LogiCORE™ IP 1-Gigabit Ethernet MAC v8.5

Getting Started Guide

UG143 April 24, 2009

XILINX®

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 1-Gigabit Ethernet MAC version 6.0, Xilinx tools 7.1i.
01/18/06	3.0	Updated to 1-Gigabit Ethernet MAC version 7.0, Xilinx tools 8.1i.
07/13/06	4.0	Updated to 1-Gigabit Ethernet MAC version 8.0, Xilinx tools 8.2i.
09/21/06	5.1	Updated to 1-Gigabit Ethernet MAC version 8.1, support for Spartan®-3A devices.
02/15/07	6.0	Updated to 1-Gigabit Ethernet MAC version 8.2, Xilinx tools 9.1i.
08/08/07	7.0	Updated to core version 8.3, and supported tool updates for IP1 I Minor release.
04/24/09	8.0	Updated to core version 8.5, Xilinx tools 11.1. Updated System Requirements section.
The product was discontinued as of August 31, 2009.		

Table of Contents

Chapter 4: Detailed Example Design

Schedule of Figures

Chapter 1: Introduction

Chapter 2: Licensing the Core

Chapter 3: Quick Start Example Design

Chapter 4: Detailed Example Design

About This Guide

This guide provides information about generating a LogiCORE™ IP 1-Gigabit Ethernet MAC (GEMAC) core, customizing and simulating the core using 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 this guide and the conventions used in the guide.
- Chapter 1, "Introduction" describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Licensing the Core" provides information about licensing the core.
- Chapter 3, "Quick Start Example Design" 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 files and directory structure generated by CORE Generator™, the contents of the HDL example design, and the operation of the demonstration test bench.

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`*
Italic font	References to other manuals	See the *User Guide* for details.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
<text in brackets>	User-defined variable for directory names.	<component_name>
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild [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 . . .	Repetitive material that has been omitted	**`IOB #1: Name = QOUT'`** **`IOB #2: Name = CLKIN'`** . . .
Horizontal ellipsis `...`	Repetitive material that has been omitted	**`allow block`** *`block_name loc1 loc2 ... locn;`*
Notations	The prefix '`0x`' or the suffix '`h`' indicate hexadecimal notation	A read of address `0x00112975` returned `45524943`h.
	A '`_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.

List of Acronyms

The following table describes acronyms used in this manual.

Acronym	Spelled Out
DA	Destination Address
DCM	Digital Clock Manager
DDR	Double Data Rate
FCS	Frame Check Sequence
FIFO	First In First Out
FPGA	Field Programmable Gate Array.
Gbps	Gigabit per second
GEMAC	Gigabit Ethernet Media Access Controller
GMII	Gigabit Media Independent Interface
HDL	Hardware Description Language
IOB	Input/Output Block
IP	Intellectual Property
MAC	Media Access Controller
RGMII	Reduced Gigabit Media Independent Interface
SA	Source Address
TEMAC	Tri-Mode Ethernet MAC
VHDL	VHSIC Hardware Description Language (VHSIC an acronym for Very High-Speed Integrated Circuits).

Chapter 1

Introduction

The 1-Gigabit Ethernet MAC (GEMAC) 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.

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

System Requirements

Windows

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

Linux

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

Software

- ISE® 11.1

About the Core

The GEMAC core is a Xilinx CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see the GEMAC product page. For information about licensing options, see Chapter 2, "Licensing the Core."

Recommended Design Experience

Although the GEMAC 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 (UCF) is recommended.

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

Additional Core Resources

For detailed information and updates about the GEMAC core, see the following documents, available from either the GEMAC product page.

- *1-Gigabit Ethernet MAC Data Sheet*
- *1-Gigabit Ethernet MAC Getting Started Guide*

After generating the core, the following documents are available from the doc directory:

- *1-Gigabit Ethernet MAC Release Notes*
- 1-*Gigabit Ethernet MAC User Guide*

Technical Support

To obtain technical support specific to the GEMAC core, visit support.xilinx.com/. Questions are routed to a team of engineers with expertise using the GEMAC core.

Xilinx will provide technical support for use of this product as described in the *1-Gigabit Ethernet MAC User Guide* and the *1-Gigabit Ethernet MAC Getting Started 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 GEMAC core and the documentation supplied with the core.

GEMAC Core

For comments or suggestions about the GEMAC core, please submit a WebCase from support.xilinx.com/. 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 support.xilinx.com/. Be sure to include the following information:

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

XILINX®

Chapter 2

Licensing the Core

This chapter provides instructions for obtaining a license for the GEMAC core, which you must do before using it in your designs. The GEMAC core is provided under the terms of the Xilinx LogiCORE Site License Agreement, which conforms to the terms of the SignOnce IP License standard defined by the Common License Consortium. Purchase of the core entitles you to technical support and access to updates for a period of one year.

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 GEMAC product page.

License Options

The GEMAC core provides three licensing options. After installing the required Xilinx ISE software and IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator. This key lets you assess core functionality with either the example design provided with the GEMAC core, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place and route the design, evaluate timing, and perform back-annotated gate-level simulation of the GEMAC core using the demonstration test bench provided with the core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before *timing out* (ceasing to function) at which time it can be reactivated by reconfiguring the device.

Full

The Full license key is available when you purchase the core and 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 Key

This section contains information about obtaining a simulation, full system hardware, and full license keys.

Simulation License

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

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license, do the following:

- Navigate to the GEMAC product page.
- Click Evaluate.
- Follow the instructions to install the required Xilinx ISE software and IP Service Packs.

Obtaining a Full License

To obtain a Full license key, you must purchase a license for the core. After doing so, click the "Access Core" link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

Chapter 3

Quick Start Example Design

This chapter introduces the example design included with the GEMAC core and demonstrates how to generate and use the example design with default options.

Overview

The GEMAC example design includes the following:

- An instance of the GEMAC core
- An HDL example design top level and sub-components
- A demonstration test bench, to exercise the example design and core

Figure 3-1 illustrates the GEMAC example design and test bench. The example design has been tested with:

- Xilinx ISE® v11.1 software
- Mentor Graphics ModelSim 6.4b and above.
- Cadence IUS v8.1-s009 and above.
- Synopsys 2008.09 and above.

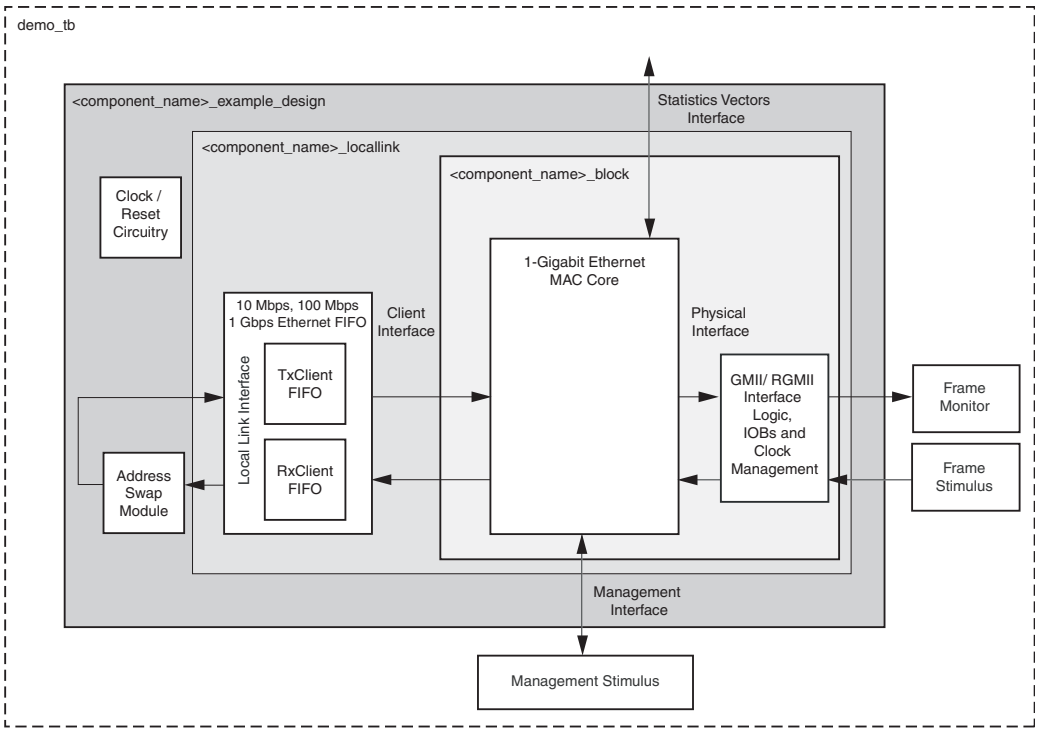

Figure 3-1: **Default Example Design and Test Bench**

Generating the Core

This section describes how to generate a GEMAC core using the Xilinx CORE Generator™. The generated core contains default values.

To generate the core:

1. Start the Xilinx CORE Generator.

   For help starting and using the CORE Generator, see the Xilinx CORE Generator Guide, available from the ISE documentation web page.
2. Choose File > New Project.
3. Enter a directory name. In this example, the directory is named <*project_dir*>.
4. Set project options:
   a. From the Part tab, select an FPGA family that supports the core; for example, Virtex®-5.

      Note: If an unsupported silicon family is selected, the GEMAC core does not appear in the taxonomy tree. For a list of supported architectures, see the *1-Gigabit Ethernet MAC Data Sheet*.

      Note: Leave the device, package, and speed grade files at their default values.

   b. From the Generation tab select VHDL or Verilog for Design Entry select; select Other for Vendor.

   c. On the Advanced tab, leave Options at default values.

5. After creating the project, locate the directory containing the core in the taxonomy tree.

   The project appears in one of the following:

   – Communication & Networking /Ethernet
   – Communication & Networking /Networking
   – Communication & Networking/Telecommunications

6. Double-click the core. If the license file is not properly configured, the CORE Generator displays an error. See Chapter 2, "Licensing the Core" for details.
7. If a warning appears regarding the limitations of the available license, click OK. The Gigabit Ethernet MAC customization screen appears.

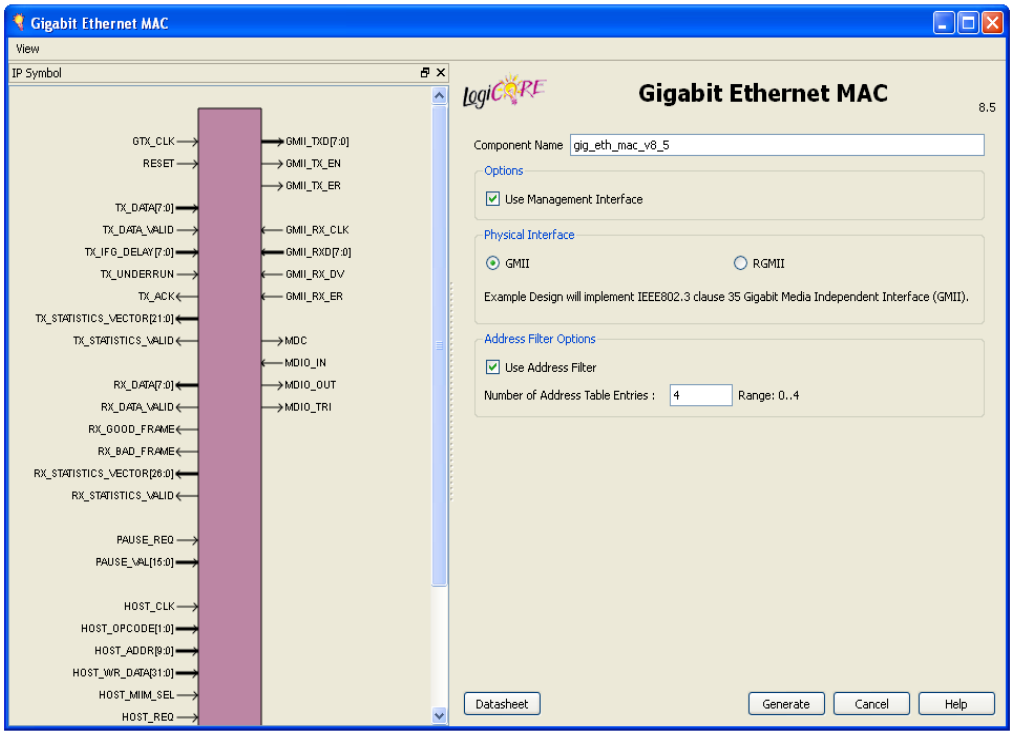

Figure 3-2: **Gigabit Ethernet MAC Customization Screen**

8. Accept the default values; then click Finish.

   The core, named *gig_eth_mac_v8_5* by default, along with supporting core files including the example design, is generated in the project directory. For detailed information about the example design files and directories, see "Directory and File Contents," on page 26.

Implementing the Example Design

Note: If the core is generated with a Simulation Only license, the implementation feature of the example design is not available. In this instance, go to "Simulating the Example Design," on page 20.

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

In the implementation example that follows, *gig_eth_mac_v8_5* is the component name as generated by default from the core customization screen. If a core is generated with a different name, substitute the core name you use in the following commands.

From the CORE Generator project directory window, type the following to implement the GEMAC example design:

Windows

```
ms-dos> cd gig_eth_mac_v8_5\implement
ms-dos> implement.bat
```

Linux

```
% cd gig_eth_mac_v8_5/implement
% ./implement.sh
```

These commands execute a script that synthesizes, builds, maps, and place-and-routes the example design together with the core netlist. The script also generates a gate-level model of the example design and netlist for use in timing simulation. The resulting files are placed in the results directory, which is created by the implement script at runtime.

Simulating the Example Design

Functional Simulation

To run the functional 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 the simulation examples that follow, *<project_dir>* is the CORE Generator project directory and *gig_eth_mac_v8_5* is the component name as generated by default from the core customization screen. If a core has been generated with a different name, substitute that core name in the sections that follow.

VHDL Simulation

To run a VHDL functional simulation using Mentor ModelSim

1. Launch the ModelSim simulator and set the current directory to
   <project_dir>/gig_eth_mac_v8_5/simulation/functional
2. Map the UniSim library:
   ```
   ModelSim> vmap unisim <path to compiled libraries>/unisim
   ```
3. Launch the simulation script:
   ```
   ModelSim> do simulate_mti.do
   ```

To run a VHDL functional simulation using Cadence IUS

1. Open a command prompt or shell in your project directory, then set the current directory to:

   <project_dir>/gig_eth_mac_v8_5/simulation/functional

2. Launch the simulation script:

   ./simulate_ncsim.sh

The scripts compile the structural VHDL model for the core, the example design HDL files and the demonstration test bench. It adds some relevant signals to a wave window, then runs the simulation to completion. At this point, you can review the simulation transcript and waveform to observe the operation of the core.

Verilog Simulation

To run a Verilog functional simulation using Mentor ModelSim

1. Launch the ModelSim simulator and set the current directory to **<project_dir>/gig_eth_mac_v8_5/simulation/functional**
2. Map the UniSim library:

   ```
   ModelSim> vmap unisims_ver <path to compiled libraries>/unisims_ver
   ```

3. Launch the simulation script:

   ```
   ModelSim> do simulate_mti.do
   ```

To run a Verilog functional simulation using Cadence IUS

1. Open a command prompt or shell in your project directory, then set the current directory to:

   <project_dir>/gig_eth_mac_v8_5/simulation/functional

2. Launch the simulation script:

   ./simulate_ncsim.sh

To run a Verilog functional simulation using Synopsys VCS

1. Open a command prompt or shell in your project directory, then set the current directory to:

   <project_dir>/gig_eth_mac_v8_5/simulation/functional

2. Launch the simulation script:

   ./simulate_vcs.sh

The scripts compile the structural Verilog model for the core, the example design HDL files and the demonstration test bench. It adds some relevant signals to a wave window, then runs the simulation to completion. At this point, you can review the simulation transcript and waveform to observe the operation of the core.

Timing Simulation

Note: If the core is generated with a Simulation Only license, the timing simulation feature of the example design is not available. In this case, proceed to "What's Next?," on page 23.

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

In the simulation examples that follow, *<project_dir>* is the CORE Generator project directory; *gig_eth_mac_v8_5* is the component name as generated by default from the core customization screen. If a core has been generated with a different name, substitute the core name in the following commands.

VHDL Simulation

To run a VHDL timing simulation using Mentor ModelSim

1. Launch the ModelSim simulator and set the current directory to **<project_dir>/gig_eth_mac_v8_5/simulation/timing**
2. Map the SimPrim library:

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

3. Launch the simulation script:

   ```
   ModelSim> do simulate_mti.do
   ```

To run a VHDL timing simulation using Cadence IUS

1. Open a command prompt or shell in your project directory, then set the current directory to:

   <project_dir>/gig_eth_mac_v8_5/simulation/timing

2. Launch the simulation script:

   ```
   ./simulate_ncsim.sh
   ```

The scripts compile the gate-level model of the example design, generated in the implementation stage, and the demonstration test bench. It adds some relevant signals to a wave window, then runs the simulation to completion. At this point, you can review the simulation transcript and waveform to observe the operation of the core.

Verilog Simulation

To run a Verilog timing simulation using Mentor ModelSim

1. Launch the ModelSim simulator and set the current directory to **<project_dir>/gig_eth_mac_v8_5/simulation/timing**
2. Map the SimPrim library:

   ```
   ModelSim> vmap simprims_ver <path to compiled
   libraries>/simprims_ver
   ```

3. Launch the simulation script:

   ```
   ModelSim> do simulate_mti.do
   ```

To run a Verilog timing simulation using Cadence IUS

1. Open a command prompt or shell in your project directory, then set the current directory to:

   `<project_dir>/gig_eth_mac_v8_5/simulation/timing`

2. Launch the simulation script:

   `./simulate_ncsim.sh`

To run a Verilog timing simulation using Synopsys VCS

1. Open a command prompt or shell in your project directory, then set the current directory to:

   `<project_dir>/gig_eth_mac_v8_5/simulation/timing`

2. Launch the simulation script:

   `./simulate_vcs.sh`

The scripts compile the gate-level model of the example design, generated in the implementation stage, and the demonstration test bench. It adds some relevant signals to a wave window, and then runs the simulation to completion. At this point, you can review the simulation transcript and waveform to observe the operation of the core.

What's Next?

For more information about the example design, including guidelines on modifying the design and extending the test bench, see Chapter 4, "Detailed Example Design." To begin using the GEMAC core in your own design, see the *1-Gigabit Ethernet MAC User Guide.*

XILINX®

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.

<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

example_design/fifo
FIFO directory, contains files for the FIFO instanced in LocalLink example design

example_design/physical
Files for the physical interface of the MAC

<component name>/implement
Implementation script files

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

<component name>/simulation
Simulation scripts

simulation/functional
Functional simulation files

simulation/timing
Timing simulation files

Directory and File Contents

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

Note: The implement and timing simulation directories are only present when the core is generated with a Full System Hardware Evaluation license or Full license.

<project directory>

The project directory contains all the CORE Generator project files.

Table 4-1: **Project Directory**

Name	Description
<project_dir>	
<component_name>.ngc	Binary Xilinx implementation netlist. Describes how the core is to be implemented. Used as input to the Xilinx™ Implementation Tools.
<component_name>.v[hd]	VHDL or Verilog structural simulation model. File used to support VHDL or Verilog functional simulation of a core. The VHDL or Verilog model passes customized parameters to the generic core simulation model.
<component_name>.xco	As an output file, the XCO file is a log file that records the settings used to generate a particular 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>_flist.txt	Text file listing all of the output files produced when customized core was generated in the CORE Generator.
<component_name>.{vho|veo}	VHDL or Verilog template for the core, which can be copied into your design.

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_mac_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_mac_ds200.pdf	*1-Gigabit Ethernet MAC Data Sheet*
gig_eth_mac_gsg143.pdf	*1-Gigabit Ethernet MAC Getting Started Guide*
gig_eth_mac_ug144.pdf	*1-Gigabit Ethernet MAC User Guide*

Back to Top

<component name>/example_design

The example design directory contains the example design files provided with the core. See "Example Design," page 35.

Table 4-4: **Example Design Directory**

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_example_design.v[hd]	Top-level VHDL or Verilog file for the example design. This instantiates the LocalLink block along with the address swap block, providing a simple loopback function.
<component_name>_example_design.ucf	User constraints file (UCF) for the core and the example design.
<component_name>_locallink.v[hd]	Example design with a LocalLink client interface. This instantiates the block level GEMAC wrapper together with a receive and a transmit FIFO.
<component_name>_block.v[hd]	Block-level GEMAC wrapper containing the core and all clocking and physical interface circuitry.
<component_name>_mod.v	Verilog module declaration for the core instance in the block level example design.
address_swap_module.v[hd]	Top-level example design instances this to swap the source and destination addresses of the incoming frames.

Back to Top

example_design/fifo

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

Table 4-5: **FIFO Directory**

Name	Description
<project_dir>/<component_name>/example_design/fifo	
tx_client_fifo.v[hd]	Transmit client FIFO. This takes data from the client in LocalLink format, stores it and sends it to the MAC.
rx_client_fifo.v[hd]	Receive client FIFO. This reads in and stores data from the MAC before outputting it to the client in LocalLink format.
ten_100_1g_eth_fifo.v[hd]	FIFO top level. This instantiates the transmit and receive client FIFOs.

Back to Top

example_design/physical

This directory contains a file for the physical interface of the MAC. A GMII or RGMII version will be delivered by CORE Generator depending on the selected option.

Table 4-6: **Physical Directory**

Name	Description
<project_dir>/<component_name>/example_design/physical	
gmii_if.v[hd]	For GMII only: all clocking and logic required to provide a GMII physical interface.
rgmii_v2_0_if.v[hd]	For RGMII only: all clocking and logic required to provide a RGMII v2.0 physical interface.
sync_block.v[hd]	Only present when the dcm_reset module is also present: this is a synchronization flip-flop pair, for crossing signals across a clock domain.
reset_sync.v[hd]	Only present when the dcm_reset module is also present: this is a reset synchronization module, for creating a synchronous reset output signal from an asynchronous input.

Table 4-6: **Physical Directory *(Continued)***

Name	Description
dcm_reset.v[hd]	Only present when a DCM is used (Virtex®-4 devices and all Spartan®-3 devices). This is a self contained module for resetting a DCM following Power-on-Reset or loss of lock.

Back to Top

<component name>/implement

This directory contains the support files necessary for implementation of the example design with the XILINX tools. See "Example Design," page 35. Execution of an implement script results in creation of the results directory and an xst project directory.

Table 4-7: **Implement Directory**

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

Back to Top

implement/results

This directory is created by the implement scripts and is used to run the example design files and the `<component_name>.ngc` file through the Xilinx implementation tools. On completion of an implement script, this directory contains the following files for timing simulation. Output files from the Xilinx implementation tools are also located in this directory.

Table 4-8: **Results Directory**

Name	Description
`<project_dir>/<component_name>/implement/results`	
`routed.v[hd]`	The back-annotated SimPrim based gate-level VHDL or Verilog model. Used for timing simulation.
`routed.sdf`	Timing information for simulation.

Back to Top

<component name>/simulation

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

Table 4-9: **Simulation Directory**

Name	Description
`<project_dir>/<component_name>/simulation`	
`demo_tb.v[hd]`	VHDL or Verilog demonstration test bench for the GEMAC core.

Back to Top

simulation/functional

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

Table 4-10: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_mti.do	ModelSim macro file that compiles the example design sources and the structural simulation model, then runs the functional simulation to completion.
wave_mti.do	ModelSim macro file that opens a wave window and adds interesting signals to it. It is called by the simulate_mti.do macro file.
simulate_ncsim.sh	Linux shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using Cadence IUS.
wave_ncsim.sv	IUS macro file that opens a wave window and adds interesting signals to it. It is used by the simulate_ncsim.sh script.
simulate_vcs.sh	VCS script file that compiles the Verilog sources and runs the functional simulation to completion.
vcs_commands.key	This file is sourced by VCS at the start of simulation: it configures the simulator.
vcs_session.tcl	VCS macro file that opens a wave window and adds signals of interest to it. It is called by the simulate_vcs.sh script file

Back to Top

simulation/timing

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

Table 4-11: **Timing Directory**

Name	Description
<project_dir>/<component_name>/simulation/timing	
simulate_mti.do	ModelSim macro file that compiles the VHDL gate-level model of the example design and demo test bench, then runs the timing simulation to completion.
wave_mti.do	ModelSim macro file that opens a wave window and adds interesting signals to it. It is called by the simulate_mti.do macro file.
simulate_ncsim.sh	Linux shell script that compiles the example design sources and the VHDL gate-level model, then runs the timing simulation to completion using Cadence IUS.
wave_ncsim.sv	IUS macro file that opens a wave window and adds interesting signals to it. It is used by the simulate_ncsim.sh script.
simulate_vcs.sh	VCS script file that compiles the Verilog sources and runs the timing simulation to completion.
vcs_commands.key	This file is sourced by VCS at the start of simulation: it configures the simulator.
vcs_session.tcl	VCS macro file that opens a wave window and adds signals of interest to it. It is called by the simulate_vcs.sh script file

Back to Top

Implementation and Test Scripts

Implementation Scripts for Timing Simulation

When CORE Generator has been run with a Full-system Evaluation License or a Full License an implement script is generated in the `<project_dir>/<component_name>/implement` directory. The implementation script is either a shell script or batch file that processes the example design through the Xilinx tool flow.

Linux

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

Windows

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

The implement script performs the following steps:

1. The HDL example design is synthesised using XST.
2. Ngdbuild is run to consolidate the core netlist and the HDL example 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 VHDL or Verilog gate-level models and timing information in the form of SDF files.

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

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

Test Scripts For Functional Simulation

The functional simulation flow is available no matter which license type has been used by CORE Generator. The test script that automates the simulation of the test bench is located at:

Mentor ModelSim

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

Cadence IUS

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

Synopsys VCS

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

The test script performs the following tasks:

1. Compiles the structural simulation model of the core.
2. Compiles the example design files.
3. Compiles the demonstration test bench.
4. Starts a simulation of the test bench with no timing information.
5. Opens a Wave window and adds some signals of interest.
6. Runs the simulation to completion.

Test Scripts For Timing Simulation

When the CORE Generator has been run with a Full System Hardware Evaluation license or Full license, a test script for running timing simulation is generated. The test script that automates the simulation of the test bench is located at:

Mentor ModelSim

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

Cadence IUS

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

Synopsys VCS

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

The test script performs the following tasks:

1. Compiles the gate-level model of the example design.
2. Compiles the demonstration test bench.
3. Starts a simulation of the test bench using timing information.
4. Opens a Wave window and adds some signals of interest.
5. Runs the simulation to completion.

Example Design

HDL Example Design

Figure 4-1 illustrates the top-level design for the GEMAC core example design.

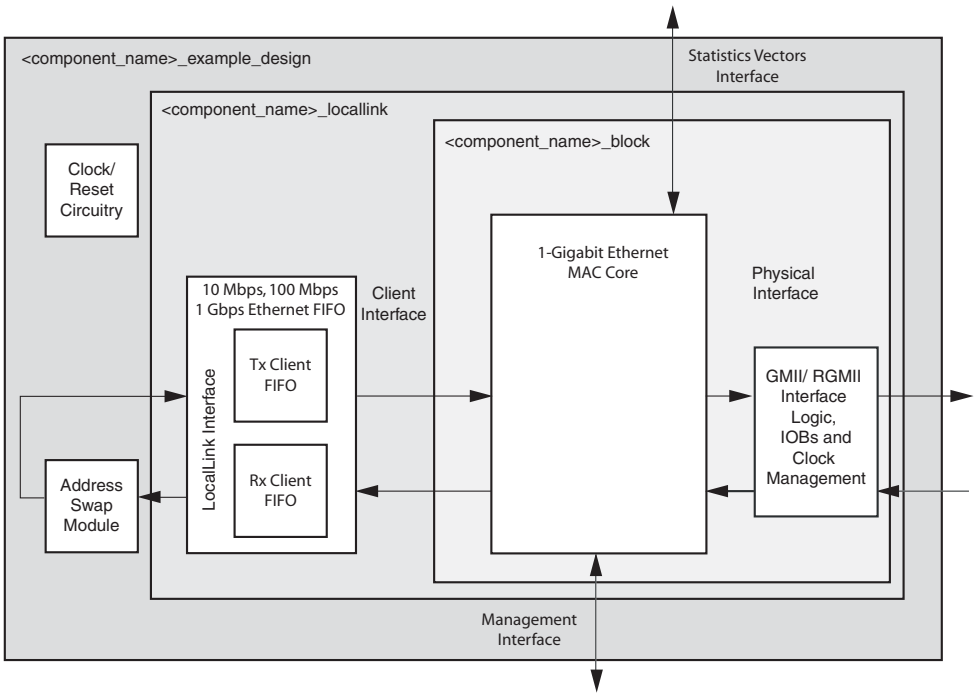

Figure 4-1: **Example Design HDL Wrapper**

The top-level example design for the GEMAC 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 HDL example design contains the following:

- An instance of the GEMAC core
- Clock management logic, including DCM, DCM reset circuitry, and Global Clock Buffer instances, where required
- GMII or RGMII interface logic, including IOB and DDR registers instances, where required
- Client Transmit and Receive FIFOs with a LocalLink interface
- Client LocalLink loopback module that performs address swapping

The HDL example design provides client loopback functionality on the client side of the GEMAC core and connects the GMII/RGMII 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.

10M/100M/1G Ethernet FIFO

The 10 Mbps/100 Mbps/1 Gbps Ethernet FIFO is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/fifo/ten_100_1g_eth_fifo.vhd
<project_dir>/<component_name>/example_design/fifo/tx_client_fifo.vhd
<project_dir>/<component_name>/example_design/fifo/rx_client_fifo.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/fifo/ten_100_1g_eth_fifo.v
<project_dir>/<component_name>/example_design/fifo/tx_client_fifo.v
<project_dir>/<component_name>/example_design/fifo/rx_client_fifo.v
```

For a full description of the 10 Mbps/100 Mbps/1 Gbps Ethernet FIFO, see Appendix A, "Using the Client Side FIFO" in the *1-Gigabit Ethernet MAC User Guide*.

See also direct.xilinx.com/bvdocs/appnotes/xapp691.pdf for a detailed description of the LocalLink interface.

The 10 Mbps/100 Mbps/1 Gbps Ethernet FIFO contains an instance of `tx_client_fifo` to connect to the MAC client side transmitter interface, and an instance of the `rx_client_fifo` to connect to the MAC client receiver interface, via the address swap module. 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 a LocalLink interface.

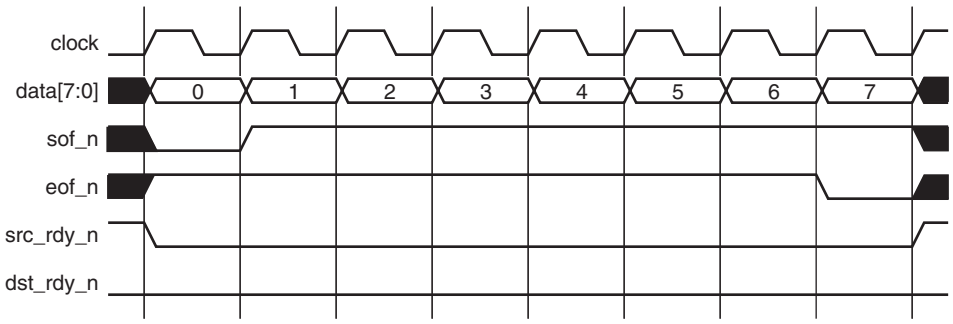

Figure 4-2: **Frame Transfer across LocalLink Interface**

rx_client_fifo

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

If the receive FIFO memory overflows, the frame currently being received is dropped, regardless of whether it is a good or bad frame, and the signal `rx_overflow` is asserted.

Situations in which the memory may overflow are:

- The FIFO may overflow if the receiver clock is running at a faster rate than the transmitter clock or if the interpacket gap between the received frames is smaller than the interpacket 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 then the FIFO may overflow and data will be lost. It is therefore recommended that the example design is not used with the GEMAC in jumbo frame mode for frames of larger than 4000 bytes.

tx_client_fifo

The `tx_client_fifo` is built around two Dual Port block RAMs, providing a total memory capacity of 4096 bytes.

When a full frame has been written into the transmit FIFO, the FIFO presents data to the MAC transmitter. On receiving the `tx_ack` signal from the MAC, the rest of the frame is transmitted to the MAC.

If the FIFO memory fills up, the `dst_rdy_out_n` signal is used to stop the LocalLink interface from writing further 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 asserts the `tx_overflow` signal and continues to accept the rest of the frame. The overflow frame is then dropped by the FIFO, ensuring that the LocalLink interface does not lock up.

VHDL

The generic `FULL_DUPLEX_ONLY` is provided to allow removal of logic and performance constraints necessary only in half-duplex operation; that is, when the FIFO is used with the Tri-Mode Ethernet MAC (TEMAC) core. This generic can always be set to *true* when the FIFO is used with the GEMAC.

Verilog

The compiler directive `FULL_DUPLEX_ONLY` is defined to allow removal of logic and performance constraints necessary only in half-duplex operation; that is, when the FIFO is used with the Tri-Mode Ethernet MAC (TEMAC) core. This directive can always be defined when the FIFO is used with the GEMAC.

Address Swap Module

The address swap module is described in the following files:

VHDL

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

Verilog

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

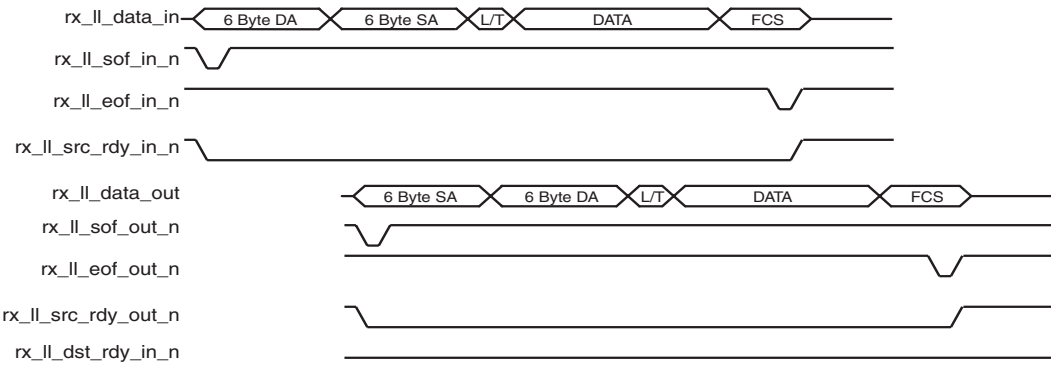

Figure 4-3: **Modification of Frame Data by Address Swap Module**

The address swap module takes frame data from the GEMAC receiver client interface. As illustrated in Figure 4-3, the module swaps the destination address (DA) and source address (SA) of each frame, which ensures 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.

Demonstration Test Bench

Test Bench Functionality

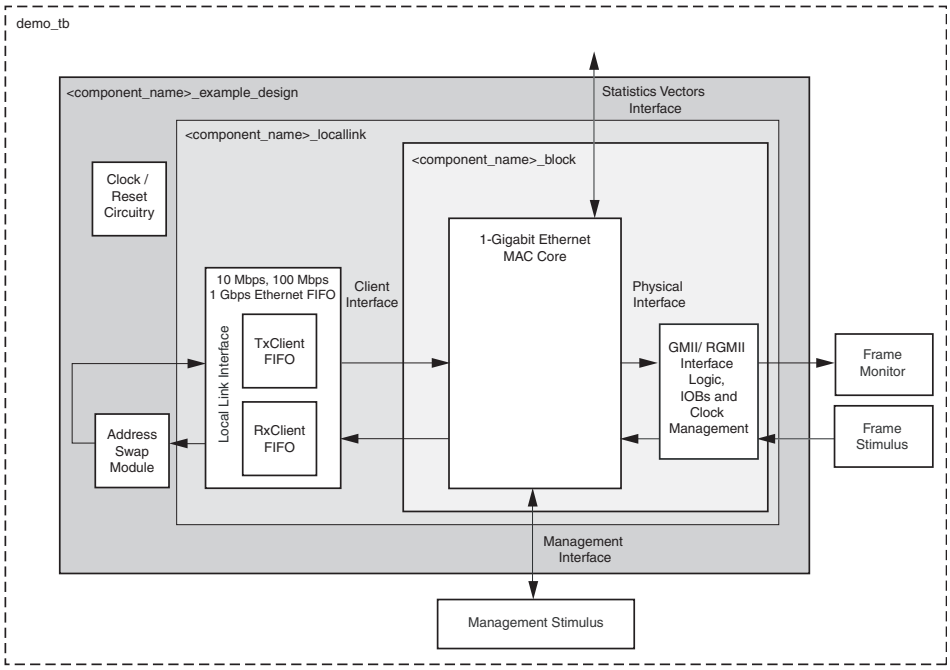

Figure 4-4: **Demonstration Test Bench**

The demonstration test bench is described in the following files:

VHDL

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

Verilog

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

The demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself.

The test bench consists of the following:

- Clock generators
- A stimulus block that connects to the GMII/ RGMII receiver interface of the example design
- A monitor block to check data returned through the GMII/RGMII transmitter interface
- A management block to exercise the Management Interface, if selected
- A control mechanism to manage the interaction of management, stimulus and monitor blocks

Core with Management Interface

The demonstration test bench performs the following tasks:

1. Input clock signals are generated.
2. A reset is applied to the example design.
3. The GEMAC core is configured through the Management Interface, setting up the MDC clock frequency, disabling flow control, and if the Address Filter is selected, writing to the Unicast Address registers and enabling the Address Filter.
4. The stimulus block pushes four frames into the GMII/RGMII receiver interface:
   + 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
5. The frames received at the GMII/RGMII transmitter interface are checked against the stimulus frames to ensure data is the same. The monitor process takes into account the source/destination address field and FCS modifications resulting from the address swap module.

Core with No Management Interface

Because no Management Interface is present, the configuration of the GEMAC core is controlled by hard wiring of the configuration vector to required logic levels. See the *1-Gigabit Ethernet MAC User Guide* for more information about the configuration vector.

The demonstration test bench performs the following tasks:

1. Input clock signals are generated
2. A reset is applied to the example design
3. The stimulus block pushes four frames into the GMII/RGMII receiver interface:
   + 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
4. The frames received at the GMII/RGMII transmitter interface are checked against the stimulus frames to ensure data is the same. The monitor process takes into account the source/destination address field and FCS modifications resulting from the address swap module.

Changing the Test Bench

Changing Frame Data

You can change the contents of the frame data passed into the GEMAC receiver by changing the data fields for each frame defined in the test bench. The test bench will automatically calculate the new FCS field to pass into the GEMAC, as well as calculating the new expected FCS value.

Further frames can be added by defining a new frame of data. Care should be taken to update the expected statistics values if the statistics option has been chosen.

Changing Frame Error Status

Errors can be inserted into any of the predefined frames in the following way:

- A `gmii_rx_er` signal can be asserted 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 to disable the monitor checking for that frame. If statistics option has been chosen, the expected statistics values also need to be modified accordingly.

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 GEMAC Configuration

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

> ***Caution!*** Certain configurations of the GEMAC cause the test bench to result in failure or to cause processes to run indefinitely. You must determine the configurations that can safely be used with the test bench.

If the Management Interface option is selected, the GEMAC can be reconfigured by adding further steps in the test bench management process to write new configurations to the GEMAC. See the *1-Gigabit Ethernet MAC User Guide* for more information about the Management Interface.

If the Management Interface option is not selected, the GEMAC can be reconfigured by modifying the configuration vector directly. See the *1-Gigabit Ethernet MAC User Guide* for information about the configuration vector.