LogiCORE™ XAUI v6.2

Getting Started Guide

UG149 July 13, 2006

Revision History

The following table shows the revision history for this document.

	Version	Revision
9/30/04	1.0	Initial Xilinx release.
04/28/05	1.1	Document updated to support XAUI core v6.0 and Xilinx ISE v7.1i.
01/18/05	1.2	Document updated to support XAUI core v6.1 and Xilinx ISE v8.1i.
7/13/06	1.3	Document updated to support XAUI core v6.2 and Xilinx tools ISE 8.2i.

Table of Contents

Chapter 4: Detailed Example Design

Schedule of Figures

Preface

About This Guide

The *XAUI v6.2 Getting Started Guide* provides information about generating a LogiCORE™ XAUI 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, a list of additional resources, and the conventions used in this document.
- Chapter 1, "Introduction" introduces the XAUI core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Installing and 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 using ModelSim® and NC-Sim.
- 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

For additional information, go to http://support.xilinx.com. The following table lists some of the resources you can access from this website. You can also directly access these resources using the provided URLs.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answer Browser	Database of Xilinx solution records http://support.xilinx.com/xlnx/xil_ans_browser.jsp
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm

Resource	Description/URL
Data Sheets	Device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://support.xilinx.com/xlnx/xweb/xil_publications_index.jsp
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues http://support.xilinx.com/support/troubleshoot/psolvers.htm
Tech Tips	Latest news, design tips, and patch information for the Xilinx design environment http://support.xilinx.com/xlnx/xil_tt_home.jsp

Conventions

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

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands you enter in a syntactical statement	**`ngdbuild`** *`design_name`*
Italic font	Variables in a syntax statement for which you must supply values	See the *Development System Reference Guide* for more information.
	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.
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}`**

Convention	Meaning or Use	Example

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 `45524943h`.
	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 http://www.xilinx.com for the latest speed files.

Chapter 1

Introduction

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

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

About the Core

The XAUI core is a CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center.

For detailed information about the core, see http://www.xilinx.com/systemio/xaui/index.htm.

For information about system requirements, installation, and licensing options, see Chapter 2, "Installing and Licensing the Core."

Recommended Design Experience

Although the XAUI 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 and updates about the XAUI core, see the following documents, located on the XAUI product page at:
http://www.xilinx.com/systemio/xaui/index.htm

- *XAUI Release Notes*
- *XAUI Data Sheet*
- *XAUI User Guide*

For updates to this document, see the *XAUI Getting Started Guide*, also located on the XAUI product page.

Technical Support

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

Xilinx will provide technical support for use of this product as described in the *LogiCORE XAUI User Guide* and the *LogiCORE XAUI 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 XAUI core and the documentation supplied with the core.

Core

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

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

Chapter 2

Installing and Licensing the Core

This chapter provides instructions for installing the XAUI core and obtaining a license for the core, which you must do before using the core in your designs. The XAUI 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.

System Requirements

Windows

- Windows® 2000 Professional with Service Pack 2-4
- Windows XP Professional with Service Pack 1

Solaris/Linux

- Sun Solaris® 8/9
- Red Hat® Enterprise Linux 3.0 (32-bit and 64-bit)

Software

- ISE™ 8.2i with applicable Service Pack

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.

Before you Begin

Before installing the core, you must have a Xilinx.com account and the ISE 8.2i software installed on your system. If you have already completed these steps, go to "Installing the Core."

1. Click Login at the top of the Xilinx home page; then follow the onscreen instructions to create a support account.
2. Install the ISE 8.2i software and the applicable Service Pack software.

Installing the Core

You can install the core in two ways—using the CORE Generator IP Software Update option to select from a list of updates, or by performing a manual installation after downloading the core from the web.

Using the CORE Generator Software Update Installer

Note: To use this installation method behind a firewall, you must know your proxy settings. Contact your administrator to determine the proxy host address and port number before you begin, if necessary.

1. Start the CORE Generator; then open an existing project or create a new one.
2. From the main CORE Generator window, choose Tools > Software Update. The WebUpdate screen appears.
3. If you are behind a firewall, click Set Proxy to either verify or set your proxy host and port settings.
4. Click Check for Updates. The Software Update installer appears.
5. Select the ISE 8.2IP Update 1 option; then click Install Selected. Informational messages may appear indicating that additional installations are required.
6. Click OK to accept any messages and continue. The User Login dialog box appears.
7. Enter your login name and password; then click OK. The selected update products are downloaded and installed.
8. To confirm the installation, check the following file: `C:\Xilinx\coregen\install\install_history`.

   Note that this step assumes your Xilinx software is installed in C:\Xilinx.

Manually

1. Close the CORE Generator if it is running.
2. Download the IP Update ZIP file from the following location and save it to a temporary directory: www.xilinx.com/support/download.htm.
3. Unpack the ZIP files using either WinZip (Windows) or Unzip (UNIX).
4. Extract the **ise_82i_ip_update1.zip** archive to the root directory of your Xilinx software installation. (Allow the extractor utility you use to overwrite all existing files and maintain the directory structure defined in the archive.)
5. If you do not have a zip utility, do one of the following:
   - **Windows**. From a command window, type the following:

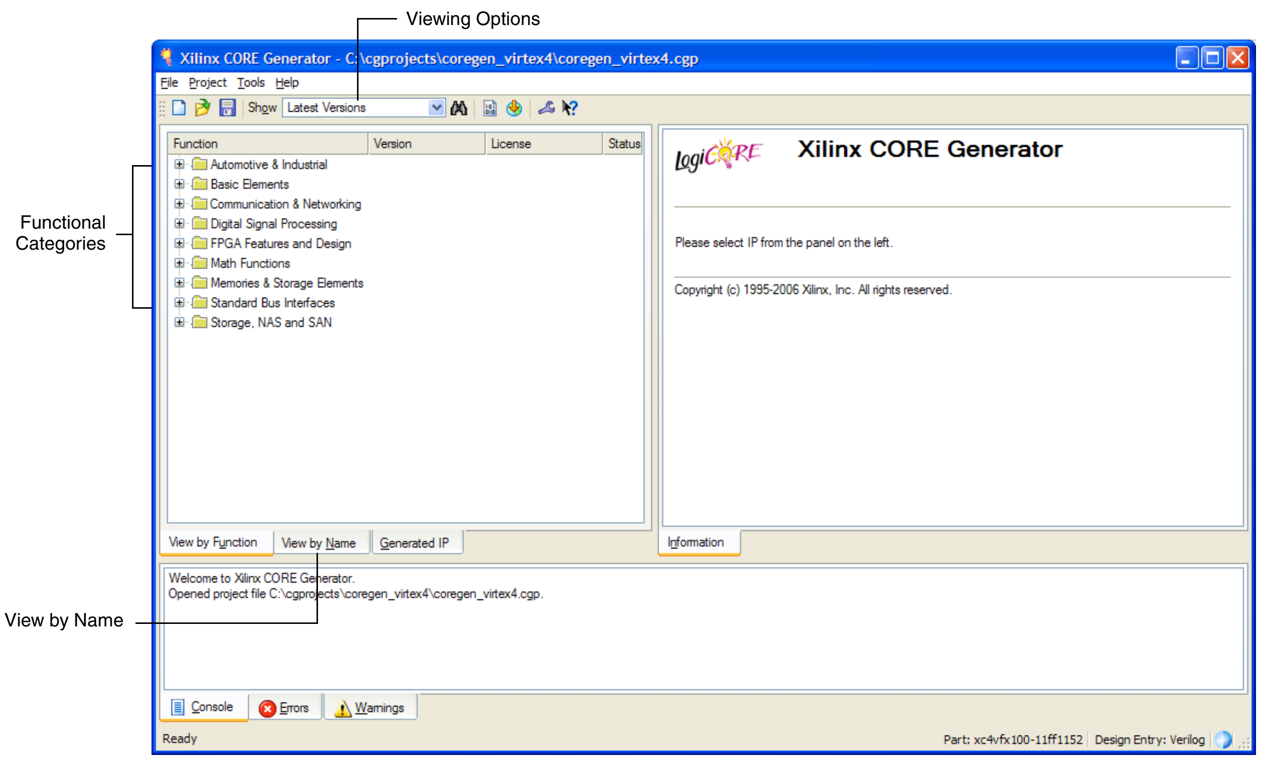
     ```
     %XILINX%/bin/nt/unzip -d %XILINX% ise_82i_ip_update1.zip
     ```
   - **Linux**. From a UNIX shell, type the following:

     ```
     $XILINX/bin/lin/unzip -d $XILINX ise_82i_ip_update1.zip
     ```
   - **Solaris**. From a UNIX shell, type the following:

     ```
     $XILINX/bin/sol/unzip -d $XILINX ise_82i_ip_update1.zip
     ```
6. To verify the root directory of your Xilinx installation, do one of the following:
   - **Windows**. Type `echo %XILINX%` from a DOS prompt.
   - **UNIX**. If you have already installed the Xilinx ISE software, the Xilinx variable defined by your set-up script identifies the location of the Xilinx installation directory. After sourcing the Xilinx set-up script, type `echo $XILINX` to determine the location of the Xilinx installation.

Verifying your Installation

1. Start the CORE Generator.
2. After creating a new project or opening an existing one, the IP core functional categories appear at the left side of the window.



Please select IP from the panel on the left.

Copyright (c) 1995-2006 Xilinx, Inc. All rights reserved.

```
Welcome to Xilinx CORE Generator.
Opened project file C:\cgprojects\coregen_virtex4\coregen_virtex4.cgp.
```

Figure 2-1: **CORE Generator Window**

3. Click to expand or collapse the view of individual functional categories, or click the View by Name tab at the bottom of the list to see an alphabetical list of all cores in all categories.
4. To view specific versions of the cores, choose an option from the Show drop-down list at the top of the window:
   - **Latest Versions**. Display the latest versions of all cores.
   - **All Versions**. Display all versions of cores, including new cores and new versions of cores.
   - **All Versions including Obsolete**. Display all cores, including those scheduled to become obsolete.
5. To determine that the installation is successful, be sure that the new core or cores appear in the CORE Generator GUI.

   For additional assistance installing the IP Update, contact www.xilinx.com/support.

License Options

The XAUI core provides two licensing options: a Simulation Only Evaluation license and a Full license. After installing the core, the Simulation Only Evaluation license is provided by default with the CORE Generator. To install the Full License, follow the directions below.

Simulation Only

The Simulation Only Evaluation license is provided with the XAUI core from the Xilinx CORE Generator. 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 dynamically generated gate-level netlist).

Full

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

- Gate-level 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

Simulation Only Evaluation License

No action is required because the Simulation Only Evaluation license is provided 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 XAUI 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 XAUI 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 e-mail containing the license and installation instructions will be sent immediately to the email address you specified.

Installing Your License File

After selecting a license option, an email will be sent to you that includes instructions for installing your license file. In addition, information about advanced licensing options and technical support is provided.

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 ModelSim or NC-Sim.

Introduction

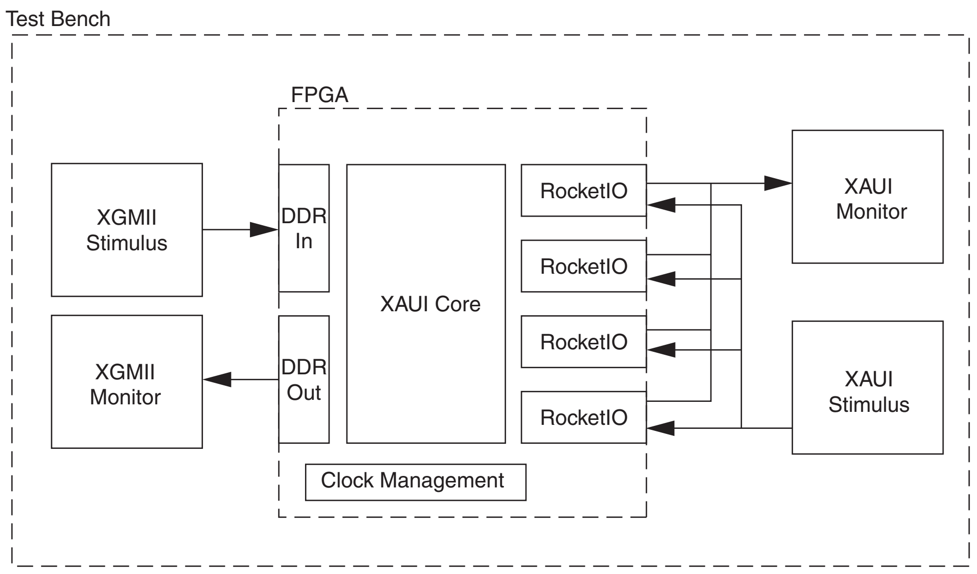


Figure 3-1: **The XAUI Example Design and Test Bench (Default Configuration)**

The XAUI example design consists of the following:

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

The XAUI Design Example has been tested with Xilinx ISE v8.2i and ModelSim SE 6.1e and IUS 5.5.

Generating the Core

To generate a XAUI core with default values using the CORE Generator, do the following:

1. Start the CORE Generator.

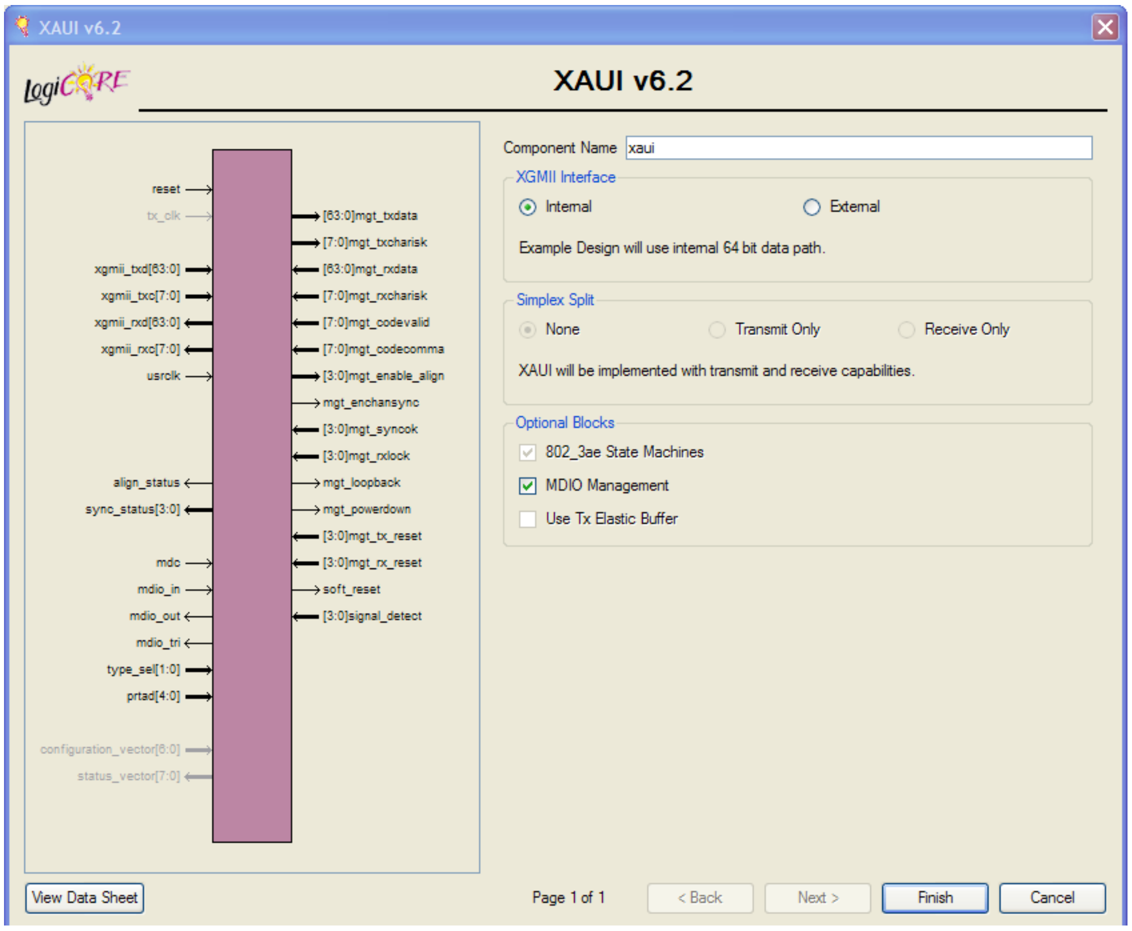
   For help starting and using CORE Generator, see the documentation supplied with ISE.

2. Choose File > New Project.
3. Type a directory name. In this example, the directory name *design* is used.
4. Do the following to set project options:
   - From the Part tab, select a silicon family, part, speed grade, and package that supports the XAUI core, for example, Virtex-II™ Pro or Virtex-4

   Note: If an unsupported silicon family is selected, the XAUI core will not appear in the taxonomy tree. For a list of supported architectures, see the *XAUI Data Sheet*.

   - From the Generation tab; for Design Entry 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 window. The XAUI 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 XAUI customization screen appears.
7. In the Component Name field, enter a name for the core instance. Accept all other options.
8. 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 Structure and File Descriptions" in Chapter 4.



XAUI v6.2

Component Name: xaui

XGMII Interface: [✓ Internal] External

Example Design will use internal 64 bit data path.

Simplex Split: [✓ None] Transmit Only Receive Only

XAUI will be implemented with transmit and receive capabilities.

Optional Blocks

- [✓] 802_3ae State Machines
- [✓] MDIO Management
- [ ] Use Tx Elastic Buffer

Page 1 of 1

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

Implementing the XAUI 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 XAUI 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:

For Unix

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

For 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 XAUI Example Design

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

Setting up for Simulation

The Xilinx UniSim and SimPrim libraries must be mapped into the simulator. If the UniSim and SimPrim libraries are not set up 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.

Pre-implementation Simulation

To run a functional 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/functional
```
2. Launch the simulation script:
```
modelSim: vsim -do simulate_mti.do
nc-sim: ./simulate_ncsim.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
```

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 XAUI core in your own design, see the *XAUI User Guide*.

Chapter 4

Detailed Example Design

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

Directory Structure and File Descriptions

VHDL Design Flow

Directory Structure

Figure 4-1 illustrates the directories and files created by the CORE Generator for a VHDL based project. <*project_dir*> is the CORE Generator project directory; <*component_name*> is the component name as entered in the XAUI core customization dialog box. The implement and timing simulation directories are only present when the core is generated with a full license.

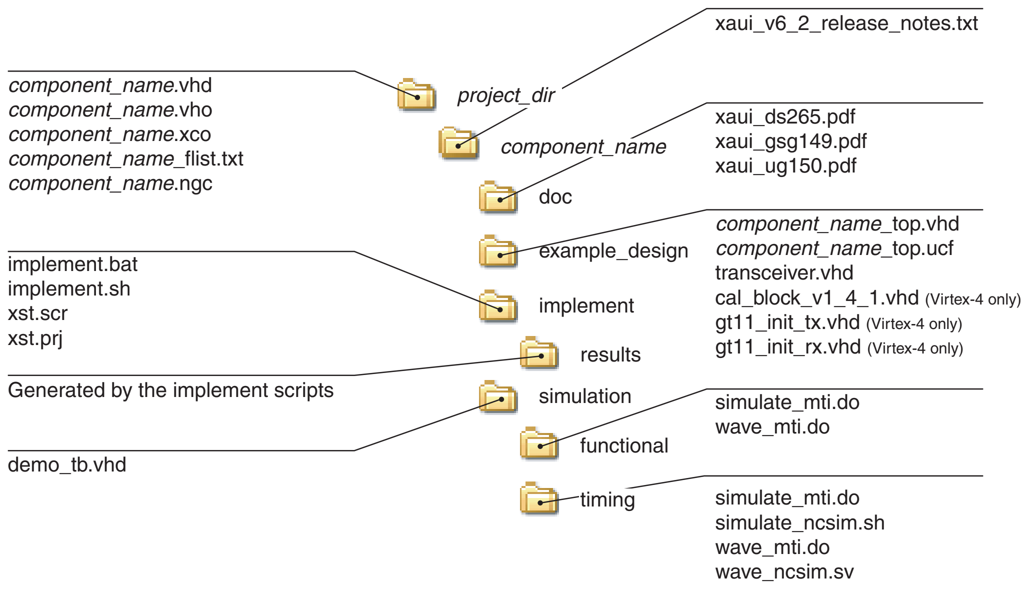


Figure 4-1: **XAUI Core Directories and Files: VHDL**

Project Directory (*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*.vhd**

VHDL structural simulation model. File used to support VHDL functional simulation of a core.

***component_name*.vho**

A VHDL template for the core. This can be copied into the user design.

***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 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**

A text file listing all of the output files produced when customized core was generated in the CORE Generator.

project_dir/component_name

xaui_v6_2_release_notes.txt

The XAUI release notes text file, which contains updates and information about the core.

project_dir/component_name/doc

xaui_ds265.pdf

The *XAUI Data Sheet*.

xaui_gsg149.pdf

The *XAUI Getting Started Guide.*

xaui_ug150.pdf

The *XAUI User Guide*.

project_dir/component_name/example_design

This directory contains the support files necessary for a VHDL implementation of the example design.

***component_name*_top.vhd**

The top-level entity for the example design.

transceiver.vhd (Virtex-II Pro only)

transceivers.vhd (Virtex-4 only)

Wrappers for the RocketIO™ transceivers.

cal_block_v1_4_1.vhd (Virtex-4 only)

Virtex-4 FX Calibration Block.

`gt11_init_tx.vhd (Virtex-4 only)`

`gt11_init_rx.vhd (Virtex-4 only)`

Virtex-4 transceiver reset circuitry.

`component_name` **`_top.ucf`**

User constraints file for the core and example design.

project_dir/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.

`implement.sh`

A UNIX shell script that processes the example design through the Xilinx tool flow.

`implement.bat`

A Windows batch file that process the example design through the Xilinx tool flow.

`xst.prj`

The XST project file for the example design.

`xst.scr`

The XST script file for the example design.

project_dir/component_name/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.

`routed.vhd`

The back-annotated SimPrim based VHDL design. Used for timing simulation.

`routed.sdf`

The timing information for simulation is contained in this file.

project_dir/component_name/simulation

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

`demo_tb.vhd`

The VHDL demonstration test bench for the XAUI core.

project_dir/component_name/simulation/functional

`simulate_mti.do`

A 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.

`wave_mti.do`

A 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.do`

A UNIX shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using NC-Sim.

`wave_ncsim.sv`

A NC-Sim macro file that opens a wave windows and adds interesting signals to it. It is called used by the simulate_ncsim.sh script.

project_dir/component_name/simulation/timing

`simulate_mti.do`

A ModelSim macro file that compiles the timing simulation model and the demonstration test bench then runs the timing simulation to completion.

`wave_mti.do`

A 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.do`

A UNIX shell script that compiles the test bench and the timing model then runs the timing simulation to completion using NC-Sim.

`wave_ncsim.sv`

A NC-Sim macro file that opens a wave windows and adds interesting signals to it. It is called used by the simulate_ncsim.sh script.

Verilog Design Flow

Directory Structure

Figure 4-2 illustrates the directories and files created by the CORE Generator for a VHDL based project. <*project_dir*> is the CORE Generator project directory; <*component_name*> is the component name as entered in the XAUI core customization dialog box. The implement and timing simulation directories are only present when the core is generated with a full or hardware evaluation license.

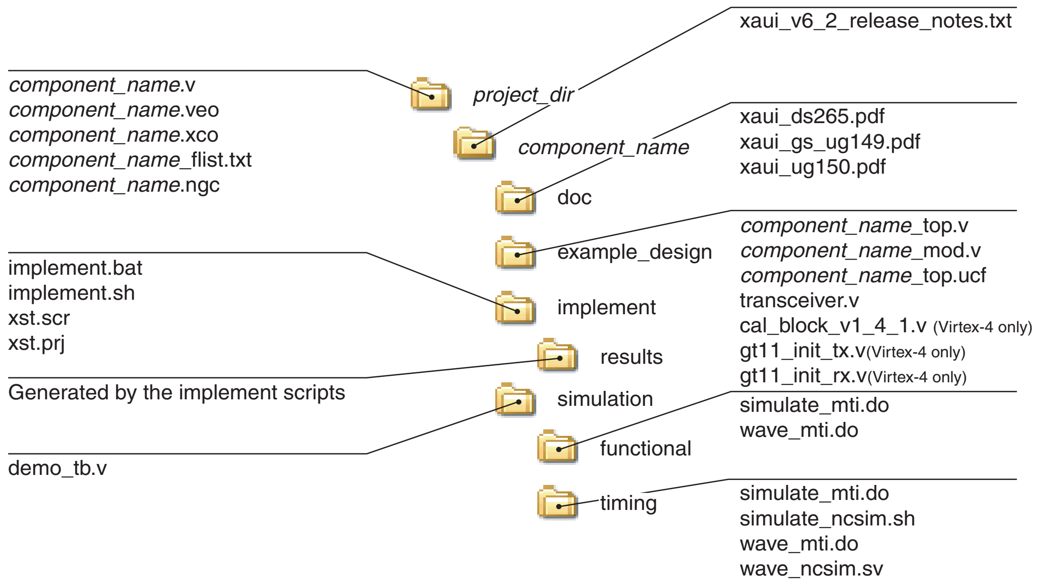


Figure 4-2: **XAUI Core Directories and Files: Verilog**

Project Directory (*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**

Verilog structural simulation model. File used to support Verilog functional simulation of a core.

***component_name*.veo**

A Verilog template for the core. This can be copied into the user design.

***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 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**

A text file listing all of the output files produced when customized core was generated in the CORE Generator.

project_dir/component_name

`xaui_v6_2_release_notes.txt`

The XAUI release notes text file, which contains updates and information about the core.

project_dir/component_name/doc

`xaui_ds265.pdf`

The *XAUI Data Sheet*.

`xaui_gsg149.pdf`

The *XAUI Getting Started Guide.*

`xaui_ug150.pdf`

The *XAUI User Guide*.

project_dir/component_name/example_design

This directory contains the support files necessary for a Verilog implementation of the example design.

`component_name` **`_top.v`**

The top level entity for the example design.

`transceiver.v (Virtex-II Pro only)`

`transceivers.v (Virtex-4 only)`

Wrappers for the RocketIO transceivers.

`cal_block_v1_4_1.v (Virtex-4 only)`

Virtex-4 FX Calibration Block.

`gt11_init_tx.v (Virtex-4 only)`

`gt11_init_rx.v (Virtex-4 only)`

Virtex-4 transceiver reset circuitry.

`component_name` **`_top.ucf`**

User constraints file for the core and example design.

project_dir/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.

`implement.sh`

A UNIX shell script that processes the example design through the Xilinx tool flow.

`implement.bat`

A Windows batch file that process the example design through the Xilinx tool flow.

`xst.prj`

The XST project file for the example design.

`xst.scr`

The XST script file for the example design.

project_dir/component_name/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.

`routed.v`

The back-annotated SimPrim based Verilog design. Used for timing simulation.

`routed.sdf`

The timing information for simulation is contained in this file.

project_dir/component_name/simulation

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

`demo_tb.v`

The Verilog demonstration test bench for the XAUI core.

project_dir/component_name/simulation/functional

`simulate_mti.do`

A 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.

`wave_mti.do`

A 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.do`

A UNIX shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using NC-Sim.

`wave_ncsim.sv`

A NC-Sim macro file that opens a wave windows and adds interesting signals to it. It is called used by the simulate_ncsim.sh script.

project_dir/component_name/simulation/timing

`simulate_mti.do`

A ModelSim macro file that compiles the timing simulation model and the demonstration test bench then runs the timing simulation to completion.

`wave_mti.do`

A 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.do`

A UNIX shell script that compiles the test bench and the timing model then runs the timing simulation to completion using NC-Sim.

`wave_ncsim.sv`

A NC-Sim macro file that opens a wave windows and adds interesting signals to it. It is called used by the simulate_ncsim.sh script.

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. It is located at:

For UNIX

`project_dir``/`*`component_name`*`/implement/implement.sh`

For 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
- These files are copied into the <*core_instance*>/test/vhdl and <*core_instance*>/test/verilog directories.

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

Simulation Scripts

Simulation macro files are provided for ModelSim and shell scripts are provided for NC-Sim. 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`

Timing

`project_dir``/`*`component_name`*`/simulation/timing/simulate_mti.do`

`project_dir``/`*`component_name`*`/simulation/functional/simulate_ncsim.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)
- Runs the simulation to completion

XAUI Core with External XGMII Client-side Interface

Example HDL Wrapper

Figure 4-3 illustrates an example HDL wrapper for XAUI with XGMII (Virtex-II Pro).

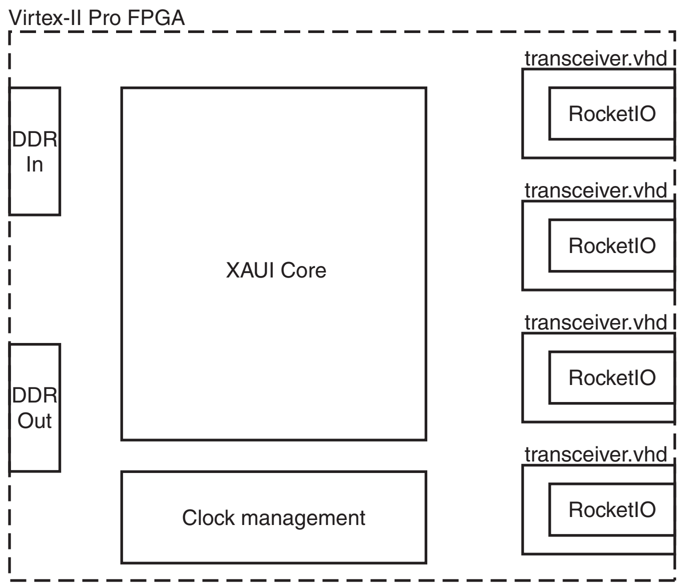


Figure 4-3: **Example HDL Wrapper for XAUI with XGMII (Virtex-II Pro)**

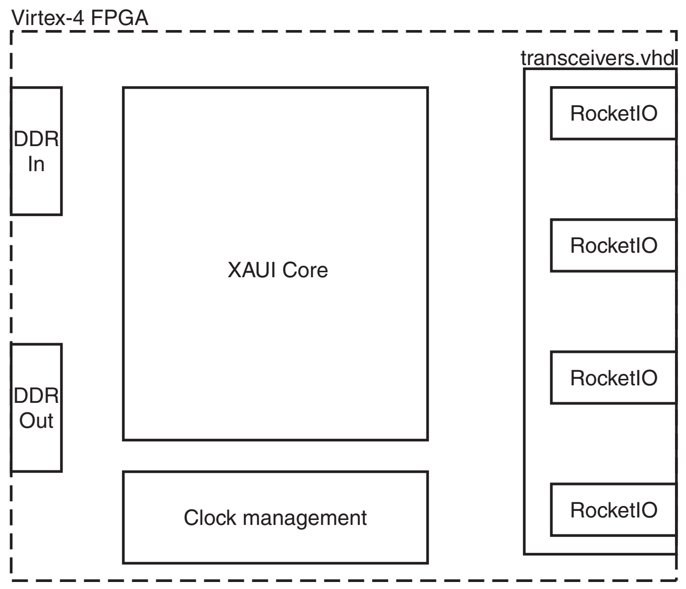


Figure 4-4: **Example HDL Wrapper for XAUI with XGMII (Virtex-4)**

In Figure 4-4, the example generated HDL wrapper contains the following:

- The RocketIO transceiver instances
- Clock management logic, including DCM and Global Clock Buffer instances
- DDR logic for the XGMII interface

Demonstration Test Bench

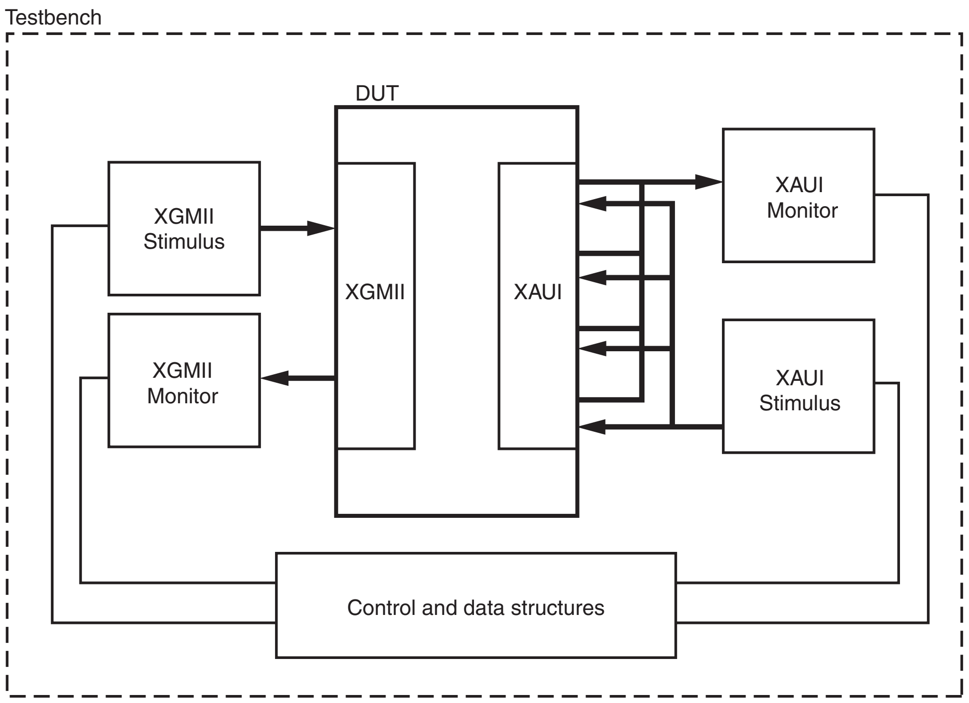


Figure 4-5: **Demonstration Test Bench for XAUI with XGMII Interface**

The demonstration test bench in Figure 4-5 is a simple VHDL or Verilog program to exercise the example design and the core itself. It consists of transactor procedures or tasks which 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.

XAUI Core with Internal Client-side Interface

Example HDL Wrapper

Figure 4-6 shows an example HDL wrapper for XAUI without XGMII (Virtex-II Pro).

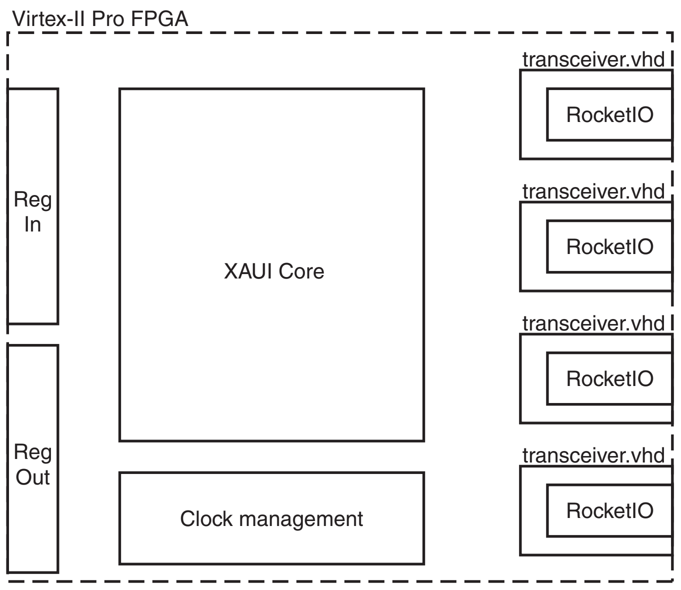


Figure 4-6: **Example HDL Wrapper for XAUI without XGMII (Virtex-II Pro)**

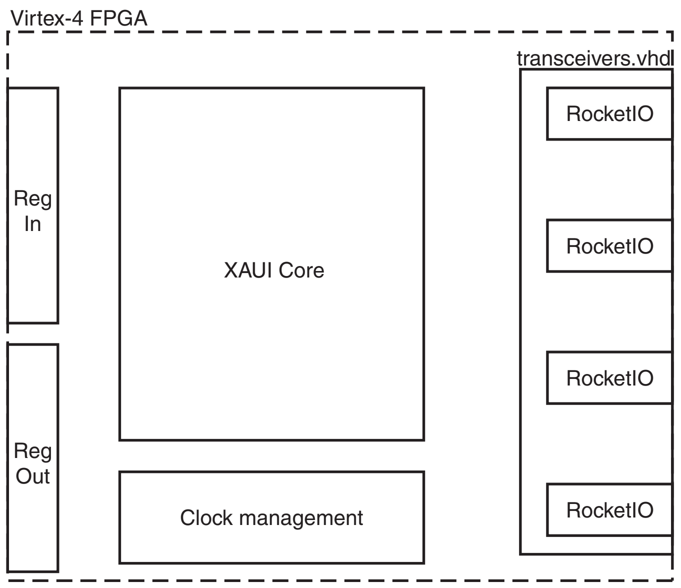


Figure 4-7: **Example HDL Wrapper for XAUI without XGMII (Virtex-4)**

In Figure 4-7, the example HDL wrapper generated when the internal client-side interface is selected contains the following:

- The RocketIO transceiver instances
- Virtex-4 RocketIO Calibration Blocks (see Answer Record 22477 for information about the Calibration Block User Guide)
- Clock management logic, including DCM and Global Clock Buffer instances
- Re-timing registers on the parallel data interface, both on input and output

Demonstration Test Bench

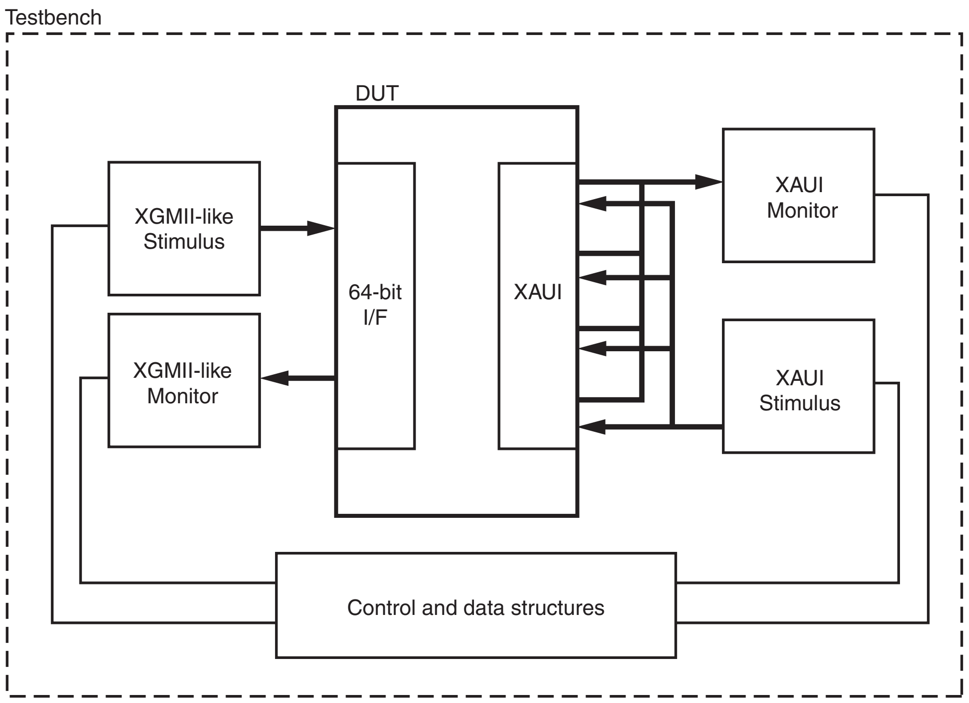


Figure 4-8: **Demonstration Test Bench for XAUI without XGMII Interface**

In Figure 4-8, the demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself. It 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.