LogiCORE™ IP RXAUI v1.2

Getting Started Guide

UG694 April 19, 2010

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/16/09	1.1	Initial Xilinx release.
4/19/2010	1.2	Updated to v1.2 for 12.1 release; added Marvell RXAUI specification support.

Table of Contents

Chapter 4: Detailed Example Design

About This Guide

The *RXAUI Getting Started Guide* provides information about generating a LogiCORE™ IP RXAUI core, customizing and simulating the core utilizing the provided example design, and running the design files through implementation using the Xilinx tools.

Guide Contents

This guide contains the following chapters:

- Preface, "About this Guide," introduces the organization and purpose of the design guide, provides links to additional resources, and describes the conventions used in this document.
- Chapter 1, "Introduction" introduces the RXAUI core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Licensing the Core" provides instructions for installing and obtaining a license for the core, which must be completed before using the core in your designs.
- Chapter 3, "Quick Start Example Design" provides instructions for generating a core using the default configuration, implementing the example design, and simulating the core.
- Chapter 4, "Detailed Example Design" provides detailed information about the example design, including the directory structure and associated files, as well as how to modify the design and the associated tests for your applications.

Additional Resources

To find additional documentation, see the Xilinx website at:

www.xilinx.com/support/documentation/index.htm.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

www.xilinx.com/support/mysupport.htm.

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 that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *User Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`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`**}
Angle brackets < >	User-defined variable for directory names or in code samples	<directory name>

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Convention	Meaning or Use	Example
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	**usr_teof_n** is active low.

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "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	Definition
DVE	Discovery Visualization Environment
FPGA	Field Programmable Gate Array.
HDL	Hardware Description Language
IP	Intellectual Property
ISE®	Integrated Software Environment
IUS	Incisive Unified Simulator
MMCM	Mixed Code Clock Manager
NGD	Native Generic Database
RXAUI	Reduced Pin eXtended Attachment Unit Interface
SDF	Standard Delay Format
UCF	User Constraints File
VCS	Verilog Compiled Simulator (Synopsys)
VHDL	VHSIC Hardware Description Language (VHSIC an acronym for Very High-Speed Integrated Circuits).
XCO	Xilinx CORE Generator™ software core source file
XGMII	10-Gigabit Ethernet Media Independent Interface
XST	Xilinx Synthesis Technology

XILINX®

Chapter 1

Introduction

The RXAUI core is a fully-verified solution that supports both Verilog and VHDL. In addition, the example design in this guide is provided in both Verilog and VHDL.

This chapter introduces the RXAUI 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® software v12.1

About the Core

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

Recommended Design Experience

Although the RXAUI 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 constraints files (UCF) is recommended.

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

Additional Core Resources

For detailed information about RXAUI technology and updates to the RXAUI core, see the following sections.

Documentation

From the RXAUI product page:

- *RXAUI Data Sheet*
- *RXAUI Getting Started Guide*

From the /doc directory after generating the RXAUI core:

- *RXAUI Release Notes*
- RXAUI User Guide

RXAUI Technology

For information about RXAUI technology basics, including features, FAQs, the RXAUI chip interface, typical applications, specifications, and other important information, see www.xilinx.com/products/ipcenter/RXAUI.htm.

Ethernet Specifications

Relevant XAUI IEEE standards, which can be downloaded in PDF format from standards.ieee.org/getieee802/:

- *IEEE Std. 802.3-2008*
- The Dune Networks RXAUI standard can be obtained directly from Dune Networks.
- The Marvell RXAUI standard can be obtained directly from Marvell.

Technical Support

For technical support, visit www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the RXAUI core.

Xilinx provides technical support for use of this product as described in the *LogiCORE™ IP RXAUI User Guide* and the *LogiCORE IP RXAUI 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 RXAUI core and the documentation supplied with the core.

Core

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

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

Document

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

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

XILINX®

Chapter 2

Licensing the Core

This chapter provides instructions for installing the RXAUI core and obtaining a license for the core, which you must do before using the core in your designs. The RXAUI core is provided under the terms of the Xilinx End User 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 all required software specified on the product page for this core.

License Options

The RXAUI core provides two 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™ tool. This key lets you assess core functionality with either the example design provided with the RXAUI 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

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

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

Obtaining Your License Key

This section contains information about obtaining a license key for both licensing options.

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.

Obtaining a Full License Key

To obtain a Full license key:

1. Navigate to the product page for this core: www.xilinx.com/products/ipcenter/RXAUI.htm.
2. Click **Get License**.
3. Follow the instructions to install the required Xilinx ISE software and IP updates and generate a Full license key.

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 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 provides instructions for generating a core using the default configuration, implementing the example design, and simulating your design using Mentor Graphics ModelSim v6.5c, Cadence IUS v9.2, and Synopsys 2009.12.

Introduction

Figure 3-1 illustrates the default configuration of the example design.

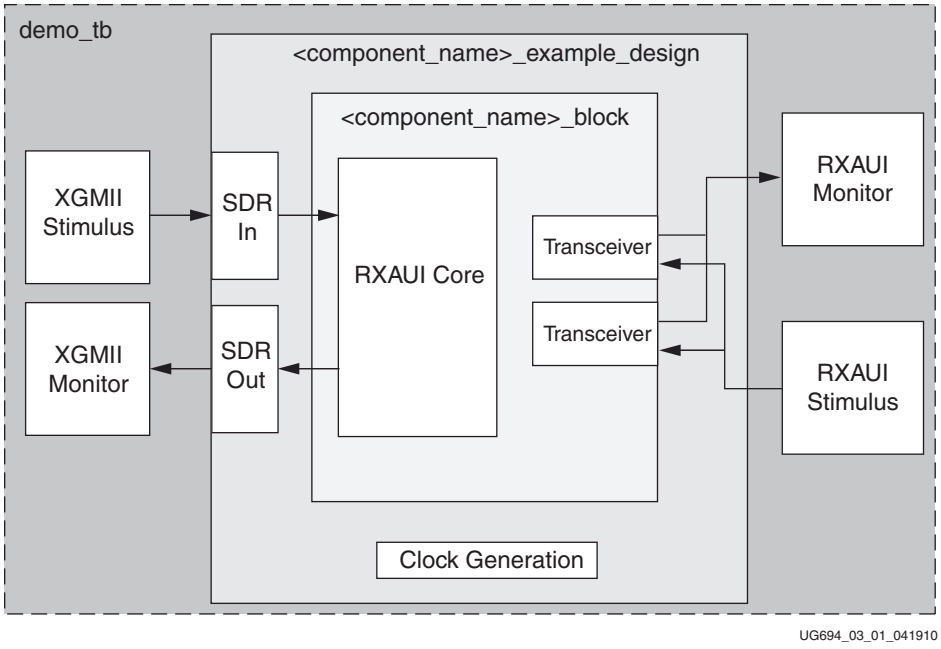

Figure 3-1: **RXAUI Example Design and Test Bench: Default Configuration**

The RXAUI example design consists of the following:

- A RXAUI core netlist
- Transceiver wrappers
- An example HDL wrapper
- A demonstration test bench to exercise the example design

The RXAUI Design Example has been tested with Xilinx® ISE® software v12.1, Mentor Graphics ModelSim v6.5c, Cadence IUS v9.2, and Synopsys 2009.12.

Generating the Core

To generate a RXAUI core with default values using the CORE Generator™ software do the following:

1. Start the CORE Generator software.

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

2. Choose File > New Project.
3. Type a directory name.
4. Do the following to set project options:
   - From the Part tab, select a silicon family, part, speed grade, and package that supports the RXAUI core, for example, Virtex®-6 FPGAs.

     Note: If an unsupported silicon family is selected, the RXAUI core does not appear in the taxonomy tree. For a list of supported architectures, see the *RXAUI Data Sheet*.
   - From the Generation tab, select VHDL or Verilog; for Vendor, select Other.
   - On the Advanced tab, accept the default values.
5. After creating the project, locate the core in the taxonomy tree at the left side of the CORE Generator software window. The RXAUI core appears under the following categories:
   - Communications & Networking/Ethernet
   - Communications & Networking/Networking
   - Communications & Networking/Telecommunications
6. Double-click the core to open it. A message may appear warning you about the limitations of the Simulation Only license, and then the RXAUI customization screen appears.
7. In the Component Name field, enter a name for the core instance.
8. Accept the remaining default options and click Finish to generate the core.

   The core and its supporting files, including the example design, are generated in the project directory. For a detailed description of the directory structure and files, see "Directory and File Contents" in Chapter 4.

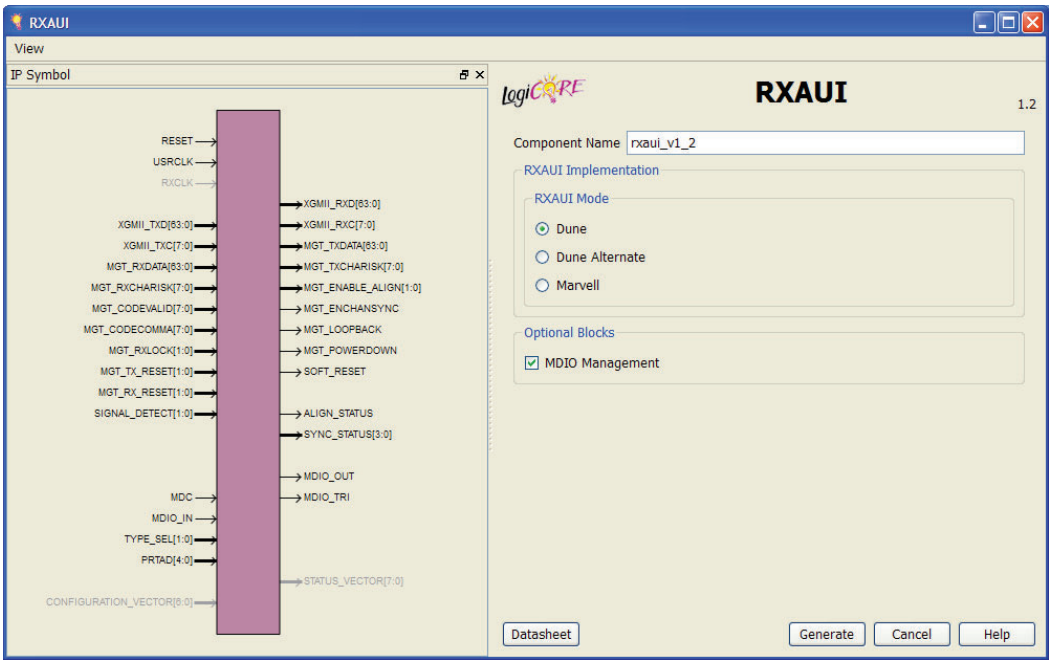

Figure 3-2: **RXAUI Main Screen**

Implementing the RXAUI Example Design

If the core is generated with a Simulation Only license, the implementation feature of the example design is not available; in this case, go directly to Chapter 3, "Simulating the RXAUI Example Design."

After the core is successfully generated, the netlist and example design HDL wrapper can be processed through the Xilinx implementation tools. The generated outputs include several scripts to assist in processing.

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

Linux

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

Windows

```
> cd <component_name>\implement
> implement.bat
```

The implement command accomplishes the following:

- Starts a script to synthesize the example design HDL wrapper
- Builds, maps, and place-and-routes the example design (Full license only)
- Creates gate-level netlist HDL files in both VHDL and Verilog with associated timing information (SDF files)

The created files are placed in the results directory which is created by the implement script at runtime.

Simulating the RXAUI Example Design

The example design provided with the RXAUI core provides a complete environment which allows you to simulate the core and view the outputs. Scripts are provided for pre- and post-layout simulation. The simulation model is either in VHDL or Verilog depending on the CORE Generator software Design Entry project option.

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.

The Xilinx simulation libraries must be mapped into the simulator. If the libraries are not set for your environment, go to Answer Record 15338 on www.xilinx.com/support for assistance compiling Xilinx simulation models and setting up the simulator environment.

All Virtex family designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. For a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator, the following simulators are supported.

- Mentor Graphics ModelSim v6.5c and above
- Cadence IUS 9.2 and above
- Synopsys 2009.02 and above

Pre-Implementation Simulation

To run a functional simulation of the example design:

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

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

2. Launch the simulation script:

   ```
   ModelSim: vsim -do simulate_mti.do
   Cadence sim: ./simulate_ncsim.sh
   vcs: ./simulate_vcs.sh
   ```

The simulation script compiles the functional model and the demonstration test bench, adds some 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.

Post-Implementation Simulation

To run a timing simulation of the example design:

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

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

2. Launch the simulation script:

   ```
   modelSim: vsim -do simulate_mti.do
   ncsim: ./simulate_ncsim.sh
   vcs: ./simulate_vcs.sh
   ```

The simulation script compiles the gate-level model and the demonstration test bench, adds some 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.

Additional Information

For more information about the example design, including guidelines for modifying the design and extending the test bench, see Chapter 4, "Detailed Example Design." To start using the RXAUI core in your own design, see the *RXAUI User Guide*.

XILINX®

Chapter 4

Detailed Example Design

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

- **<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 sections.

<project directory>

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

Table 4-1: **Project Directory**

Name	Description
<project_dir>	
<component_name>.ngc	A binary Xilinx implementation netlist. Describes how the core is to be implemented. Used as an input to the Xilinx implementation tools.
<component_name>.v[hd]	VHDL or Verilog structural simulation model. File used to support functional simulation of a core.
<component_name>.xco	As an output file, the XCO file is a log file which records the settings used to generate a particular core. An XCO file is generated by the CORE Generator software for each core that it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator software.
<component_name>_flist.txt	List of files delivered with the core
<component_name>.{veo|vho}	A VHDL or Verilog template for the core. This 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>	
rxaui_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	
rxaui_ds740.pdf	RXAUI Data Sheet
rxaui_gsg694.pdf	RXAUI Getting Started Guide
rxaui_ug693.pdf	RXAUI User Guide

Back to Top

<component name>/example_design

The example design directory contains the example design files provided with the core.

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

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_block.v[hd]	Block entity containing the RXAUI core and transceiver wrappers
<component_name>_example_design.v[hd]	Top-level entity for the example design containing the block level design and clocking circuitry
component_name>_example_design.ucf	User constraints file for the core and example design
<component_name>_mod.v	Wrapper file for the RXAUI core
chanbond_monitor.v[hd]	Transceiver Channel Bonding Monitor
gtx_wrapper.v[hd] gtx_wrapper_gtx.v[hd]	Wrappers for the transceivers

Back to Top

<component name>/implement

This directory contains the support files necessary for implementation of the example design with the Xilinx tools. Execution of an implement script creates a results directory and an `xst` project directory.

Table 4-5: **Implement Directory**

Name	Description
`<project_dir>/<component_name>/implement`	
`implement.bat`	Windows batch file that process the example design through the Xilinx tool flow
`implement.sh`	Linux shell script that processes the example design through the Xilinx tool flow
`xst.scr`	XST script file for the example design
`xst.prj`	XST project file for the example design
`xst.xcf`	XCF constraint 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 can also be found in this directory.

Table 4-6: **Results Directory**

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

Back to Top

<component name>/simulation

The simulation directory and the subdirectories below it contain the files necessary to test a VHDL or Verilog implementation of the example design.

Table 4-7: **Simulation Directory**

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

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 the example design sources, the structural simulation model and the demonstration test bench then runs the functional simulation to completion.
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 the Cadence IUS simulator.
simulate_vcs.sh (verilog only)	Linux shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using VCS.
ucli_commands.key (verilog only)	VCS command file. This file is called by the simulate_vcs.sh script.
vcs_session.tcl (verilog only)	VCS DVE tcl script that opens wave windows and adds interesting signals to it. This macro is used by the simulate_vcs.sh script.
wave_mti.do	ModelSim macro file that opens a wave window and adds interesting signals to it. This macro is called by the simulate_mti.do macro file.
wave_ncsim.sv	The Cadence IUS simulator macro file that opens a wave windows and adds interesting signals to it. This macro is called by the simulate_ncsim.sh script.

Back to Top

simulation/timing

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

Table 4-9: **Timing Directory**

Name	Description
<project_dir>/<component_name>/simulation/timing	
`simulate_mti.do`	ModelSim macro file that compiles the timing simulation model and the demonstration test bench then runs the timing simulation to completion.
`simulate_ncsim.sh`	Linux shell script that compiles the test bench and the timing model then runs the timing simulation to completion using the Cadence IUS simulator.
`simulate_vcs.sh`	Linux shell script that compiles the example design sources and the timing simulation model then runs the timing simulation to completion using VCS.
`ucli_commands.key (verilog only)`	VCS command file. This file is called by the simulate_vcs.sh script.
`vcs_session.tcl (verilog only)`	VCS DVE tcl script that opens wave windows and adds interesting signals to it. This macro is called by the simulate_vcs.sh script.
`wave_mti.do`	ModelSim macro file that opens a wave window and adds interesting signals to it. This macro is called by the `simulate_mti.do` macro file.
`wave_ncsim.sv`	The Cadence IUS simulator macro file that opens a wave windows and adds interesting signals to it. This macro is called by the `simulate_ncsim.sh` script.

Back to Top

Implementation and Test Scripts

Implementation Script

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

Linux

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

Windows

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

The implement script performs the following steps:

- The example HDL wrapper is synthesized using XST.
- ngdbuild is run to consolidate the core netlist and the wrapper netlist into the NGD file containing the entire design.
- The design is mapped to the target technology.
- The design is place-and-routed on the target device.
- Static timing analysis is performed on the routed design using trce.
- A bitstream is generated.
- netgen runs on the routed design to generate VHDL and Verilog netlists and timing information in the form of SDF files.

The implement script is only generated when Full license is available for the RXAUI core.

Setting up for Simulation

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

All Virtex family designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. For a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator, the following simulators are supported.

- Mentor Graphics ModelSim v6.5c and above
- Cadence IUS v9.2 and above
- Synopsys 2009.12 and above

Simulation Scripts

Simulation macro files are provided for ModelSim and shell scripts are provided for the Cadence IUS simulator and Synopsys VCS simulator. The scripts automate the simulation of the test bench and can be found in the following location:

Functional

```
<project_dir>/<component_name>/simulation/functional/simulate_mti.do
<project_dir>/<component_name>/simulation/functional/simulate_ncsim.sh
<project_dir>/<component_name>/simulation/functional/simulate_vcs.sh
```

Timing

```
<project_dir>/<component_name>/simulation/timing/simulate_mti.do
<project_dir>/<component_name>/simulation/timing/simulate_ncsim.sh
<project_dir>/<component_name>/simulation/timing/simulate_vcs.sh
```

The scripts perform the following tasks:

- Compiles the gate level netlist
- Compiles the demonstration test bench
- Starts a simulation of the test bench (with timing information if a Full-system Evaluation license or Full license is in use)
- Opens a Wave window and adds some interesting signals (`wave_mti.do/wave_ncsim.sv/vcs_session.tcl`)
- Runs the simulation to completion

RXAUI Core with Internal Client-Side Interface

Example HDL Wrapper

In Figure 4-1, the example HDL wrapper generated when the internal client-side interface is selected contains the following:

- The Transceiver Instances
- The wrapper for the two transceivers
- A transceiver transmit initialization block
- Clock management logic and Clock Buffer instances
- Re-timing registers on the parallel data interface, both on inputs and outputs

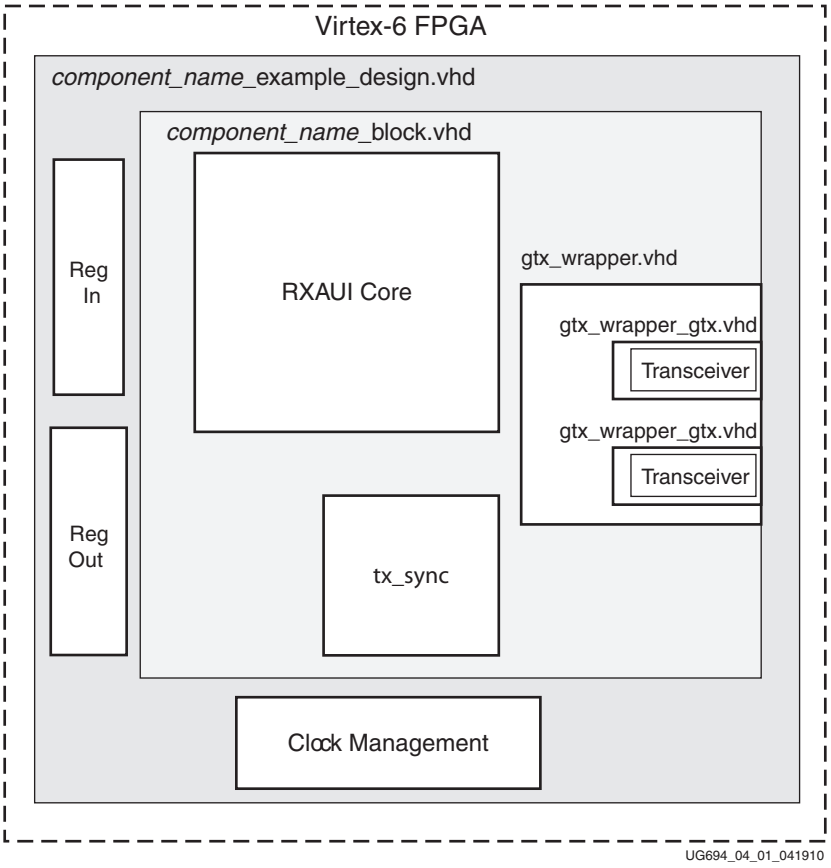

Figure 4-1: **Example HDL Wrapper for RXAUI (Virtex-6 FPGAs)**

Demonstration Test Bench

In Figure 4-2, the demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself. This test bench consists of transactor procedures or tasks that connect to the major ports of the example design, and a control program that pushes frames of varying length and content through the design and checks the values as they exit the core.

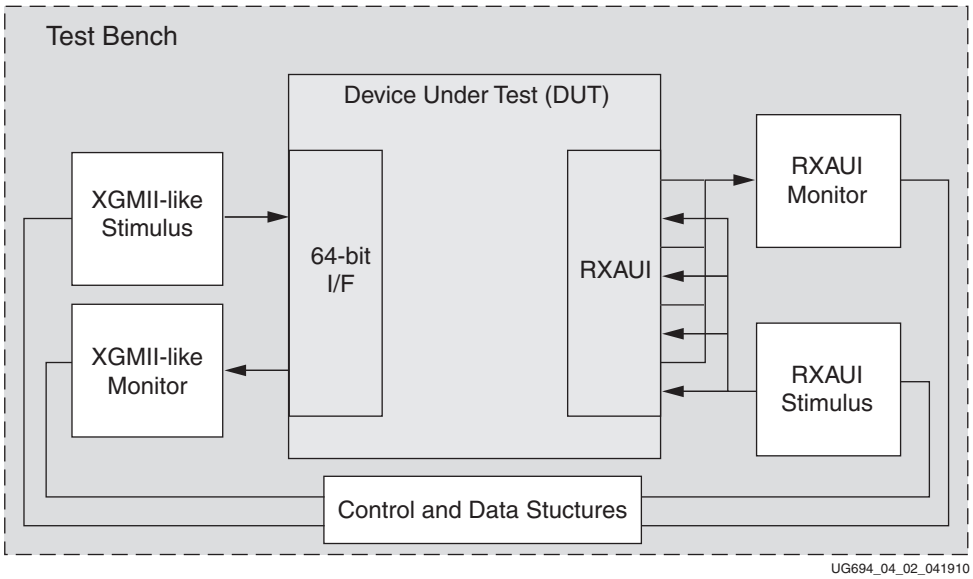

Figure 4-2: **Demonstration Test Bench for RXAUI**