LogiCORE™ IP Fibre Channel v3.5

Getting Started Guide

UG135 April 19, 2010

XILINX®

Revision History

Release Date	Version	Description
6/29/04	1.0	Initial Xilinx® release.
4/28/05	2.0	Updated to v2.0 of the core; Xilinx tools v7.1i SP2.
1/18/06	2.1	Updated core to v2.1; Xilinx tools v8.2i.
2/15/07	3.1	Updated core to v3.1; Xilinx tools v9.1i.
8/8/07	3.2	Updated core to v3.2; Xilinx tools 9.2i.
3/24/08	3.5	Updated core to v3.3; Xilinx tools 10.1.
4/24/09	3.6	Updated core to v3.4; Xilinx tools 11.1.
4/19/10	3.7	Updated core to v3.5; Xilinx tools 12.1.

Table of Contents

Schedule of Figures

Chapter 1: Introduction

Chapter 2: Licensing the Core

Chapter 3: Fibre Channel Example Design

About This Guide

Guide Contents

The *Fibre Channel v3.5 Getting Started Guide* provides initial information about using the LogiCORE™ IP Fibre Channel (FC) core and provides instructions for using the example design provided with the core. The following chapters are included:

- Preface, "About this Guide" introduces the organization and purpose of the *Getting Started Guide* and describes the conventions used in this document.
- Chapter 1, "Introduction" describes where to obtain the core, lists references to related material, provides contact information about technical support, and explains how to send feedback to Xilinx.
- Chapter 2, "Licensing the Core" describes licensing options and provides instructions for obtaining and installing a license key.
- Chapter 3, "Fibre Channel Example Design" provides instructions to quickly generate the core and run the example design through implementation and simulation.

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. Signal names also.	`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**

Convention	Meaning or Use	Example
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 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;*
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 Documents

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, "Conventions" for details. See "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com.

XILINX®

Chapter 1

Introduction

The Fibre Channel (FC) core is a fully verified, pre-implemented interface that can be used to provide connectivity in storage networking and other data transfer applications. The core supports both Verilog-HDL and VHDL design flows.

This guide provides information about what you need to get started using the FC core and is also the reference document for the example design included with the core. This chapter describes how to obtain the core, provides references to related material, and tells you how to obtain technical support and provide feedback about the documentation and the core. For system requirements and installation instructions, see the *Fibre Channel Release Notes (readme.txt)* file accompanying the core documentation.

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 FC core is a Xilinx® CORE Generator™ software IP core. The core is included in the latest IP Update on the Xilinx IP Center. Details about the IP Update, including information you need to access the core, are located on the Fibre Channel product page.

With the default evaluation license available through the CORE Generator software, you can generate a functional simulation model. To access additional capabilities, you need to request either a Full System Hardware Evaluation or Full license. See Chapter 2, "Licensing the Core" for more information.

Recommended Experience

The FC core is delivered as a fully verified, pre-implemented netlist. The pre-built, already verified nature of the core frees the designer to focus on the details of his or her design. The challenge associated with implementing a complete FC design, including custom user application functions, varies depending on the configuration and functionality of your application.

In general, previous experience building high performance, pipelined FPGA designs using Xilinx implementation software and user constraint files (UCF) is recommended.

For an in-depth review of your specific requirements, contact your local Xilinx representative. More extensive design assistance is also available through Xilinx Design Services (www.xilinx.com/xds/index.htm).

Additional Core Resources

For detailed information about the FC core including the latest IP documents and known issue updates, see the following documents, located on the Fibre Channel product page.

- Fibre Channel Data Sheet
- Fibre Channel User Guide
- Fibre Channel Release Notes

Updates to this document may also be available periodically in the Xilinx Fibre Channel Product Lounge.

Technical Support

For technical support, go to www.xilinx.com/support/mysupport.htm. From this page, you can create a WebCase to route your inquiry to a support engineer with specific expertise using the Fibre Channel core.

Xilinx provides technical support for this product when used according to guidelines described in the *Fibre Channel User Guide* and the *Fibre Channel Getting Started Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Providing Feedback

Xilinx welcomes feedback about the FC core and the documentation provided with the core.

Document Feedback

If you have any comments about this document, please submit a WebCase from the Xilinx support website www.xilinx.com/support/mysupport.htm. Please provide the following information when submitting a WebCase:

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

General suggestions for additions and improvements are also welcome.

Core Feedback

If you have any comments or suggestions about this product, please submit a WebCase from the www.xilinx.com/support/mysupport.htm website. Please provide the following information when submitting a WebCase:

- Product name and version
- Options selected when generating the core
- Explanation of your comments

Licensing the Core

Please follow the instructions below for obtaining a license key before using the core in your design. This core is provided under the terms of the Xilinx LogiCORE™ Site License Agreement.

Before you Begin

This chapter assumes that you have installed the required ISE Updates specified by the data sheet for this core, following the instructions provided by the Xilinx ISE Installation, Licensing and Release Notes Guide.

License Options

This LogiCORE IP module offers three licensing options. After installing the required Xilinx ISE and EDK Updates, choose a license option.

Simulation Only

The Simulation Only Evaluation license key for this core is included with this core, shipped with the Xilinx ISE software. This key lets you assess core functionality with your own design. (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 functional simulation of this core using the example design and 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 will continue to function 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

This core requires installation of a Full License key and the relevant ISE Update.

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 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 ISE software.

Full System Hardware Evaluation License

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

1. Navigate to the product page for this core: www.xilinx.com/products/ipcenter/FIBRE_CH.htm
2. Click **Evaluate**.
3. Follow the instructions to install the required Xilinx ISE Update.

Obtaining Your Full License Key

To obtain a Full license key, please follow these instructions.

1. Navigate to the product page for this core: www.xilinx.com/products/ipcenter/FIBRE_CH.htm
2. Click **Order**.
3. Follow the instructions to install the required Xilinx ISE Update and generate the required license key on the Xilinx Product Download and Licensing Site, www.xilinx.com/getproduct.

Installing Your License File

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.

XILINX®

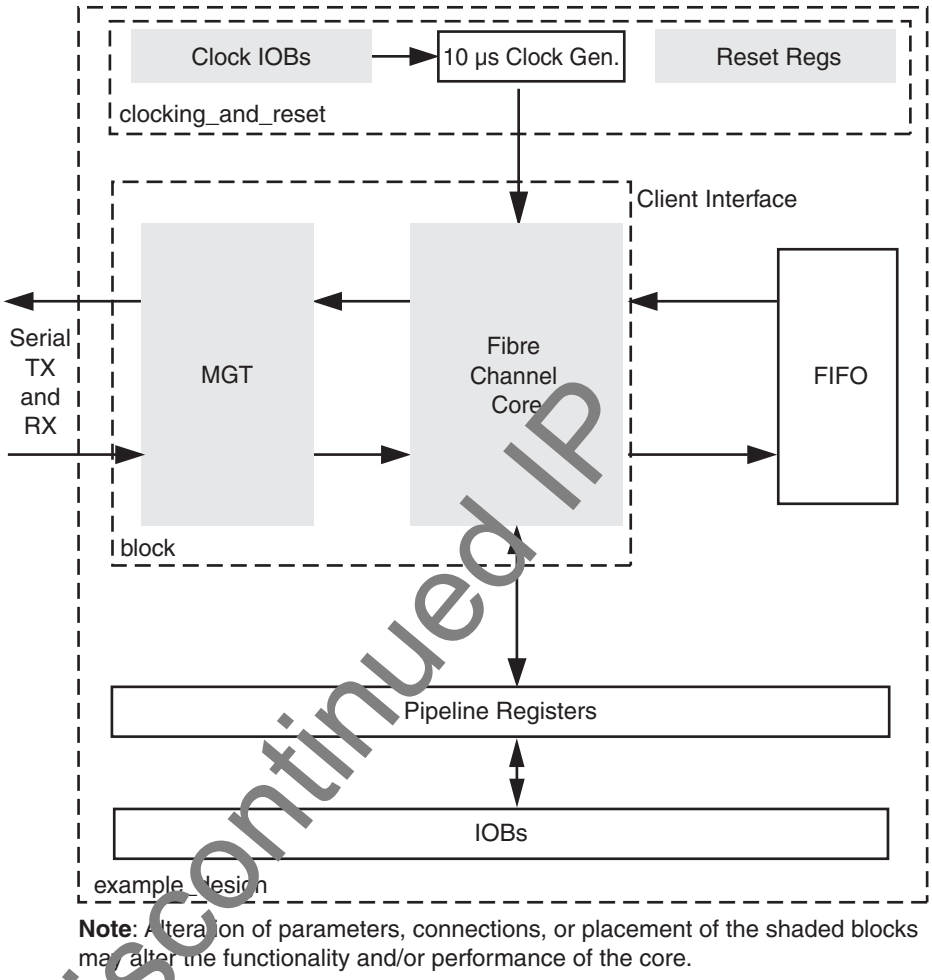
Chapter 3

Fibre Channel Example Design

This chapter demonstrates how to generate and use the FC example design with the default parameters. Detailed information about the files and directory structure generated by the CORE Generator™ software system is provided. In addition, information about the purpose and content of the implementation scripts, the content of the example HDL wrappers, and the operation of the demonstration test bench is provided.

Introduction

Figure 3-1 illustrates the example design default configuration.



Note: Alteration of parameters, connections, or placement of the shaded blocks may alter the functionality and/or performance of the core.

Figure 3-1: **Fibre Channel Example Design: Default Configuration**

The FC example design includes the following:

- FC core netlist/UniSim model
- Example HDL wrapper
- Demonstration test bench to exercise the example design

The FC example design is tested with Xilinx® ISE® software v12.1, Cadence Incisive Enterprise Simulator (IES) v9.2 and Mentor Graphics ModelSim v6.5c. In this design, the Client Rx port is connected to the Client Tx port through the FIFO. With this arrangement, any error-free frames received by the core will be retransmitted without changes.

In the example design, the FC serial interface is brought out to the RocketIO™ transceivers in the target FPGA device. Clocking and support circuitry (for example, logic to generate the 10 µs clock required by the core) is instantiated in the HDL wrapper.

The other signals going to and coming from the core (for example, the Management Interface signals) are pipelined using two registers. One of these registers should automatically be placed into an IOB by the ISE tools during the implementation process.

Configuring the example design I/Os in this way prevents timing problems that would otherwise be encountered when trying to drive the interface signals on the core directly from IOBs. In an actual user design, these interface signals would normally be driven by user logic located within the FPGA fabric and would not require pipelining.

The FIFO used in the wrapper can be replaced with your own logic. Another option is to stimulate the serial RX interface and monitor the serial TX interface using either the demonstration test bench or a third party FC Port or IP core.

Generating the Core

The FC core is generated using the CORE Generator software. For information about starting and using the CORE Generator software, see the *CORE Generator Help* available within the ISE software 12.1 Software Manuals on www.xilinx.com/support/software_manuals.htm.

To use the FC example design, do the following:

1. Start the CORE Generator software in one of the following ways:
   - Linux

     Type `coregen` at a shell prompt.
   - Windows

     Choose Programs > Xilinx ISE 12 > Accessories > CORE Generator.
2. Create a new project.
3. From the Project options, select a silicon family that supports the FC core. At this time, Virtex®-4 and Virtex-5 devices are supported.

   Note: If an unsupported silicon family or part is selected, the core will not be available for customization. The Virtex-4 FX device solutions require the latest silicon stepping and are pending hardware validation. For more information about supported devices, please see the *Fibre Channel Data Sheet*.
4. In the Design Entry section of the Project options, select either VHDL or Verilog.
5. Select Other for the Vendor.
6. Locate the FC core in the taxonomy tree under one of the following:
   - Communications & Networking/Networking
   - Communications & Networking/Telecommunications
   - Storage, NAS & SAN/Storage Area Networking

7. Double-click the core. A dialog box may appear to alert you to license limitations. Click OK to exit the dialog box. The FC screen appears.

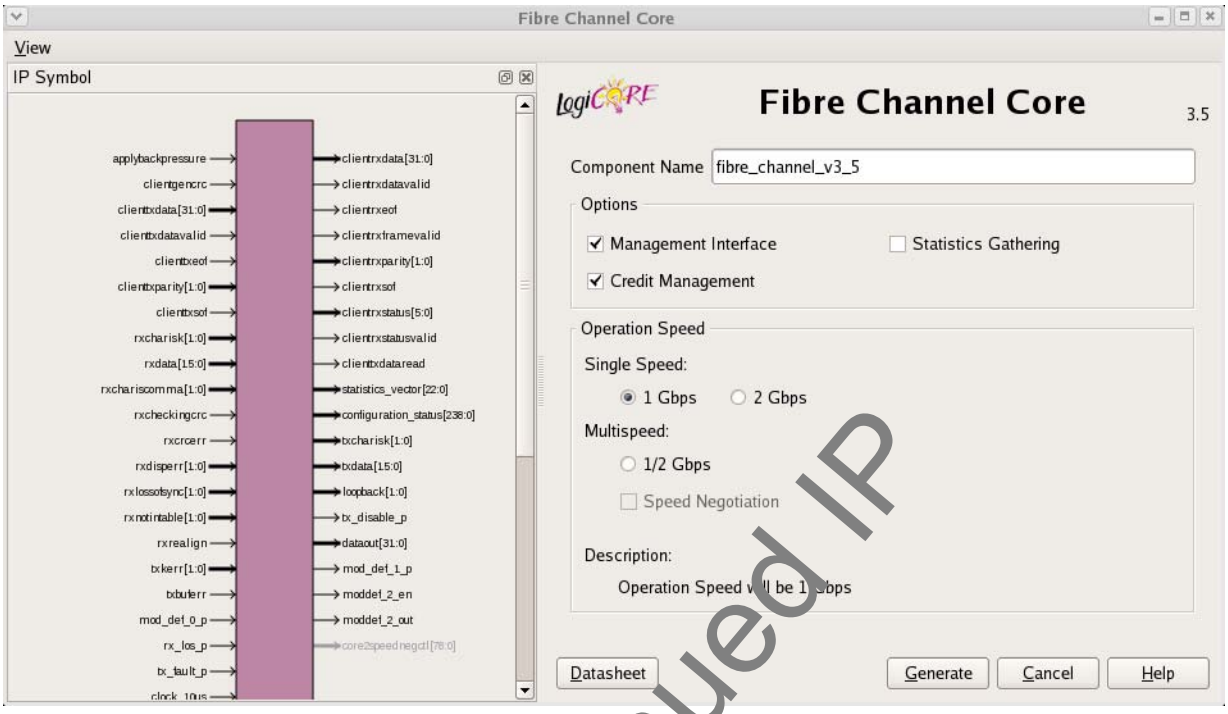


Figure 3-2: **Fibre Channel Screen**

8. For Component Name, enter a name for the core.
9. Select the desired options and click Generate.

The core and its supporting files, including the example design, are generated in your project directory. Detailed descriptions of the example design files and directories are provided in the following section.

Directory and File Contents

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

- **<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

<project directory>

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

Table 3-1: **Project Directory**

Name	Description
<project_dir>	
<component_name>_core.ngc	NGC Implementation netlist for the core. Defines how the core is implemented. Used as input to the Xilinx Implementation Tools.
<component_name>_core.v[hd]	Verilog or VHDL structural simulation model. Used to support functional simulation of a core. The Verilog or VHDL wrapper passes customized parameters to the generic core simulation model.
<component_name>.{veo|vho}	Verilog or VHDL template file. The components in this file can be used to instantiate a core.
<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 software system for each core it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator software.
<component_name>_speed_neg.ngc	NGC Implementation Netlist for the speed negotiation core. Defines how the core is implemented. Used as input to the Xilinx Implementation Tools. **Note**: Speed negotiation files are generated only when speed negotiation is requested in the GUI.
<component_name>_speed_neg.v[hd]	Verilog or VHDL structural simulation model for the speed negotiation core
<component_name>_speed_neg.{veo|vho}	Verilog or VHDL template file. The components in this file can be used to instantiate a core.
<component_name>_flist.txt	Text file listing all of the output files produced when customized core is generated in the CORE Generator software

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 3-2: **Component Name Directory**

Name	Description
<project_dir>/<component_name>	
fibre_channel_readme.txt	Fibre Channel release notes file

Back to Top

<component name>/example_design

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

Table 3-3: **Example Design Directory**

Name	Description
<project_dir>/<component_name>/example_design	
*<component_name>*_example_design.v[hd]	Top-level Verilog or VHDL file for the example design.
*<component_name>*_block.v[hd]	Instances the core and the RocketIO MGT transceiver, as shown in Figure 3-1, page 16.
*<component_name>*_clocking_and_reset.v[hd]	Instances the DCM(s) and reset logic for the example design, as shown in Figure 3-1, page 16.
<component_name>_mod.v	Verilog module declaration for the core instance in the example design.
FCMGT.v[hd]	Wrapper file for the FC device-specific RocketIO MGT transceivers, which is instanced in the top-level Verilog or VHDL file.
cal_block_v1_4_1.v[hd]	The device-specific RocketIO MGT transceiver Calibration Block, instanced in the FCMGT file (Virtex-4 FX devices only).
gt11_speed_controller.v[hd]	Control logic for speed control (multi-speed cores only) which is instanced in the top level (Virtex-4 FX devices only).
gtp_speed_controller.v[hd]	Control logic for speed control (multi-speed cores only) which is instanced in the top level (Virtex-5 devices only).
fifo.v[hd]	FIFO design instanced in the top level.
<component_name>_top.ucf	UCF for the core and the example design.

Back to Top

<component name>/doc

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

Table 3-4: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
`fibre_channel_ds270.pdf`	Fibre Channel Data Sheet
`fibre_channel_ug136.pdf`	Fibre Channel User Guide
`fibre_channel_gsg135.pdf`	Fibre Channel Getting Started Guide

Back to Top

<component name>/implement

This directory contains the scripts and project files to run the core through the Xilinx tool flow.

Table 3-5: **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 synthesized.
`xst.scr`	XST script file for the example design.

Back to Top

implement/results

The results directory is produced by the implement scripts and is used to run the example design files and `<component_name>.ngc` file through the Xilinx Implementation tools.

Table 3-6: **Results Directory**

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

Back to Top

<component name>/simulation

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

Table 3-7: **Simulation Directory**

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

Back to Top

simulation/functional

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

Table 3-8: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/functional	
`simulate_mti.do`	ModelSim macro file that compiles the Verilog or VHDL sources then runs the 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 IES.
`wave_ncsim.sv`	Cadence IES macro file that opens a wave windows and adds interesting signals to it. This macro is called used by the simulate_ncsim.sh script.

Back to Top

simulation/timing

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

Table 3-9: **Timing Directory**

Name	Description
<project_dir>/<component_name>/simulation/timing	
`simulate_mti.do`	ModelSim macro file that compiles the Verilog or VHDL sources then runs the 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 IES.
`wave_ncsim.sv`	Cadence IES macro file that opens a wave windows and adds interesting signals to it. This macro is called used by the simulate_ncsim.sh script.

Back to Top

Implementing the Example Design

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

Open a command prompt or shell in your project directory, then do one of the following:

- **Linux**

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

- **Windows**

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

These steps start a script to synthesize the example design HDL wrapper and build the design. The script then creates gate-level netlist HDL files in VHDL or Verilog, along with associated timing information (SDF) files.

Simulating the Example Design

Setting up for Simulation

Functional simulation is supported through the dynamic generation of a UniSim simulation model in the CORE Generator software. 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.

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

Virtex-5 Devices

Virtex-5 device designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator.

For a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator, ModelSim v6.5c is currently supported.

Virtex-4 Devices

Virtex-4 device designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator.

Verilog Simulation

To run a Verilog functional simulation of the example design with ModelSim:

1. Launch ModelSim and set the current directory to

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

2. Map the UniSim library:

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

3. Map the SecureIP library:

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

4. Launch the simulation script:

   `ModelSim> do simulate_mti.do`

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

VHDL Simulation

To run a VHDL functional simulation of the example design:

Launch ModelSim and set the current directory to

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

1. Map the UniSim library:

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

2. Map the SecureIP library:

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

3. Launch the simulation script:

   `ModelSim> do simulate_mti.do`

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

Timing Simulation

Gate-level simulation is supported through the dynamic generation of a gate-level simulation model in the ISE software. To run a simulation, you must have the Xilinx Simulation Libraries compiled for your system. See *Compiling Xilinx Simulation Libraries (COMPXLIB)* in the Xilinx ISE *Synthesis and Simulation Design Guide*.

Note: During the simulation of Multispeed configurations, there may be timing errors generated as the core changes speed. These may be safely ignored.

Verilog Simulation

To run a Verilog gate-level simulation of the example design with ModelSim:

1. Launch ModelSim and set the current directory to:

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

2. Map the SimPrim library:

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

3. Map the SecureIP library:

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

4. Launch the simulation script:

   `ModelSim> do simulate_mti.do`

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

VHDL Simulation

To run a VHDL gate-level simulation of the example design:

1. Launch ModelSim and set the current directory to
   `<project_dir>/<component_name>/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`

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

Implementation and Test Scripts

Implementation Script

The implement directory is only produced when CORE Generator software has been run with a Full System Hardware Evaluation license or a Full license. 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`

A directory `/results` is created in the `/implement` directory, in which all implementation files and tool reports are included.

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 an NGD file containing the entire design
- The design is mapped to the target technology
- The design is placed-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 or Verilog netlists and timing information in the form of SDF files

Test Scripts for Timing Simulation

The test script is a ModelSim macro that automates the timing-based simulation of the test bench.

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

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

The test script performs the following tasks:

- Compiles the gate level netlist
- Compiles the example design files
- Compiles the demonstration test bench
- Starts a simulation of the test bench including timing information
- Opens a Wave window and adds some signals of interest to it (`wave_mti.do/wave_ncsim.sv`)
- Runs the simulation to completion

Test Scripts for Functional Simulation

The test script is a ModelSim macro that automates the functional simulation of the test bench.

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

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

The test script performs the following tasks:

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

Demonstration Test Bench

Figure 3-4 shows the architecture of the demonstration test bench. The demonstration test bench performs the following steps:

- Resets the core (only on first pass through tests).
- Sets `ols_tov` to 10 µs.

  This is useful only for simulation purposes to reduce the time-to-wait before the Port State Machine (PSM) can leave the OL1 state.
- Enables the PHY device (for completeness only as there is no actual PHY device).
- Enables the device specific RocketIO transceiver at the correct speed and sets Parity Disable (disables Parity checking on Transmit path).
- Waits until the core has established Word Synchronization.
- Enables the Port State Machine (PSM) and enable Statistics Gathering, if required.
- Waits for the PSM to get to ACTIVE state. The test bench will stimulate the core with OLS for 10 µs or so, then LRR primitives for 3 µs and then IDLE primitives for another 3 µs.
- Performs one of the following:
  - Stimulates the Receive path of the core with 4 FC-FS frames.
  - Stimulates the core with a specific error-type and then terminate simulation.
- In the absence of any errors, the test bench waits for all frames to be received by the core and retransmitted before continuing.
- If the core is a multi-speed configuration, the operation speed is changed to the lower of the supported speeds. Once the core is in active state, the Receive path is stimulated with the same 4 FC-FS frames as before.
- Programs a BBCredit value of 16 into the core.
- Stimulates the core with LR primitives (Send Link Reset) until LRR primitives are detected on the Transmit path.
- Stimulates the core with IDLE primitives until the PSM transitions to the Active state once more.

- Checks that IDLEs are being transmitted.
- Checks that the BBCredit value has been written into core correctly.
- Checks any relevant statistics, either in the Statistics Gathering block if that is included in the core, or by using the Statistics Vector with the counters in the test bench.
- If required, start the process again. There is no reset for subsequent runs.

Checking Frame Data

The test bench checks that each word of every frame received matches the appropriate word in the frames that are retransmitted via the FIFO in the example design. Any errors are reported during the simulation. Frames are compared at the serial side of the core.

Using the Test Bench

Although not recommended, the demonstration test bench can be edited. As indicated in the LogiCORE™ IP License agreement, alterations are not supported by Xilinx.

To demonstrate a special feature of the core, you can make changes to the top-level parameters in the test bench as defined in the following sections.

SINGLE_RUN Parameter

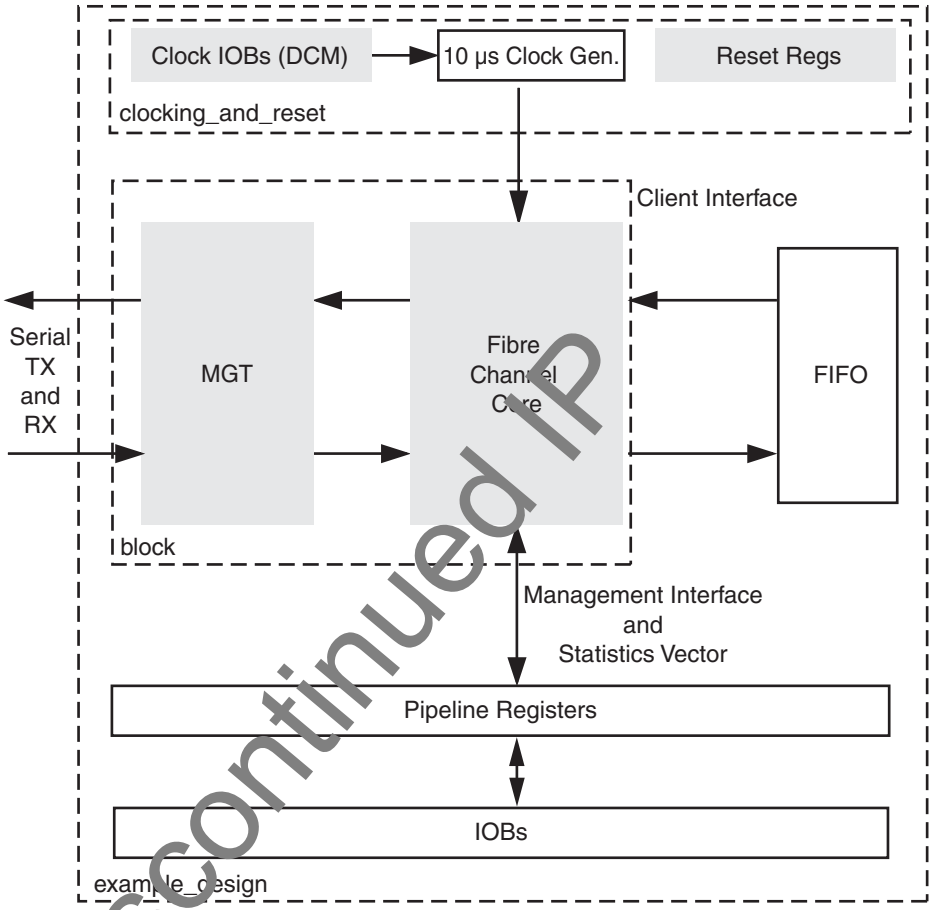
When the ERROR_TYPE parameter is set to 0, you can change the SINGLE_RUN parameter to TRUE or FALSE for VHDL; 1 or 0 for Verilog-HDL. A value of TRUE or 1 results in a single run of four frames through the core. A value of FALSE or 0 results in back-to-back runs of four frames each, continuing until the simulation is manually stopped.

ERROR_TYPE Parameter

When set to a value of 0, no specific errors are introduced during the simulation. When set to a value 1..9, specific errors are introduced during the simulation run. For these cases, the SINGLE _RUN parameter is ignored and assumed to be TRUE.

Single-speed Core with Management Interface and No Statistics

Example HDL Wrapper



Note: Alteration of parameters, connections, or placement of the shaded blocks may alter the functionality and/or performance of the core.

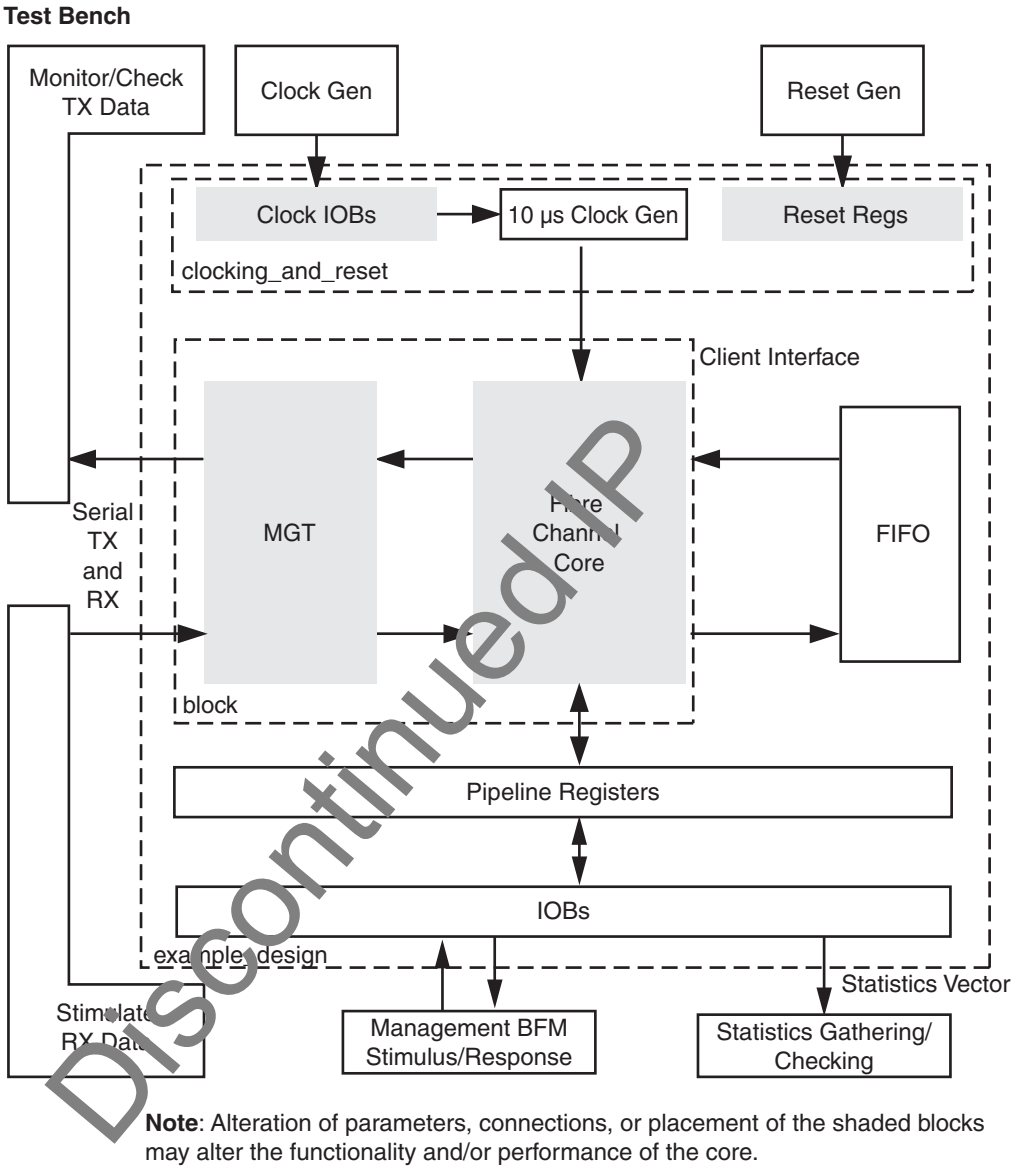
Figure 3-3: **Example HDL Wrapper for Single-speed Fibre Channel with Management Interface and No Statistics**

The example HDL wrapper contains the following:

- The device-specific RocketIO transceiver instance
- Clock management logic, including DCM and Global Clock Buffer instances
- Registers to synchronize Reset signals
- 10 µs clock generation logic
- Client Loopback FIFO
- Pipeline registers and IOBs on Management Interface signals and Statistics Vector signals

Sections labeled DO NOT MODIFY in the wrapper appear where modifications could break the example design.

Demonstration Test Bench



Note: Alteration of parameters, connections, or placement of the shaded blocks may alter the functionality and/or performance of the core.

Figure 3-4: **Demonstration Test Bench for Single-speed Fibre Channel with Management Interface**

The demonstration test bench is a simple VHDL or Verilog verification tool that exercises 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.

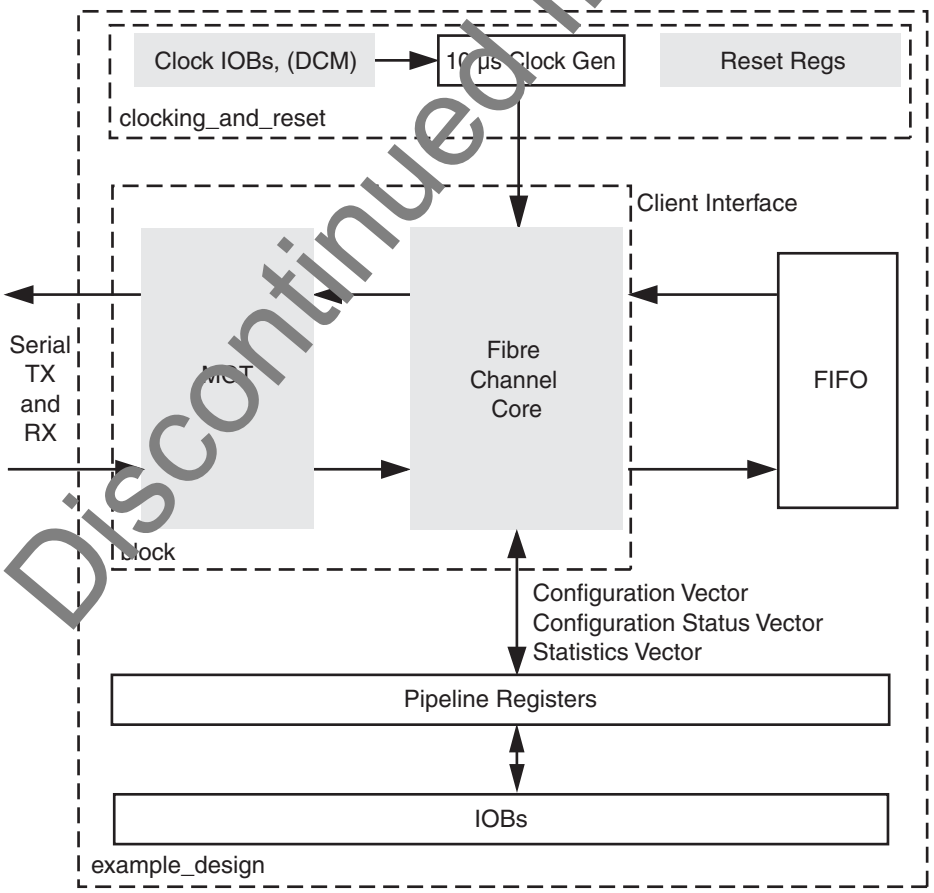
When the core is configured without a Statistics Gathering block, the test bench maintains local counters for statistics and increments them when the appropriate bit(s) of the Statistics Vector are asserted for one clock cycle. The final values of these counters are checked for consistency at the end of the simulation. If the Statistics Gathering block was available, all checks are made on that instead.

With the Management Interface, all accesses to the Configuration Registers are performed through that interface, as driven by the Management Interface Bus Functional Model (BFM) in the test bench. The Configuration Status Vector is never used.

To insert both word and bit-errors in the Stimulate Rx Data BFM, use the ERROR_TYPE parameter in the test bench. It is also possible to trigger some of the top-level PHY signals with this feature.

Single-speed Core with No Management Interface

Example HDL Wrapper



Note: Alteration of parameters, connections, or placement of the shaded blocks may alter the functionality and/or performance of the core.

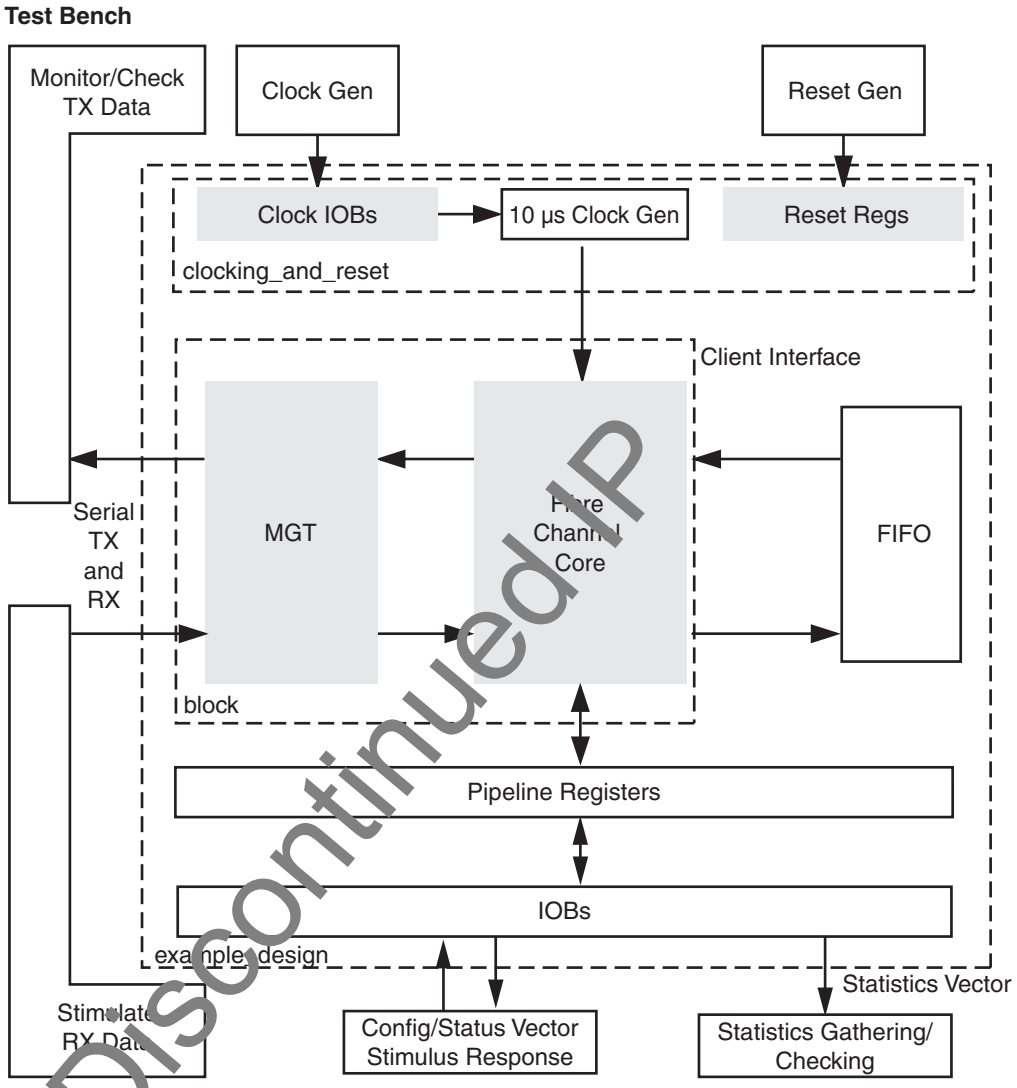
Figure 3-5: **Example HDL Wrapper for Single-Speed Fibre Channel with No Management Interface**

The example HDL wrapper generated for a single-speed FC core when no Management Interface is selected contains the following:

- The device-specific RocketIO transceiver instance
- Clock management logic, including DCM and Global Clock Buffer instances
- Registers to synchronize Reset signals
- 10 µs clock generation logic
- Client Loopback FIFO
- Pipeline registers and IOBs on Configuration Vector, Configuration Status Vector and Statistics Vector signals

Sections labeled DO NOT MODIFY in the wrapper appear where modifications could break the example design.

Demonstration Test Bench



Note: Alteration of parameters, connections, or placement of the shaded blocks may alter the functionality and/or performance of the core.

Figure 3-6: **Demonstration Test Bench for Single-speed Fibre Channel with No Management Interface**

The demonstration test bench is a simple VHDL or Verilog verification tool that exercises 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 values as they exit the core.

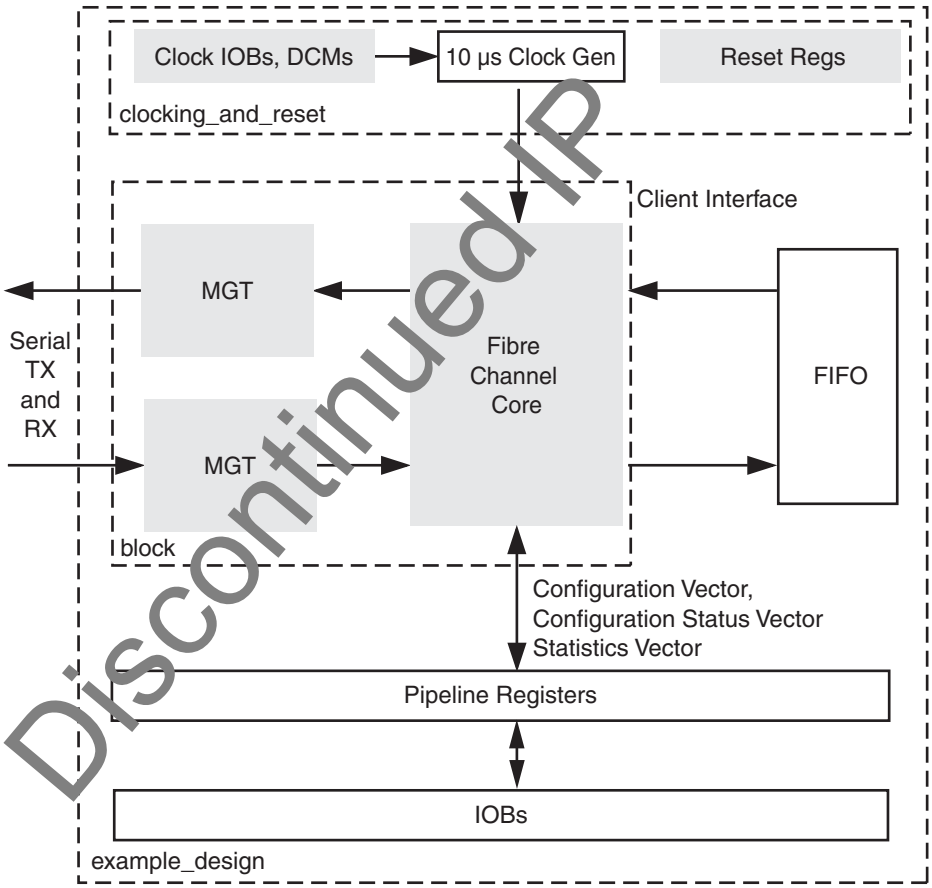
Without a Statistics Gathering block, it is up to the test bench to keep local statistics counters, which are incremented when the appropriate bit(s) of the Statistics Vector are asserted for a clock cycle. The final values of these counters are checked for consistency at the end of the simulation.

Without a Management Interface, all accesses to the Configuration Registers are made directly through the Configuration Vector (in) and Configuration Status Vector (out) in the test bench.

It is possible to insert both word and bit-errors in the Stimulate Rx Data Bus Functional Model (BFM) using the ERROR_TYPE parameter in the test bench and to trigger some of the top-level PHY signals with this feature.

Dual-speed Core with No Management Interface

Example HDL Wrapper



Note: Alteration of parameters, connections, or placement of the shaded blocks may alter the functionality and/or performance of the core.

Figure 3-7: **Example HDL Wrapper for Dual-speed Fibre Channel with No Management Interface**

The example HDL wrapper generated when no Management Interface is selected contains the following:

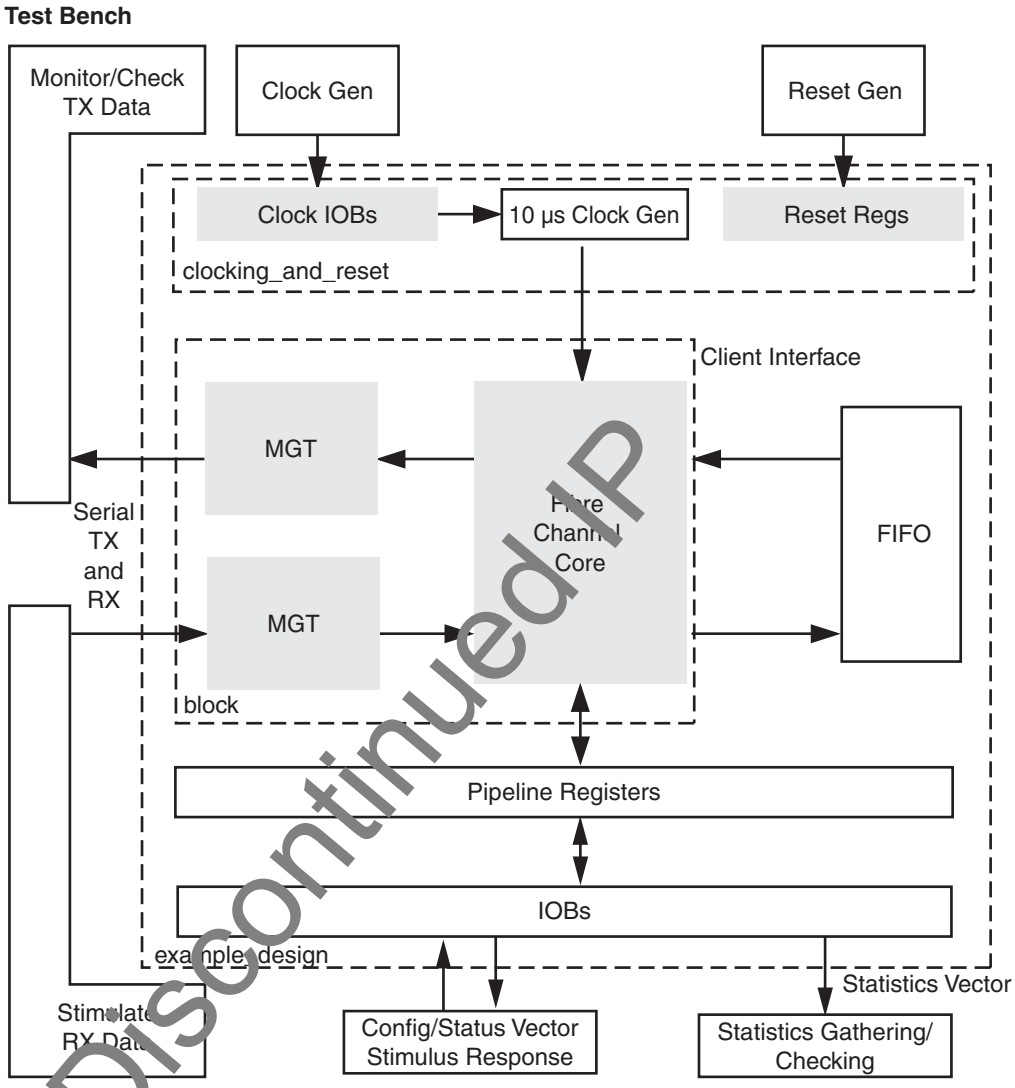
- The two device-specific RocketIO transceiver instances

  Note: The multi-speed core has a complex clocking/transceiver scheme. You are especially discouraged from changing this in your design.

  Note: The multi-speed core in Virtex-4 devices requires only a single transceiver instance.
- Clock management logic, including DCMs and Global Clock Buffer instances
- Registers to synchronize Reset signals
- 10 µs clock generation logic
- Client Loopback FIFO
- Pipeline registers and IOBs on Configuration Vector, Configuration Status Vector and Statistics Vector signals

Sections labeled DO NOT MODIFY in the wrapper appear where modifications can break the example design.

Demonstration Test Bench



Note: Alteration of parameters, connections, or placement of the shaded blocks may alter the functionality and/or performance of the core.

Figure 3-8: **Demonstration Test Bench for Fibre Channel with No Management Interface**

The demonstration test bench is a simple VHDL or Verilog verification tool which exercises 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.

With no Statistics Gathering block, it is up to the test bench to keep local statistics counters which are incremented when the appropriate bit(s) of the Statistics Vector are asserted for a clock cycle. The final values of these counters are checked for consistency at the end of the simulation.

With no Management Interface available, all accesses to the Configuration Registers are made directly through the Configuration Vector (in) and Configuration Status Vector (out) in the test bench.

Inserting Errors in the Demo Test Bench

It is possible to insert both word and bit-errors in the Stimulate Rx Data BFM using the ERROR_TYPE parameter in the test bench. It is also possible to trigger some of the top-level PHY signals with this feature.

To explicitly show the FC core reacting to non-standard stimulation using the ERROR_TYPE parameter in the demonstration test bench, one restriction applies: when this parameter is set to non-zero, only a single run through the test bench is possible. The SINGLE_RUN parameter in the test bench is ignored.

Verilog Test Bench

For the Verilog test bench, edit the `demo_tb.v` file to change the ERROR_TYPE parameter to be a value from 0 to 9. The mappings are shown in Table 3-10.

VHDL Test Bench

For the VHDL test bench, edit the `demo_tb.vhd` file to change the ERROR_TYPE parameter to a value from 0 to 9. The mappings for these values are given in Table 3-10.

Selecting the Error

Table 3-10 shows the ERROR_TYPE parameter options.

Table 3-10: **Effect of ERROR_TYPE Parameter on Simulation**

ERROR_TYPE	Effect of parameter
0	No errors introduced.
1	R_T_TOV expires (having first been set to a low value); equivalent to Event Timeout in PSM.
2	Asserts RX_LOS (Loss of signal) for > R_T_TOV.
3	Asserts TX_FAULT (Transmit Fault from the PHY).
4	Loss of Sync > R_T_TOV (having first been set to a low value).
5	Each frame has a Bad CRC.
6	Each frame has a Bad SOF (No frames will be retransmitted in this case).
7	Each frame has a Bad EOF.
8	Assert ApplyBackPressure (A single R_RDY will appear and then no more).
9	Client Underrun (EOFa will appear on TX frames).

Effect of ERROR_TYPE Parameter on Simulation

With ERROR_TYPE = 1, 2, 3, and 4, the simulation is self-explanatory.

With ERROR_TYPE = 5, each frame received by the core will have a bad CRC. On re-transmission, the EOF from the original frames is replaced with a EOFni by the core due to the incorrect CRC presented.

With ERROR_TYPE = 6, no frames will be received by the core as each has a bad SOF. Because an SOF is never recognized for re-transmission, no frames will be retransmitted by the core.

With ERROR_TYPE = 7, each frame received by the core will have a bad EOF. On re-transmission, the EOF from the original frames is replaced with a EOFa by the core.

With ERROR_TYPE = 8, a single R_RDY is transmitted by the core in response to the first frame received. After this, the ApplyBackPressure signal is driven high. No more R_RDYs will be transmitted and the count of outstanding R_RDYs to be sent is checked at the end of simulation.

With ERROR_TYPE = 9, each frame received by the core will have a missing EOF. On re-transmission, the missing EOF will cause an Underflow on the Client side and an EOF will be inserted by the core. This is slightly artificial in that the FIFO is designed to react this way to a missing EOF, but this illustrates the error condition.