping these files. The files include the following:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtp.vhd  
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtp_tile.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtp.v  
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtp_tile.v
```

To re-run the RocketIO GTP Wizard, a CORE Generator XCO file for the RocketIO GTP Wizard is included. This file defines all the RocketIO GTP Wizard attributes used to generate the files above. See the CORE Generator documentation for further information about XCO files. The XCO file is in the following location:

```
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtp.xco
```

RocketIO GTX Transceiver

For Virtex-5 FXT devices, the transceiver wrapper file directly instantiates RocketIO GTX transceiver wrapper files created from the Virtex-5 RocketIO GTX Transceiver Wizard. These files tie off (or leaves unconnected) unused I/O for the GTX pair, and apply the

1000BASE-X attributes. The files can be edited/tailored by rerunning the RocketIO GTX Transceiver Wizard and swapping these files, which include the following:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtx.vhd  
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtx_tile.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtx.v  
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtx_tile.v
```

In order to re-run the RocketIO GTX Wizard, a CORE Generator XCO file for the RocketIO GTX Wizard has also been included. This file defines all the RocketIO GTX Wizard attributes used to generate the files above. See the CORE Generator documentation for more information about XCO files. The XCO file is in the following location:

```
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtx.xco
```

Virtex-4 Specific Transceiver files

Calibration Blocks

For Virtex-4 FX devices only, Calibration Blocks are required. A Calibration block is connected to both GT11 A and B within the RocketIO transceiver tile. This occurs in the transceiver wrapper file. See [Answer Record 22477](#) for information about downloading the *Calibration Block User Guide*. The Calibration Block is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/  
cal_block_v1_4_1.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/  
cal_block_v1_4_1.v
```

GT11 Reset/Initialization Circuitry

Precise reset/initialization circuitry is required for the GT11 RocketIO transceivers.

The reset circuitry for the RocketIO receiver is illustrated in *Figure 2-18* of the *Virtex-4 FPGA RocketIO Multi-Gigabit Transceiver User Guide* (UG076) and implemented in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/  
gt11_init_rx.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/  
gt11_init_rx.v
```

The reset circuitry for the RocketIO transmitter is illustrated in *Figure 2-13* of the *Virtex-4 RocketIO Multi-Gigabit Transceiver User Guide* (UG076) and implemented in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/
gt11_init_tx.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/
gt11_init_tx.v
```

Both receiver and transmitter reset circuitry entities are instantiated from within the block level of the example design.

Transmitter Elastic Buffer

The Transmitter Elastic Buffer is described in the following files:

VHDL

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

Verilog

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

When the GMII is used externally (as in this example design), the GMII transmit signals (inputs to the core from a remote MAC at the other end of the interface) are synchronous to a clock that is likely to be derived from a different clock source to the core. For this reason, GMII transmit signals must be transferred into the core main clock domain before they can be used by the core and RocketIO transceiver. This is achieved with the Transmitter Elastic Buffer, an asynchronous FIFO implemented in distributed RAM. The operation of the elastic buffer is to attempt to maintain a constant occupancy by inserting or removing any idle sequences. This causes no corruption to the frames of data.

When the GMII is used as an internal interface, it is expected that the entire interface will be synchronous to a single clock domain, and the Transmitter Elastic Buffer should be discarded. See the *Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* for information about connecting the core to an internal GMII or an Ethernet MAC.

Demonstration Test Bench

Figure 4-2 illustrates the demonstration test bench for the Ethernet 1000BASE-X PCS/PMA using a RocketIO transceiver. The demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core.

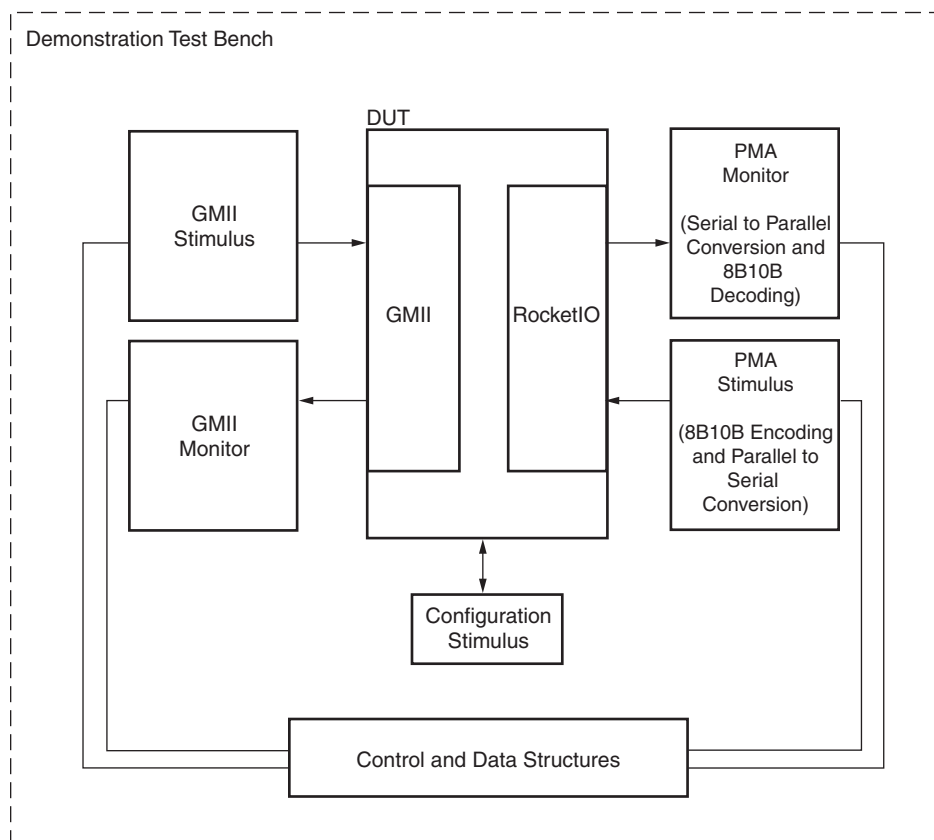


Figure 4-2: Demonstration Test Bench Using RocketIO

The top-level test bench entity instantiates the example design for the core, which is the Device Under Test (DUT). A stimulus block is also instantiated and clocks, resets, and test bench semaphores are created. The following files describe the top level of the demonstration test bench:

VHDL

```
<project_dir>/<component_name>/simulation/demo_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/demo_tb.v
```

The stimulus block entity, instantiated from within the test bench top level, creates the Ethernet stimulus in the form of four Ethernet frames, which are injected into the GMII and PHY interfaces of the DUT. The output from the DUT is also monitored for errors. The following files describe the stimulus block of the demonstration test bench.

VHDL

```
<project_dir>/<component_name>/simulation/stimulus_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/stimulus_tb.v
```

Together, the top-level test bench file and the stimulus block combine to provide the full test bench functionality, described in the sections that follow.

Note: In the Virtex-4 and Virtex-5 families, RocketIO transceivers are provided in pairs. When generated with the appropriate options, the example design is capable of connecting two instances of the core to the RocketIO transceiver pair. When this is the case, two stimulus blocks are instantiated from the top-level test bench to independently exercise both cores.

Core with MDIO Interface

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The Ethernet 1000BASE-X PCS/PMA core is configured through the MDIO interface by injecting a MDIO frame into the example design. This disables Auto-Negotiation (if present) and takes the core out of the Isolate state.
- Four frames are injected into the GMII transmitter by the GMII stimulus block.
 - + the first frame is a minimum length frame
 - + the second frame is a type frame
 - + the third frame is an errored frame
 - + the fourth frame is a padded frame
- The serial data received at the RocketIO transmitter interface is converted to 10-bit parallel data, then 8B10B decoded. The resulting frames are checked by the PMA Monitor against the stimulus frames injected into the GMII transmitter to ensure data integrity.
- The same four frames are generated by the PMA Stimulus block. These are 8B10B encoded, converted to serial data, and injected into the RocketIO transceiver receiver interface.
- Data frames received at the GMII receiver are checked by the GMII Monitor against the stimulus frames injected into the RocketIO transceiver receiver to ensure data integrity.

Core without MDIO Interface

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The Ethernet 1000BASE-X PCS/PMA core is configured using the Configuration Vector to take the core out of the Isolate state.
- Four frames are injected into the GMII transmitter by the GMII stimulus block.
 - + the first frame is a minimum length frame
 - + the second frame is a type frame
 - + the third frame is an errored frame
 - + the fourth frame is a padded frame
- The serial data received at the RocketIO transmitter interface is converted to 10-bit parallel data, then 8B10B decoded. The resultant frames are checked by the PMA Monitor against the stimulus frames injected into the GMII transmitter to ensure data integrity.
- The same four frames are generated by the PMA Stimulus block. These are 8B10B encoded, converted to serial data and injected into the RocketIO receiver interface.
- Data frames received at the GMII receiver are checked by the GMII Monitor against the stimulus frames injected into the RocketIO receiver to ensure data is the same.

Customizing the Test Bench

Changing Frame Data

You can change the contents of the four frames used by the demonstration test bench by changing the *data* and *valid* fields for each frame defined in the stimulus block. New frames can be added by defining a new frame of data. Modified frames are automatically updated in both stimulus and monitor functions.

Changing Frame Error Status

Errors can be inserted into any of the predefined frames in any position by setting the *error* field to '1' in any column of that frame. Injected errors are automatically updated in both stimulus and monitor functions.

Changing the Core Configuration

The configuration of the Ethernet 1000BASE-X PCS/PMA core used in the demonstration test bench can be altered.

Caution! Certain configurations of the core will cause the test bench to fail or to cause processes to run indefinitely. For example, the demonstration test bench will not Auto-Negotiate with the example design. Determine the configurations that can safely be used with the test bench.

When the MDIO interface option is selected, the core can be reconfigured by editing the injected MDIO frame in the demonstration test bench top level. See the *Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* for more information on using the MDIO interface.

When the MDIO interface option is not selected, the core can be reconfigured by modifying the configuration vector directly. See the *Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* for information on using the configuration vector.

Core Example Design for 1000BASE-X with Ten-Bit Interface

Figure 4-3 illustrates the example design for a top-level HDL with a 10-bit interface (TBI).

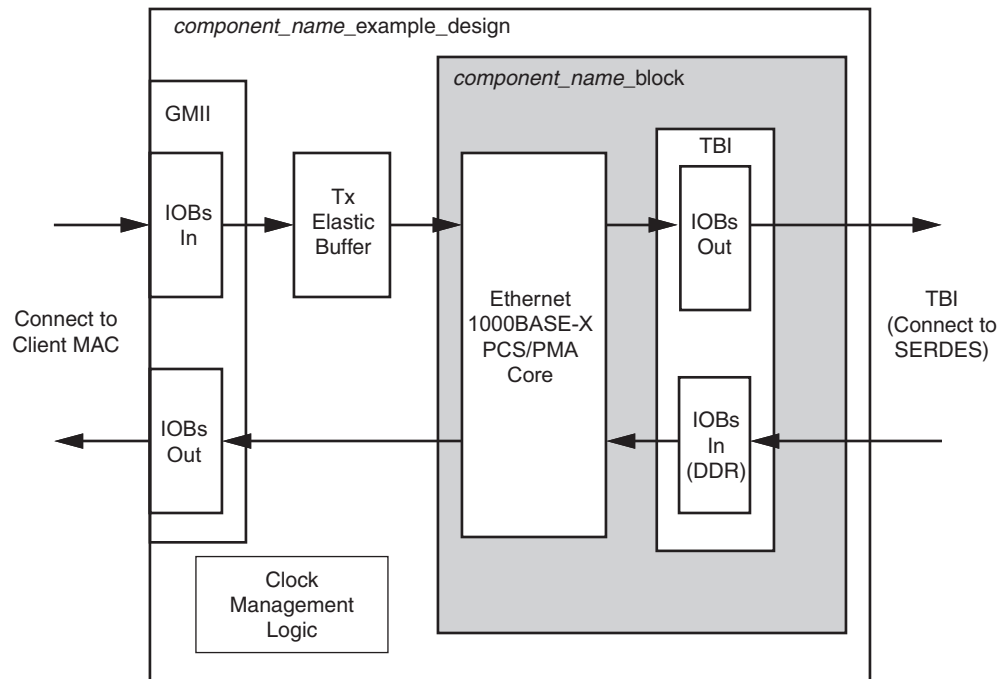


Figure 4-3: Example Design HDL for the Ethernet 1000BASE-X PCS with TBI

Top-Level Example Design HDL

The following files describe the top-level example design for the Ethernet 1000BASE-X PCS/PMA core with TBI:

VHDL

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

Verilog

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

The HDL example design top-level contains the following:

- An instance of the Ethernet 1000BASE-X PCS/PMA block level
- Clock management logic, including DCM and Global Clock Buffer instances, where required
- A transmitter elastic buffer
- GMII interface logic, including IOB and DDR registers instances, where required

The example design HDL top level connects the GMII of the block level to external IOBs. This allows the functionality of the core to be demonstrated using a simulation package as described in this guide. The example design can also be synthesized and placed on a suitable board and demonstrated in hardware, if required.

Block Level HDL

The following files describe the block level design for the Ethernet 1000BASE-X PCS/PMA core with TBI:

VHDL

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

Verilog

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

The block level HDL contains the following:

- An instance of the Ethernet 1000BASE-X PCS/PMA core
- TBI interface logic, including IOB and DDR registers instances, where required

The block-level HDL connects the TBI of the core to external IOBs (the most useful part of the example design) and should be instantiated in all customer designs that use the core.

Transmitter Elastic Buffer

The Transmitter Elastic Buffer is described in the following files:

VHDL

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

Verilog

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

When the GMII is used externally (as in this example design), the GMII transmit signals (inputs to the core from a remote MAC at the other end of the interface) are synchronous to a clock, which is likely to be derived from a different clock source to the core. For this reason, GMII transmit signals must be transferred into the core main clock domain before they can be used by the core. This is achieved with the Transmitter Elastic Buffer, an asynchronous FIFO implemented in distributed RAM. The operation of the elastic buffer is to attempt to maintain a constant occupancy by inserting or removing Idle sequences as necessary. This causes no corruption to the frames of data.

When the GMII is used as an internal interface, it is expected that the entire interface will be synchronous to a single clock domain, and the Transmitter Elastic Buffer should be discarded. See the *Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* for information about connecting the core to an internal GMII (for example, an Ethernet MAC).

Demonstration Test Bench

Figure 4-4 illustrates the demonstration test bench for the Ethernet 1000BASE-X PCS with TBI. The demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself.

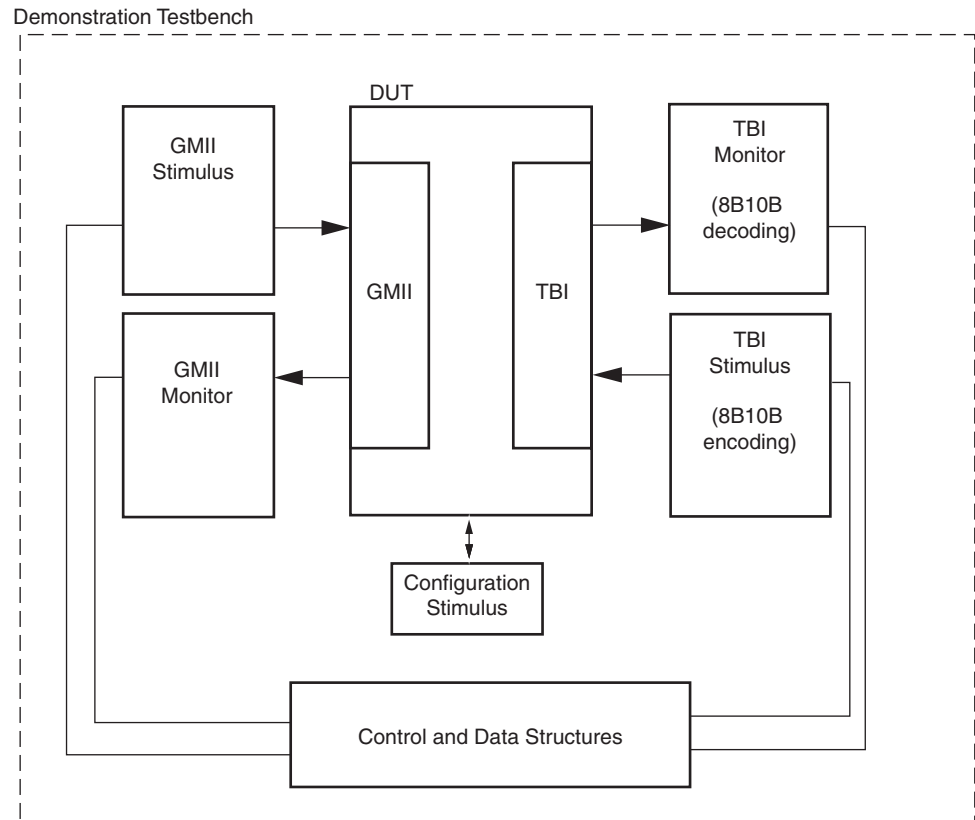


Figure 4-4: Demonstration Test Bench for the Ethernet 1000BASE-X PCS with TBI

The top-level test bench entity instantiates the example design for the core, which is the Device Under Test (DUT). A stimulus block is also instantiated and clocks, resets and test bench semaphores are created. The following files describe the top-level of the demonstration test bench:

VHDL

```
<project_dir>/<component_name>/simulation/demo_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/demo_tb.v
```

The stimulus block entity, instantiated from within the test bench top level, creates the Ethernet stimulus in the form of four Ethernet frames, which are injected into the GMII and PHY interfaces of the DUT. The output from the DUT is also monitored for errors. The following files describe the stimulus block of the demonstration test bench:

VHDL

```
<project_dir>/<component_name>/simulation/stimulus_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/stimulus_tb.v
```

Together, the top-level test bench file and the stimulus block combine to provide the full test bench functionality, described in the sections that follow.

Core with MDIO Interface

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The Ethernet 1000BASE-X PCS/PMA core is configured through the MDIO interface by injecting an MDIO frame into the example design. This disables Auto-Negotiation (if present) and takes the core out of the Isolate state.
- The following frames are injected into the GMII transmitter by the GMII stimulus block:
 - + the first is a minimum-length frame
 - + the second is a type frame
 - + the third is an errored frame
 - + the fourth is a padded frame
- The data received at the TBI transmitter interface is 8B10B decoded. The resulting frames are checked by the TBI Monitor against the stimulus frames injected into the GMII transmitter to ensure data integrity.
- The same four frames are generated by the TBI Stimulus block. These are 8B10B encoded and injected into the TBI receiver interface.
- Data frames received at the GMII receiver are checked by the GMII Monitor against the stimulus frames injected into the TBI receiver to ensure data integrity.

Core without MDIO Interface

The demonstration test bench performs the following tasks.

- Input clock signals are generated.
- A reset is applied to the example design.
- The Ethernet 1000BASE-X PCS/PMA core is configured via the Configuration Vector to take the core out of the Isolate state.
- The following frames are injected into the GMII transmitter by the GMII stimulus block.
 - + the first is a minimum length frame
 - + the second is a type frame
 - + the third is an errored frame
 - + the fourth is a padded frame
- The data received at the TBI transmitter interface is 8B10B decoded. The resultant frames are checked by the TBI Monitor against the stimulus frames injected into the GMII transmitter to ensure data is the same.
- The same four frames are generated by the TBI Stimulus block. These are 8B10B encoded and injected into the TBI receiver interface.
- Data frames received at the GMII receiver are checked by the GMII Monitor against the stimulus frames injected into the TBI receiver to ensure data is the same.

Customizing the Test Bench

This section provides information about making modifications to the demonstration test bench files.

Changing Frame Data

You can change the contents of the four frames used by the demonstration test bench by changing the *data* and *valid* fields for each frame defined in the stimulus block. Frames can be added by defining a new frame of data. Any modified frames are automatically updated in both stimulus and monitor functions.

Changing Frame Error Status

Errors can be inserted into any of the predefined frames in any position by setting the *error* field to '1' in any column of that frame. Injected errors are automatically updated in both stimulus and monitor functions.

Changing the Core Configuration

The configuration of the Ethernet 1000BASE-X PCS/PMA core used in the demonstration test bench can be altered.

Caution! Certain configurations of the core can cause the test bench to fail, or to cause processes to run indefinitely. For example, the demonstration test bench will not auto-negotiate with the design example. Determine the configurations that can safely be used with the test bench.

If the MDIO interface option has been selected, the core can be reconfigured by editing the injected MDIO frame in the demonstration test bench top level. See the *Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* for more information about using the MDIO interface.

If the MDIO interface option has not been selected, the core can be reconfigured by modifying the configuration vector directly. See the *Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* for information about using the configuration vector.

SGMII Example Design / Dynamic Switching Example Design Using a RocketIO Transceiver

Note: This is the example design provided when the core is generated for the SGMII standard; it is also provided when the core is generated with the 1000BASE-X/SGMII dynamic switching capability.

Figure 4-5 illustrates an example design for top-level HDL for the Ethernet 1000BASE-X PCS/PMA or SGMII in SGMII mode using a RocketIO transceiver.

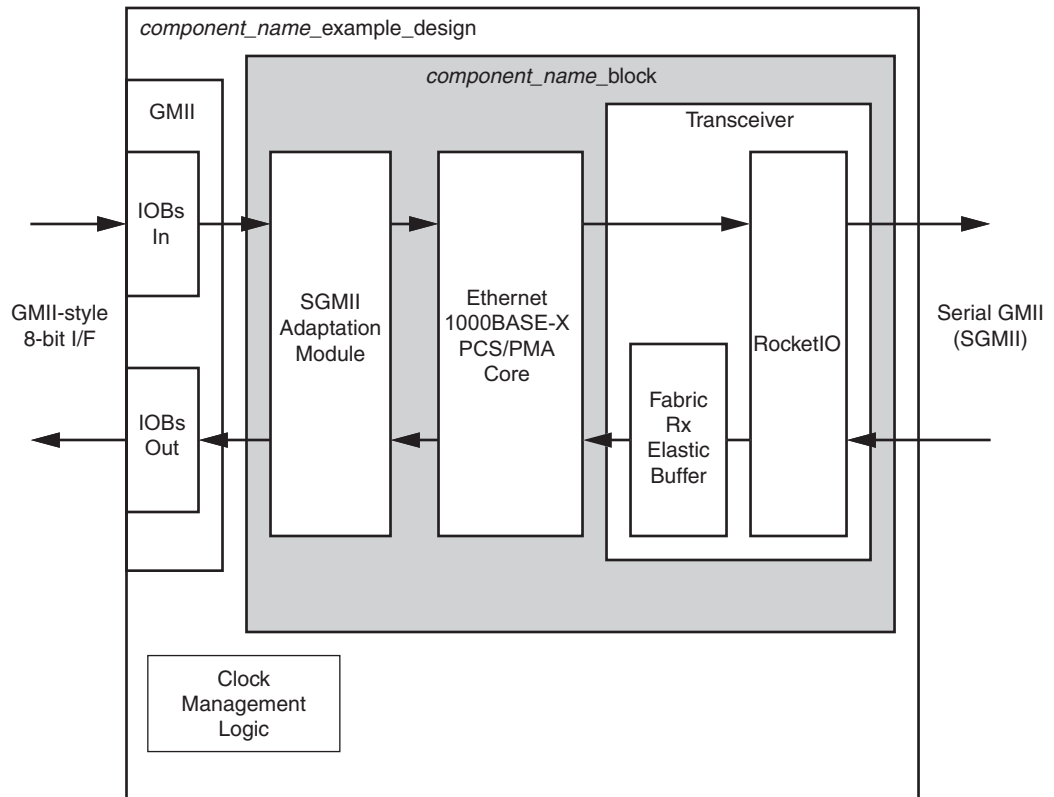


Figure 4-5: Example Design HDL for the Ethernet 1000BASE-X PCS/PMA or SGMII Core in SGMII mode using a RocketIO Transceiver

Top-Level Example Design HDL

The top-level example design for the Ethernet 1000BASE-X PCS/PMA core in SGMII mode is described in the following files:

VHDL

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

Verilog

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

The example design HDL top level contains the following:

- An instance of the SGMII block level

- Clock management logic for the core and the RocketIO transceiver, including DCM (if required) and Global Clock Buffer instances
- External GMII logic, including IOB and DDR register instances, where required

The example design HDL top level connects the GMII of the block level to external IOBs. This allows the functionality of the core to be demonstrated using a simulation package, as described in this guide.

Note: In the Virtex-4 and Virtex-5 families, RocketIO transceivers are provided in pairs. When generated with the appropriate options, the example design is capable of connecting two instances of the core to the RocketIO transceiver pair.

Block Level HDL

The following files describe the block level for the Ethernet 1000BASE-X PCS/PMA core in SGMII mode:

VHDL

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

Verilog

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

The block level contains the following:

- An instance of the Ethernet 1000BASE-X PCS/PMA core in SGMII mode.
- An instance of a RocketIO transceiver specific to the target device (Virtex-II Pro, Virtex-4, or Virtex-5)
- An SGMII adaptation module containing:
 - + The clock management logic required to enable the SGMII example design to operate at 10 Mbps, 100 Mbps, and 1 Gbps.
 - + GMII logic for both transmitter and receiver paths; the GMII style 8-bit interface is run at 125 MHz for 1 Gbps operation; 12.5 MHz for 100 Mbps operation; 1.25 MHz for 10 Mbps operation.

The block-level HDL connects the PHY side interface of the core to a RocketIO transceiver instance and the client side to SGMII Adaptation logic as illustrated in [Figure 4-5](#). This is the most useful part of the example design and should be instantiated in all customer designs that use the core.

Note: In the Virtex-4 and Virtex-5 families, RocketIO transceivers are provided in pairs. When generated with the appropriate options, the block level is capable of connecting two instances of the core to the RocketIO transceiver.

Transceiver

A wrapper file for the RocketIO transceiver specific to the target device (Virtex-II Pro, Virtex-4, or Virtex-5) is described in the following files:

VHDL

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

Verilog

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

This file instances a RocketIO transceiver and applies Gigabit Ethernet 1000BASE-X attributes to it. This transceiver wrapper is instantiated from the top-level HDL file for the example design.

In SGMII or Dynamic Switching modes, the internal RocketIO transceiver Rx Buffer is now optionally bypassed. If bypassed, a larger buffer is instead implemented in the FPGA fabric (see “Receiver Elastic Buffer” below), and is instantiated from within this transceiver wrapper.

Note: In Virtex-4 and Virtex-5 devices, RocketIO transceivers are provided in pairs. When generated with the appropriate options, the block level is capable of connecting two instances of the core to the RocketIO transceiver. When only a single instance of the core is requested, the unused RocketIO transceiver from the pair is still instantiated from within this transceiver wrapper but is left unconnected.

Virtex-5 Specific Transceiver Files

RocketIO GTP Transceiver

For Virtex-5 LXT and SXT devices, the transceiver wrapper file directly instantiates RocketIO GTP transceiver wrapper files created from the Virtex-5 RocketIO GTP Wizard. These files tie off (or leaves unconnected) unused I/O for the GTP pair, and apply the 1000BASE-X attributes. The files can be edited/tailored by rerunning the RocketIO GTP Wizard and swapping these files. These are the following files:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtp.vhd  
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtp_tile.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtp.v  
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtp_tile.v
```

In order to Re-run the RocketIO Transceiver GTP Wizard, a CORE Generator XCO file for the RocketIO Transceiver GTP Wizard has also been included. This file lists all of the RocketIO Transceiver GTP Wizard attributes used to generate the files above. See the CORE Generator documentation for more information about XCO files. The XCO file is in the following location:

```
<project_dir>/<component_name>/example_design/transceiver/  
rocketio_wrapper_gtp.xco
```

RocketIO GTX

For Virtex-5 FXT devices, the transceiver wrapper file directly instantiates RocketIO GTX transceiver wrapper files created from the Virtex-5 RocketIO GTX Wizard. These files tie off (or leaves unconnected) unused I/O for the GTX pair, and apply the 1000BASE-X attributes. The files can be edited/tailored by rerunning the RocketIO GTX Wizard and swapping these files. These are the following files:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/
rocketio_wrapper_gtx.vhd
<project_dir>/<component_name>/example_design/transceiver/
rocketio_wrapper_gtx_tile.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/
rocketio_wrapper_gtx.v
<project_dir>/<component_name>/example_design/transceiver/
rocketio_wrapper_gtx_tile.v
```

To Re-run the RocketIO GTX Wizard, a CORE Generator XCO file for the RocketIO GTX Wizard has also been included. This file lists all of the RocketIO GTX Wizard attributes which were used in the generation of the above files. Please refer to CORE Generator documentation for further information about XCO files. The XCO file is located:

```
<project_dir>/<component_name>/example_design/transceiver/
rocketio_wrapper_gtx.xco
```

Virtex-4 Specific Transceiver Files

Calibration Blocks

For Virtex-4 FX devices only, Calibration Blocks are required. A Calibration block is connected to both GT11 A and B within the RocketIO transceiver tile. This occurs in the transceiver wrapper file. See [Answer Record 22477](#) for information about downloading the *Calibration Block User Guide*.

The Calibration Block is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/
cal_block_v1_4_1.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/
cal_block_v1_4_1.v
```

GT11 Reset/Initialization Circuitry

Precise reset/initialization circuitry is required for the GT11 RocketIO transceivers.

The reset circuitry for the RocketIO receiver is illustrated in *Figure 2-18* of the *Virtex-4 RocketIO Multi-Gigabit Transceiver User Guide* (UG076). This is implemented in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/
gt11_init_rx.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/  
gt11_init_rx.v
```

The reset circuitry for the RocketIO Transmitter is illustrated in *Figure 2-13* of the *Virtex-4 FPGA RocketIO Multi-Gigabit Transceiver User Guide* (UG076). This is implemented in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/  
gt11_init_tx.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/  
gt11_init_tx.v
```

Both receiver and transmitter reset circuitry entities are instantiated from within the block level of the example design.

Receiver Elastic Buffer

The Receiver Elastic Buffer is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/  
rx_elastic_buffer.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/  
rx_elastic_buffer.v
```

In SGMII or Dynamic Switching modes, the Rx Buffer in the RocketIO transceiver is optionally bypassed. If bypassed, a larger buffer is implemented in the FPGA fabric and instantiated from within the transceiver wrapper.

This alternative Receiver Elastic Buffer uses a single block RAM to create a buffer twice as large as the one present in the RocketIO transceiver, which is able to cope with larger frame sizes before clock tolerances accumulate and result in an emptying or filling of the buffer. See the *Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* for additional information.

SGMII Adaptation Module

The GMII of the core always operates at 125 MHz. The core makes no differentiation between the three speeds of operation; it always effectively operates at 1 Gbps. However, at 100 Mbps, every data byte run through the core should be repeated 10 times to achieve the required bit rate; at 10 Mbps, each data byte run through the core should be repeated 100 times to achieve the required bit rate. Dealing with this repetition of bytes is the function of the SGMII adaptation module. The SGMII adaptation module is described in [Appendix, “SGMII Adaptation Module”](#).

Demonstration Test Bench

Figure 4-6 illustrates the demonstration test bench for the Ethernet 1000BASE-X PCS/PMA or SGMII Core in SGMII mode. The demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself.

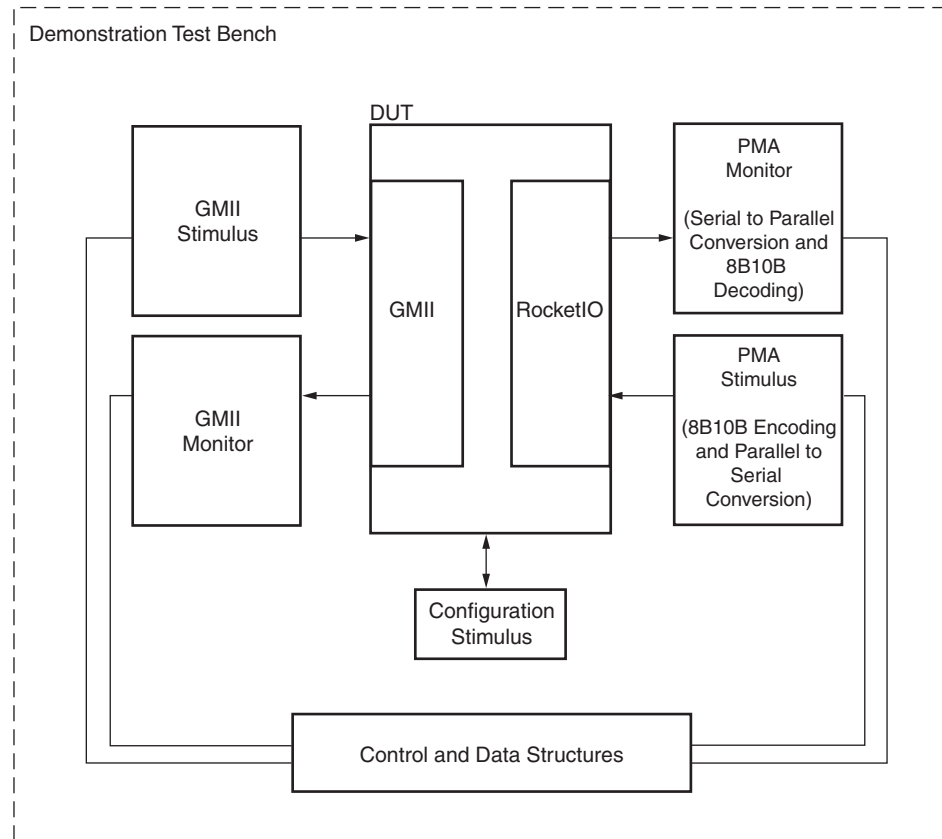


Figure 4-6: **Demonstration Test Bench for the Ethernet 1000BASE-X PCS/PMA or SGMII Core in SGMII Mode using RocketIO Transceivers**

The top-level test bench entity instantiates the example design for the core, which is the Device Under Test (DUT). A stimulus block is also instantiated and clocks, resets and test bench semaphores are created. The following files describe the top-level of the demonstration test bench.

VHDL

```
<project_dir>/<component_name>/simulation/demo_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/demo_tb.v
```

The stimulus block entity, instantiated from within the top-level test bench, creates the Ethernet stimulus in the form of four Ethernet frames, which are injected into GMII and PHY interfaces of the DUT. The output from the DUT is also monitored for errors. The following files describe the stimulus block of the demonstration test bench.

VHDL

```
<project_dir>/<component_name>/simulation/stimulus_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/stimulus_tb.v
```

Together, the top-level test bench file and the stimulus block combine to provide the full test bench functionality which is described in the sections that follow.

Note: In the Virtex-4 and Virtex-5 families, RocketIO transceivers are provided in pairs. When generated with the appropriate options, the example design is capable of connecting two instances of the core to the RocketIO transceiver pair. When this is the case, two stimulus blocks are instantiated from the top level test bench to independently exercise both cores.

Test Bench Functionality

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The Ethernet 1000BASE-X PCS/PMA core is configured through the MDIO interface by injecting an MDIO frame into the example design. This disables Auto-Negotiation and takes the core out of Isolate state.
- The following frames are injected into the GMII transmitter by the GMII stimulus block at 1 Gbps.
 - + the first is a minimum length frame
 - + the second is a type frame
 - + the third is an errored frame
 - + the fourth is a padded frame
- The serial data received at the RocketIO transceiver transmitter interface is converted to 10-bit parallel data, then 8B10B decoded. The resulting frames are checked by the PMA Monitor against the stimulus frames injected into the GMII transmitter to ensure data integrity.
- The same four frames are generated by the PMA Stimulus block. These are 8B10B encoded, converted to serial data and injected into the RocketIO transceiver receiver interface at 1 Gbps.
- Data frames received at the GMII receiver are checked by the GMII Monitor against the stimulus frames injected into the RocketIO transceiver receiver to ensure data integrity.

Customizing the Test Bench

Changing Frame Data

You can change the contents of the four frames used by the demonstration test bench by changing the *data* and *valid* fields for each frame defined in the stimulus block. New frames can be added by defining a new frame of data. Modified frames are automatically updated in both stimulus and monitor functions.

Changing Frame Error Status

Errors can be inserted into any of the predefined frames in any position by setting the *error* field to '1' in any column of that frame. Injected errors are automatically updated in both stimulus and monitor functions.

Changing the Core Configuration

The configuration of the Ethernet 1000BASE-X PCS/PMA core used in the demonstration test bench can be altered.

Caution! Certain configurations of the core cause the test bench to fail, or to cause processes to run indefinitely. For example, the demonstration test bench will not Auto-Negotiate with the design example. Determine the configurations that can safely be used with the test bench.

The core can be reconfigured by editing the injected MDIO frame in the demonstration test bench top level. See the *Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* for information about using the MDIO interface.

Changing the Operational Speed

SGMII can be used to carry Ethernet traffic at 10 Mbps, 100 Mbps or 1 Gbps. By default, the demonstration test bench is configured to operate at 1 Gbps. The speed of both the example design and test bench can be set to the desired operational speed by editing the following settings, recompiling the test bench, then running the simulation again.

1 Gbps Operation

```
set speed_is_10_100 to logic 0
```

100 Mbps Operation

```
set speed_is_10_100 to logic 1
```

```
set speed_is_100 to logic 1
```

10 Mbps Operation

```
set speed_is_10_100 to logic 1
```

```
set speed_is_100 to logic 0
```


- Clock management logic, including DCM and Global Clock Buffer instances, where required
- External GMII logic, including IOB and DDR register instances, where required

The example design HDL top level connects the GMII of the block level to external IOBs. This allows the functionality of the core to be demonstrated using a simulation package, as described in this guide.

Block Level HDL

The following files describe the block level for the Ethernet 1000BASE-X PCS/PMA core in SGMII mode:

VHDL

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

Verilog

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

The block level contains the following:

- An instance of the Ethernet 1000BASE-X PCS/PMA core in SGMII mode.
- TBI interface logic, including IOB and DDR registers instances, where required.
- An SGMII adaptation module containing:
 - + The clock management logic required to enable the SGMII example design to operate at 10 Mbps, 100 Mbps, and 1 Gbps.
 - + GMII logic for both transmitter and receiver paths; the GMII style 8-bit interface is run at 125 MHz for 1 Gbps operation; 12.5 MHz for 100 Mbps operation; 1.25 MHz for 10 Mbps operation.

The block level HDL connects the TBI of the core to external IOBs and the client side to SGMII Adaptation logic as illustrated in [Figure 4-7](#). This is the most useful part of the example design and should be instantiated in all customer designs that use the core.

The following files describe the block level design for the Ethernet 1000BASE-X PCS/PMA core with TBI:

VHDL

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

Verilog

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

SGMII Adaptation Module

The GMII of the core always operates at 125 MHz. The core makes no differentiation between the three speeds of operation; it always effectively operates at 1 Gbps. However, at 100 Mbps, every data byte run through the core should be repeated 10 times to achieve the required bit rate; at 10 Mbps, each data byte run through the core should be repeated 100 times to achieve the required bit rate. Dealing with this repetition of bytes is the function of the SGMII adaptation module. The SGMII adaptation module is described in [Appendix, "SGMII Adaptation Module"](#).

Demonstration Test Bench

Figure 4-8 illustrates the demonstration test bench for the Ethernet 1000BASE-X PCS/PMA or SGMII Core in SGMII mode with the TBI. The demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself.

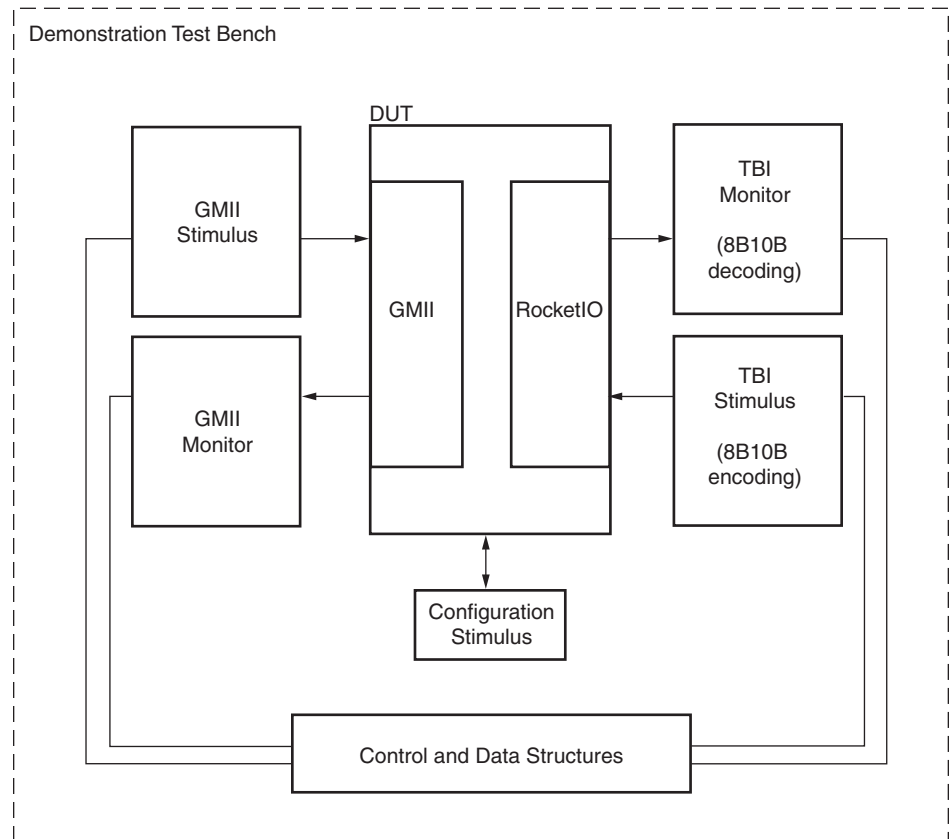


Figure 4-8: **Demonstration Test Bench for the Ethernet 1000BASE-X PCS/PMA or SGMII Core in SGMII Mode with TBI**

The top-level test bench entity instantiates the example design for the core, which is the Device Under Test (DUT). A stimulus block is also instantiated and clocks, resets and test bench semaphores are created. The following files describe the top-level of the demonstration test bench.

VHDL

```
<project_dir>/<component_name>/simulation/demo_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/demo_tb.v
```

The stimulus block entity, instantiated from within the top-level test bench, creates the Ethernet stimulus in the form of four Ethernet frames, which are injected into GMII and TBI interfaces of the DUT. The output from the DUT is also monitored for errors. The following files describe the stimulus block of the demonstration test bench.

VHDL

```
<project_dir>/<component_name>/simulation/stimulus_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/stimulus_tb.v
```

Together, the top-level test bench file and the stimulus block combine to provide the full test bench functionality which is described in the sections that follow.

Test Bench Functionality

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The Ethernet 1000BASE-X PCS/PMA core is configured through the MDIO interface by injecting an MDIO frame into the example design. This disables Auto-Negotiation and takes the core out of Isolate state.
- The following frames are injected into the GMII transmitter by the GMII stimulus block at 1 Gbps.
 - + the first is a minimum length frame
 - + the second is a type frame
 - + the third is an errored frame
 - + the fourth is a padded frame
- The data received at the TBI transmitter interface is 8B10B decoded. The resulting frames are checked by the TBI Monitor against the stimulus frames injected into the GMII transmitter to ensure data integrity.
- The same four frames are generated by the TBI Stimulus block. These are 8B10B encoded and injected into the TBI receiver interface.
- Data frames received at the GMII receiver are checked by the GMII Monitor against the stimulus frames injected into the RocketIO transceiver receiver to ensure data integrity.

Customizing the Test Bench

Changing Frame Data

You can change the contents of the four frames used by the demonstration test bench by changing the *data* and *valid* fields for each frame defined in the stimulus block. New frames can be added by defining a new frame of data. Modified frames are automatically updated in both stimulus and monitor functions.

Changing Frame Error Status

Errors can be inserted into any of the predefined frames in any position by setting the *error* field to '1' in any column of that frame. Injected errors are automatically updated in both stimulus and monitor functions.

Changing the Core Configuration

The configuration of the Ethernet 1000BASE-X PCS/PMA core used in the demonstration test bench can be altered.

Caution! Certain configurations of the core cause the test bench to fail, or to cause processes to run indefinitely. For example, the demonstration test bench will not Auto-Negotiate with the design example. Determine the configurations that can safely be used with the test bench.

The core can be reconfigured by editing the injected MDIO frame in the demonstration test bench top level. See the *Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* for information about using the MDIO interface.

Changing the Operational Speed

SGMII can be used to carry Ethernet traffic at 10 Mbps, 100 Mbps or 1 Gbps. By default, the demonstration test bench is configured to operate at 1 Gbps. The speed of both the example design and test bench can be set to the desired operational speed by editing the following settings, recompiling the test bench, then running the simulation again.

1 Gbps Operation

```
set speed_is_10_100 to logic 0
```

100 Mbps Operation

```
set speed_is_10_100 to logic 1  
set speed_is_100 to logic 1
```

10 Mbps Operation

```
set speed_is_10_100 to logic 1  
set speed_is_100 to logic 0
```


SGMII Adaptation Module

Introduction

When the core is generated in SGMII or Dynamic Switching mode, the block level of the core contains the *SGMII Adaptation Module* (see [Figure 4-5](#) and [Figure 4-7](#)). This is because the GMII of the core always operates at 125 MHz: the core makes no differentiation between the three speeds of operation, it always effectively operates at 1 Gbps. However, at 100 Mbps, every data byte run through the core should be repeated ten times to achieve the required bit rate; at 10 Mbps, each data byte run through the core should be repeated 100 times to achieve the required bit rate. Dealing with this repetition of bytes is the function of the SGMII adaptation module.

The provided SGMII adaptation module ([Figure 5-1](#)) creates a GMII-style interface that clocks the following frequencies:

- 125 MHz when operating at a speed of 1 Gbps (with no repetition of data bytes)
- 12.5 MHz at a speed of 100 Mbps (each data byte is repeated and run through the core 10 times)
- 1.25 MHz at a speed of 10 Mbps (each data byte is repeated and run through the core 100 times)

This GMII-style interface is not a standard interface (true GMII only operates at a clock frequency of 125 MHz), but it does allow a straightforward internal connection to an Ethernet MAC core. For example, the SGMII adaptation module can be used to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core, operating in SGMII mode, to the Xilinx Tri-Mode Ethernet MAC core. See the *LogiCORE IP Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* for more information).

File Structure and Functional Description

SGMII Adaptation Module Top Level

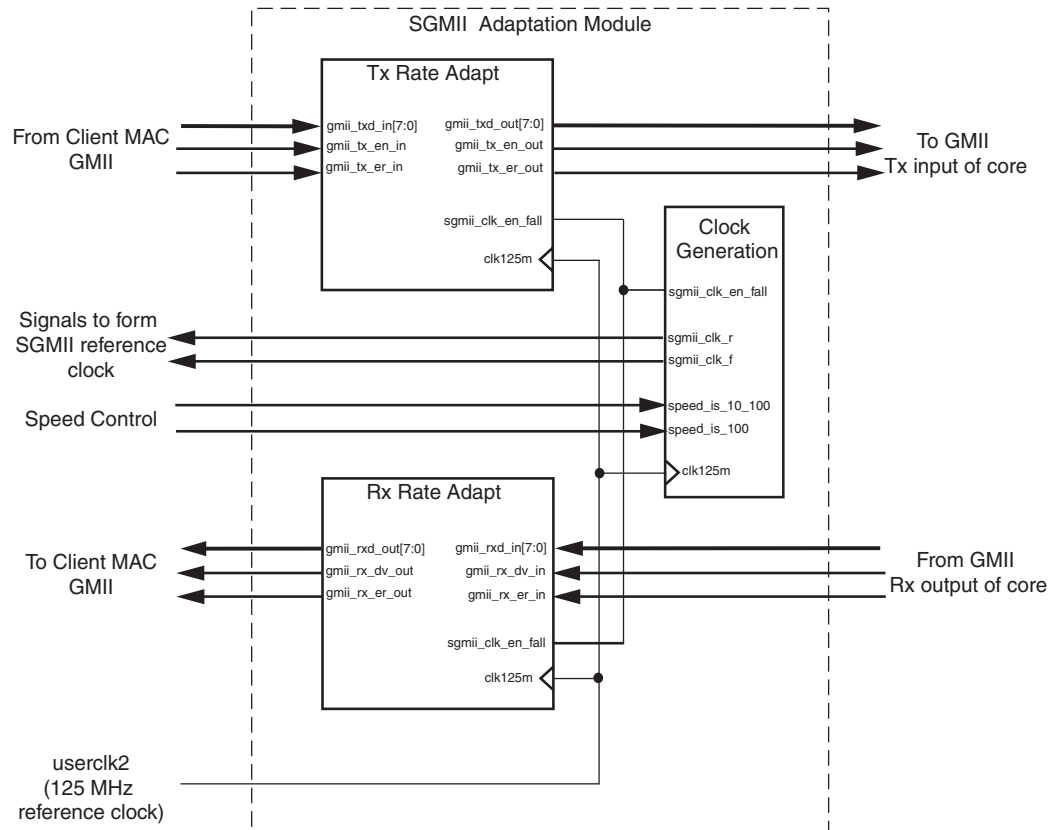


Figure 5-1: SGMII Adaptation Module

The top-level HDL for the SGMII adaptation module is described in the following files:

VHDL

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

Verilog

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

The SGMII adaptation module is described in several hierarchical sub-modules as illustrated in Figure 5-1. These sub-modules are described in separate HDL files as shown in the following sections.

Clock Generation

The clock generation module is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/sgmii_adapt/  
clk_gen.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/sgmii_adapt/  
clk_gen.v
```

This file creates the necessary clocks and clock enables for use throughout the SGMII adaptation module. Clock frequencies are:

- 125 MHz at an operating speed of 1 Gbps
- 12.5 MHz at an operating speed of 100 Mbps
- 1.25 MHz at an operating speed of 10 Mbps

[Figure 5-2](#) illustrates the output clock and clock enable signals for the Clock Generation module at 1 Gbps and 100 Mbps speeds.

At 1 Gbps, `sgmii_clk_r` is fixed at logic 0; `sgmii_clk_f` is fixed at logic 1. `sgmii_clk_r` is connected to the rising edge triggered flip-flop of an IOB output DDR, clocked with `clk125m`. `sgmii_clk_f` is connected to the falling edge triggered flip-flop of the same IOB output DDR. The result is the production of an inverted clock, `sgmii_clk`, that is forwarded off-chip. This IOB DDR output register is included in the top-level HDL for the example design.

At 100 Mbps, the `sgmii_clk_r` and `sgmii_clk_f` signals toggle at the required clock frequency (every five clock periods of `clk125m`), also illustrated in [Figure 5-2](#). `sgmii_clk_r` is synchronous to the rising edge of the 125 MHz reference clock (`clk125m`); `sgmii_clk_f` is synchronous to the falling edge of the `clk125m`. These are routed to the rising and falling edges of the IOB DDR output register to forward the SGMII reference clock (`sgmii_clk`) off chip.

At 10 Mbps, the situation is identical to that of 100 Mbps, with the exception that `sgmii_clk_r` and `sgmii_clk_f` toggle every 50 clock periods of `clk125m`.

`sgmii_clk_en_fall` is used as a clock enable throughout the SGMII adaptation logic. At 1 Gbps it is fixed at logic 1 indicating that every `clk125m` period is significant. At 100 Mbps, this signal is valid for a single period of `clk125m` every ten clocks and marks the falling edge of the SGMII reference clock, `sgmii_clk`. At 10 Mbps, this signal is valid for a single period of `clk125m` every one hundred clocks and again marks the falling edge `sgmii_clk`. This clock enable signal is used as the control for the data byte repetition in the Transmitter and Receiver Rate Adaptation modules.

Figure 5-2 shows the clock generator output clocks and clock enable configurations.

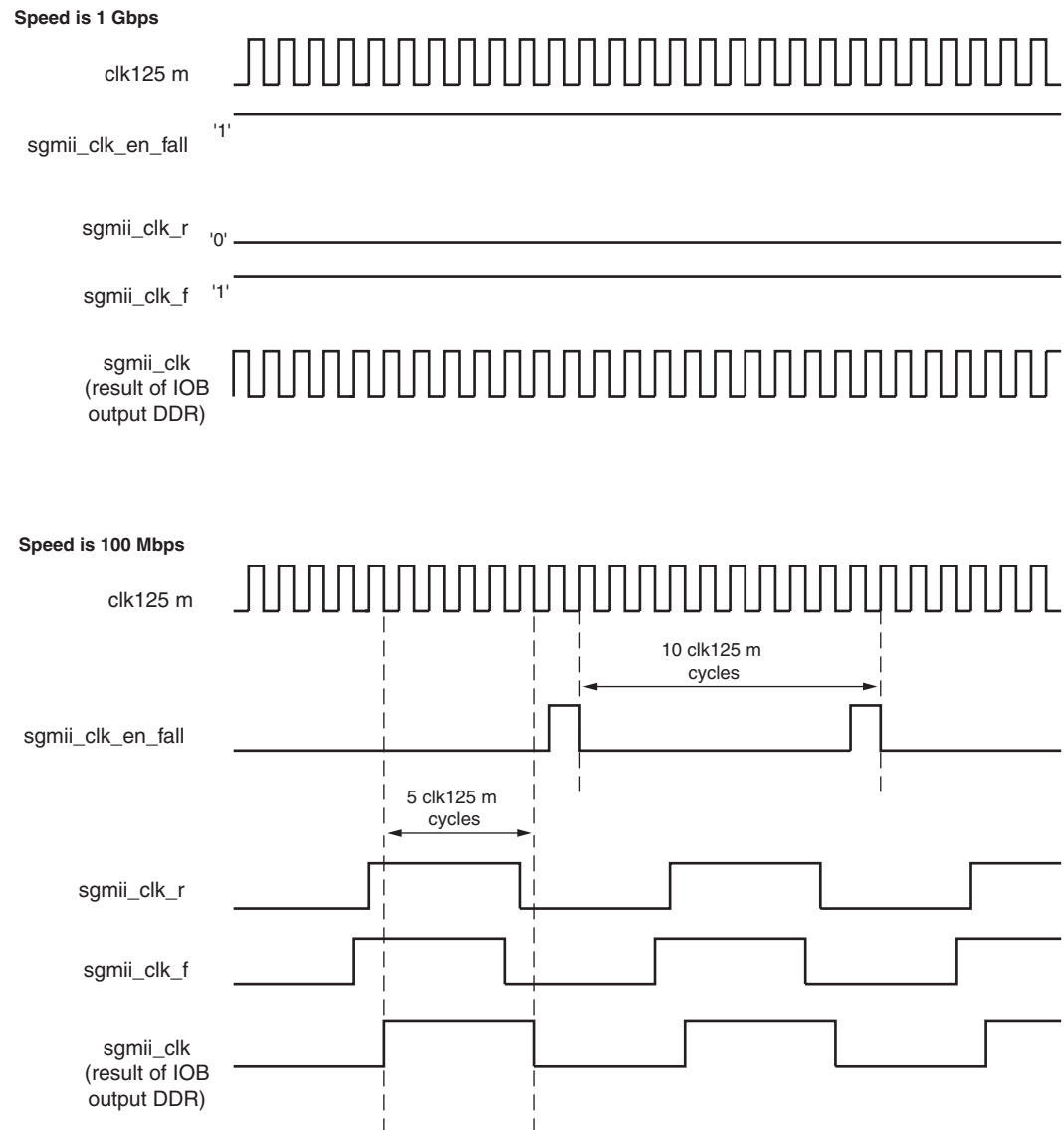


Figure 5-2: Clock Generator Output Clocks and Clock Enable

Johnson Counter

The Johnson Counter is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/sgmmi_adapt/  
johnson_cntr.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/sgmmi_adapt/  
johnson_cntr.v
```

The Johnson Counter is instantiated twice by the clock generation circuitry. The Johnson Counter is a shift register based clock divider that provides a divide-by-ten clock output.

The divide-by-ten clock is output directly from a flip-flop triggered on the rising edge of the 125 MHz reference clock, `clk125m`.

Johnson Counter capabilities are extended by using the clock enables; it is only the clock-enabled cycles that trigger the shift register and are therefore divided down.

Transmitter Rate Adaptation Module

The Transmitter Rate Adaptation module is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/sgmii_adapt/  
tx_rate_adapt.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/sgmii_adapt/  
tx_rate_adapt.v
```

This module accepts transmitter data from the GMII-style interface from the attached client MAC, and samples the input data on the 125 MHz reference clock, `clk125m`. This sampled data can then be connected directly to the input GMII of the Ethernet 1000BASE-X PCS/PMA or SGMII core. The 1 Gbps and 100 Mbps cases are illustrated in [Figure 5-3](#).

At 1 Gbps, the client MAC should drive the GMII transmitter data synchronously to the rising edge of `clk125m`. `sgmii_clk_en_fall` (derived from the Clock Generation module) is fixed at logic 1, and the input data is sampled on every clock cycle.

At 100 Mbps and 10 Mbps, the client MAC should drive the GMII transmitter data synchronously to the rising edge of `sgmii_clk`. The data is sampled (see [Figure 5-3](#)) on the `sgmii_clk_en_fall` pulse. Because this pulse marks the falling edge of `sgmii_clk`, it guarantees that the data is stable when sampled. The frequency of the `sgmii_clk_en_fall` pulse ensures that this data is repeated exactly 10 times when operating at a speed of 100 Mbps and 100 times when operating at a speed of 10 Mbps.

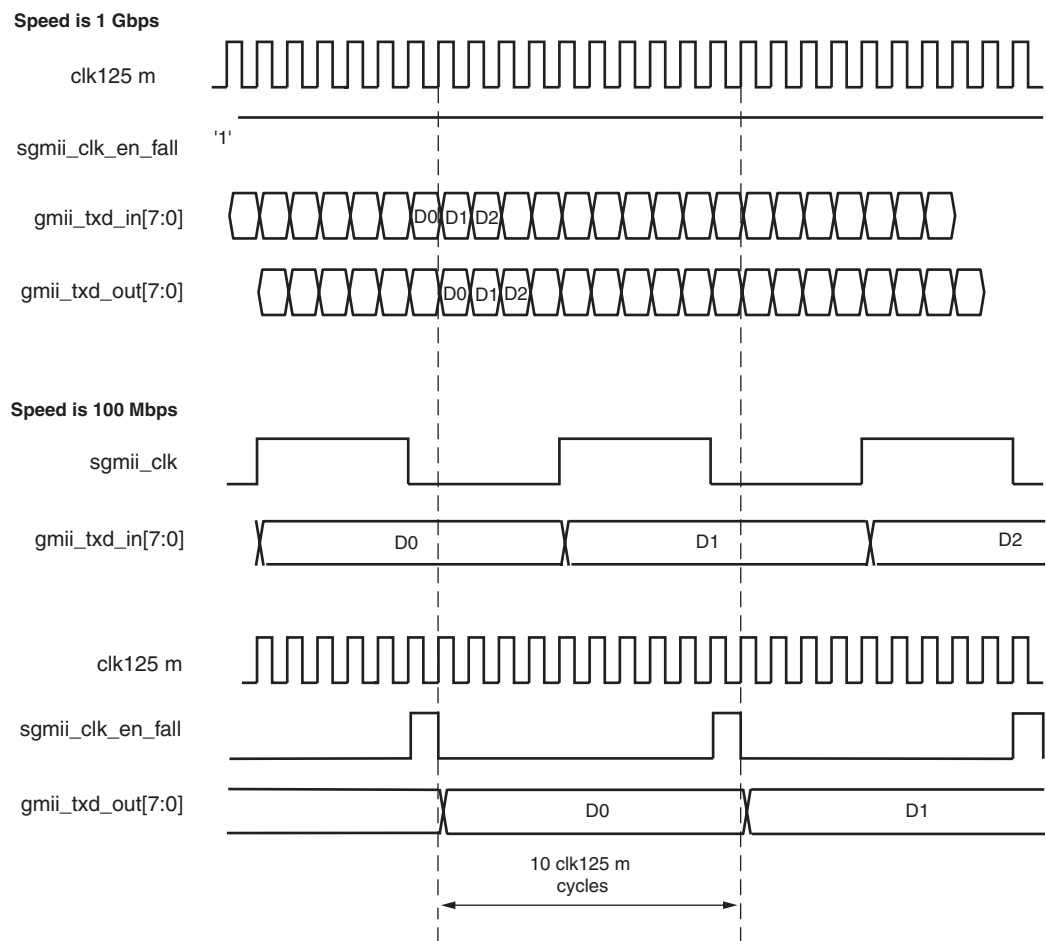


Figure 5-3: Transmitter Rate Adaptation Module Data Sampling

Receiver Rate Adaptation Module

The Receiver Rate Adaptation module is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/sgmii_adapt/  
rx_rate_adapt.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/sgmii_adapt/  
rx_rate_adapt.v
```

This module accepts received data from the Ethernet 1000BASE-X PCS/PMA or SGMII core. This data is sampled and sent out of the GMII receiver interface for the attached client MAC. The 1 Gbps and 100 Mbps cases are illustrated in [Figure 5-4](#).

At 1 Gbps, the data is valid on every clock cycle of the 125 MHz reference clock (clk125m). Data received from the core is clocked straight through the Receiver Rate Adaptation module.

At 100 Mbps, the data is repeated for a 10 clock period duration of `clk125m`; at 10 Mbps, the data is repeated for a 100 clock period duration of `clk125m`. The Receiver Rate Adaptation Module samples this data on the `sgmii_clk_en_fall` signal produced from the Clock Generation circuitry. Because this pulse marks the falling edge of `sgmii_clk`, it guarantees that setup and hold time is provided for the attached client MAC.

The Receiver Rate Adaptation module also performs a second function that accounts for the latency inferred in [Figure 5-4](#). The 8-bit Start of Frame Delimiter (SFD) code is detected, and if required, it is realigned across the 8-bit data path of `gmii_rxd_out[7:0]` before being presented to the attached client MAC. It is possible that this SFD could have been skewed across two separate bytes by MACs operating on a 4-bit data path.

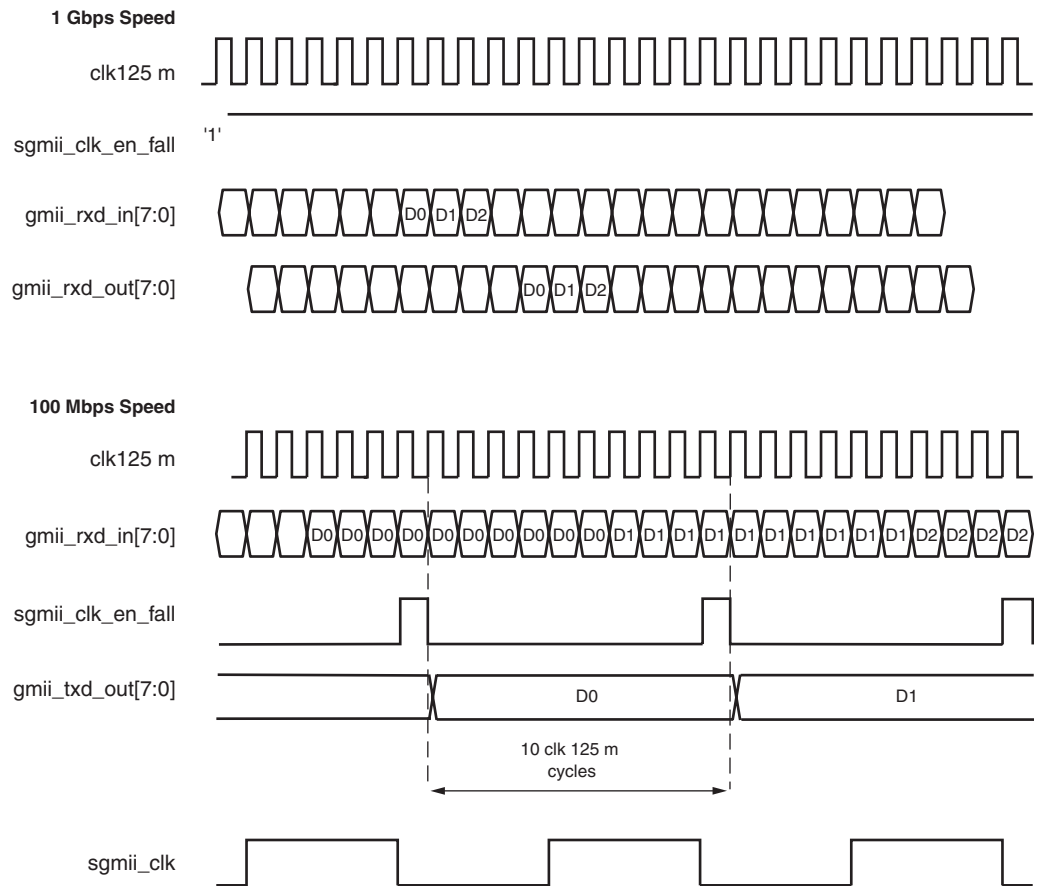


Figure 5-4: Receiver Rate Adaptation Module Data Sampling

