- THIS IS A DISCONTINUED IP CORE -

LogiCORE™ IP Fibre Channel Arbitrated Loop v2.3

Getting Started Guide

UG220 March 24, 2008

XILINX®

Revision History

Release Date	Version	Description
1/18/06	1.0	Initial Xilinx release.
9/18/06	2.0	Core version 2.1 and Xilinx tools 8.2i; limited early access release.
2/15/07	2.1	Core version 2.1; Xilinx tools 9.1i.
8/8/07	2.2	Core version 2.2; Xilinx tools 9.2i.
3/24/08	2.5	Core version 2.3; Xilinx tools 10.1

Table of Contents

XILINX®

Preface

About This Guide

The *Fibre Channel Arbitrated Loop v2.3 Getting Started Guide* provides information about what you need to begin using the LogiCORE™ IP Fibre Channel Arbitrated Loop (FC-AL) core also serves as the reference document for the example design provided with the core.

- 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 how to obtain the core, lists references to related material, and provides contact information for obtaining technical support and providing feedback to Xilinx.
- Chapter 2, "Licensing the Core" describes available licensing options and provides instructions for obtaining and installing a license file.
- Chapter 3, "Fibre Channel Arbitrated Loop Example Design" provides instructions for quickly generating the core and running 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	`speed grade: - 100`
`Courier bold`	Literal commands you enter in a syntactical statement	**`ngdbuild`** `design_name`
Italics	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.
Shading	Unsupported or reserved items	This feature is not supported
<Brackets>	User-defined variable.	<project directory>

Convention	Meaning or Use	Example
Square brackets [ ]	Optional entry or parameter, with the exception of bus specifications. For bus specifications, brackets are required, for example **bus[7:0]**.	**ngdbuild** [option_name] design_name
Braces { }	A list of items from which you must choose one or more	**lowpwr ={on|off}**
Vertical bar |	Separates items in a list of choices	**lowpwr ={on|off}**
Vertical ellipsis . . .	Omitted repetitive material	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Omitted repetitive material	**allow block** block_name loc1 loc2 ... locn;
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returns 45524943h
	An '_n' means the signal is active low	usr_teof_n is active low

Online 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 "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.

XILINX®

Chapter 1

Introduction

The FC-AL 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 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 AL Release Notes (readme.txt)* file accompanying the product.

System Requirements

Windows

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

Linux

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

Software

- Xilinx ISE™ v10.1

Obtaining the Core

The FC-AL core is a Xilinx CORE Generator™ IP core included in the latest IP Update on the Xilinx IP Center. Details about the IP Update, including information you need to access the core, can be found on the FC-AL product page.

The core is delivered with a default installed license, so that you can try out the core generation GUI and 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-AL core is delivered as a fully verified, pre-implemented netlist, allowing the designer to focus on design details. The challenge associated with implementing a complete Fibre Channel 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 FAE at www.xilinx.com/company/sales/sales_offices.htm.

More extensive design assistance is also available through Xilinx Design Services at www.xilinx.com/xds/index.htm.

Additional Core Resources

For additional information about the FC-AL core, including the latest support documents and known issue updates, see the following documents:

- *Fibre Channel Arbitrated Loop Release Notes*
- *Fibre Channel Arbitrated Loop Data Sheet*
- *Fibre Channel Arbitrated Loop User Guide*

These documents are available from the FC-AL product page lounge.

Technical Support

For technical support, go to

www.xilinx.com/support/clearexpress/websupport.htm.

From this page, you can create a WebCase to route your inquiry to a support engineer with specific expertise using the FC-AL core.

Xilinx provides technical support for this product when used according to guidelines described in the *Fibre Channel Arbitrated Loop User Guide* and the *Fibre Channel Arbitrated Loop 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-AL core and the documentation provided with the core.

Document Feedback

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

Please be sure to include the following information:

- 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.

Fibre Channel Arbitrated Loop Core Feedback

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

Please be sure to provide the following information:

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

XILINX®

Chapter 2

Licensing the Core

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

Before you Begin

This chapter assumes you have installed the core using either the CORE Generator IP Software Update installer or by performing a manual installation after downloading the core from the web. For information about installing the core, see the FC-AL product page.

License Options

The FC-AL core provides three licensing options. After installing the core, choose a license option.

Simulation Only

The Simulation Only Evaluation license is provided with the Xilinx CORE Generator tool by default, and does not require an electronic license key. This license lets you assess the core functionality with either the provided example design or alongside your own design and demonstrates the various interfaces on the core in simulation. (Functional simulation is supported by a structural model generated by the CORE Generator tool.)

Full System Hardware Evaluation

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

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

You can obtain the Full System Evaluation License in one of the following ways, depending on the core:

- Register on the Xilinx IP Evaluation page and fill out a form to request an automatically generated evaluation license.
- Contact your local Xilinx FAE to request a Full System Hardware Evaluation license key.

Click Evaluate on the FC-AL product page for information about how to obtain a Full System Hardware Evaluation License.

Full

The Full license is provided when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

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

Obtaining Your License

Note: The Simulation Evaluation license is provided through the Xilinx CORE Generator software and requires no action on your part.

Obtaining a Full System Hardware Evaluation License

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

- Navigate to the FC-AL product page.
- Click Evaluate; then click Full System Hardware Evaluation.
- Follow the onscreen instructions to both download the CORE Generator files (delivered as an IP Update) and satisfy any additional requirements associated with the license.

Obtaining a Full License

The Full license is available for the Fibre Channel Arbitrated Loop core at no charge after purchase of the Fibre Channel Point to Point core. To request a Full license key, go to the FC-AL product page. From the product page, click Order and Register to register and request access to the lounge.

- Xilinx will review your access request and typically grants access to the lounge in 48 hours. (Contact Xilinx Customer Service if you need faster turnaround.)
- After you receive confirmation of lounge access, click Access Lounge on the Fibre Channel Arbitrated Loop product page and log in.
- Follow the instructions in the lounge to fill out the license request form; then click Submit to automatically generate the license. An e-mail containing the license and installation instructions will be sent to you immediately.

Installing Your License File

After selecting either the Full System Hardware or Full license option, an email will be sent to you that provides instructions for installing your license file. In addition, information about advanced licensing options and technical support is provided.

XILINX®

Chapter 3

Fibre Channel Arbitrated Loop Example Design

This chapter demonstrates how to generate and the FC-AL core using the default parameters in the CORE Generator. Detailed information about the files and directory structure created by the CORE Generator 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

The FC-AL example design includes the following:

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

The FC-AL example design is tested with Xilinx ISE™ v10.1, Mentor ModelSim® v6.3c, and Cadence® IUS v6.1, and Synopsys® vcs_mxY-2006.06-SP1.

In this design, the Client Rx port is connected to the Client Tx port through the FIFOs. With this arrangement, any uncorrupted frames received by the core are retransmitted without changes.

Figure 3-1 illustrates the FC-AL example design with default configurations.

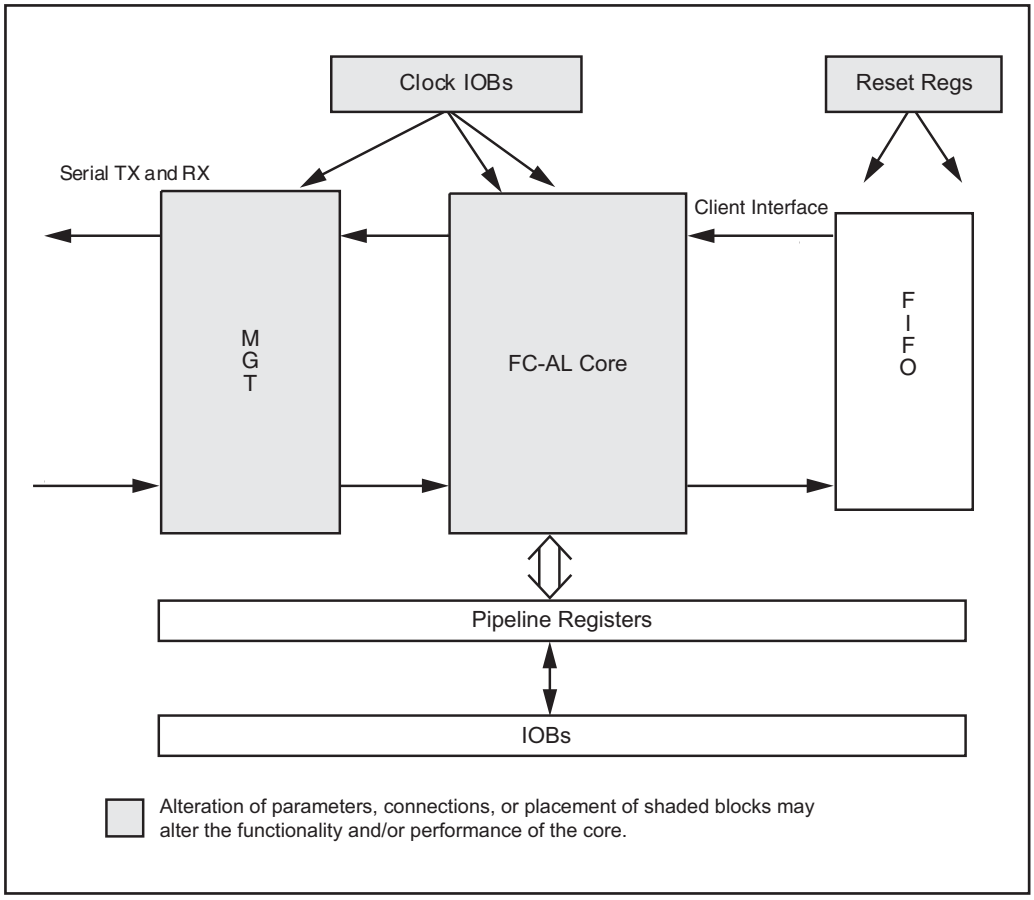

Figure 3-1: **FC-AL Example Design Default Configuration**

In the HDL Wrapper, the FC-AL serial interface is brought out to the Virtex™-4 and Virtex-II Pro FPGA RocketIO™ Multi-Gigabit Transceivers (MGTs) in the target FPGA device. Clocking and support circuitry is also instantiated in the wrapper.

The other signals going to and coming from the core (such as 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 may be encountered when trying to drive the interface signals on the core directly from IOBs. In a user design, these interface signals would normally be driven by the user-supplied logic located within the FPGA fabric, and would not require pipelining.

The FIFO used in the wrapper can be replaced with the user-supplied back-end 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 Fibre Channel Port or IP core.

Generating the Core

The FC-AL core is generated using the CORE Generator, which provides an easy-to-use Graphical User Interface (GUI)) for selecting options and parameters. For information

about starting and using the CORE Generator, see the *CORE Generator Guide*, available from www.xilinx.com/support/software_manuals.htm.

To generate the FC-AL core:

1. Start the CORE Generator using one of the following methods:

   Windows

   ♦ Choose Programs > Xilinx ISE 10> Accessories > CORE Generator.

   Linux

   ♦ In a shell prompt, run `coregen`.
2. Create a new project.
3. From the Project options, select a silicon family that supports the FC-AL core. (At this time, Virtex -II Pro, Virtex-4, and Virtex-5 devices are supported.)

   Note: If an unsupported silicon family or part is selected, the core is not available for customization. See the *Fibre Channel Arbitrated Loop Data Sheet* for more information.
4. In the Design Entry section of the Project options, select either VHDL or Verilog.
5. For Vendor, select Other.
6. Navigate to the FC-AL core in one of the following locations:

   ```
   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.
8. If necessary, click OK to exit the license limitation message dialog box.

   The FC-AL core main screen appears.

Note: If an unsupported silicon family or part is selected, the core is not available for customization. The smallest devices supported in CORE Generator are Virtex-II Pro 2VP20, Virtex-4 4VFX20 and Virtex-5 LX30T. See the *Fibre Channel Arbitrated Loop Data Sheet* for information about supported devices.

9. Enter a name for the core in the Component Name field.

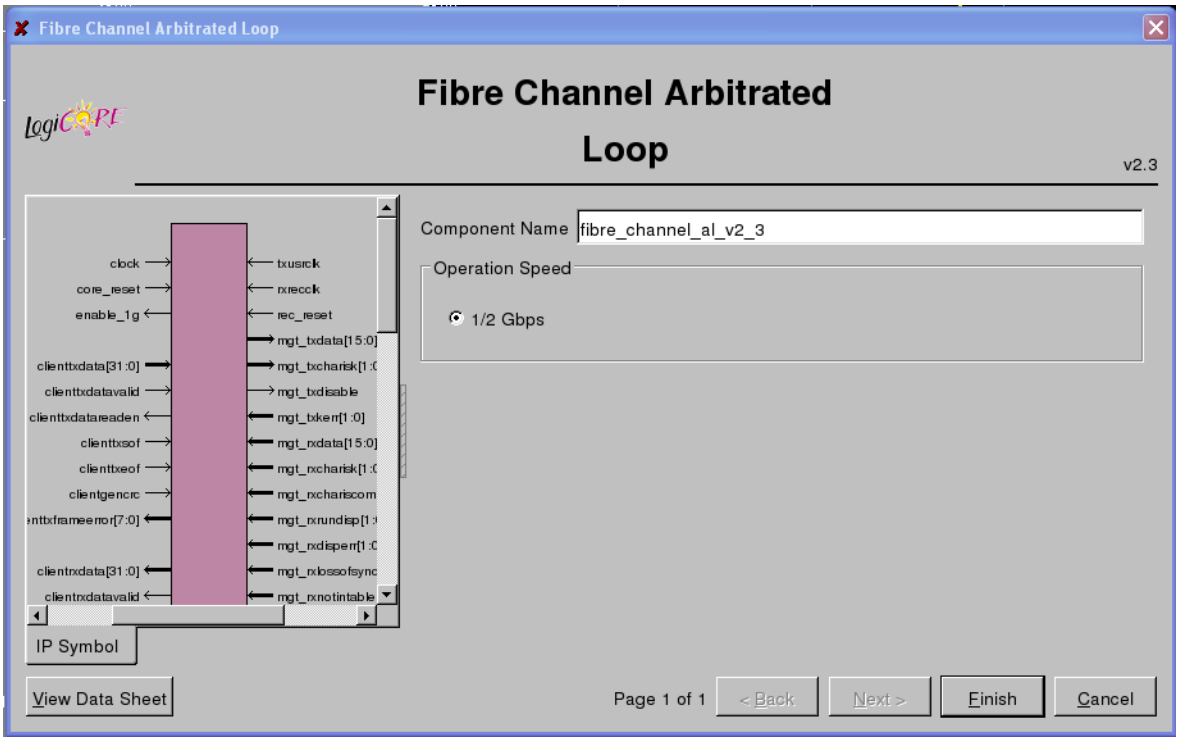

Figure 3-2: **FC-AL Main Screen**

10. Select the required speed options and click Finish to generate the core using the default parameters.

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 sections.

Directory Structure and File Descriptions

This section provides detailed information about the example design, including a description of files and the directory structure generated by the Xilinx CORE Generator, the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench. Note that the implement and timing subdirectories are only present when the core is generated with a Full System Hardware Evaluation license or a Full license.

<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 (or whichever, if it's only one) 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
Simulation files.

<project directory>

The project directory contains all the CORE Generator project files.

Table 3-1: **Project Directory**

Name	Description
<project_dir>	
<component_name>.ngc	NGC Implementation netlist for the core. Defines how the core is implemented. Used as input to the Xilinx implementation tools.

Table 3-1: **Project Directory *(Continued)***

Name	Description
<component_name>.v[hd]	Verilog or VHDL structural simulation model, used to support Verilog/VHDL functional simulation of a core. The Verilog/VHDL wrapper passes customized parameters to the generic core simulation model.
<component_name>.{veo|vho}	VHDL or Verilog instantiation template.
<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 CORE Generator for each core 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 the output files produced when a customized core is generated in the CORE Generator.

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_al_readme.txt	FC-AL release notes file.

Back to Top

<component name>/example design

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

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

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_top.v[hd]	The top-level Verilog/VHDL file for the example design.

Table 3-3: **Example Design Directory *(Continued)***

Name	Description
FCMGT.v[hd]	The wrapper file for the Fibre Channel MGT transceivers, which is instanced in the top-level Verilog/VHDL file.
<component_name>_mod.v	Verilog module declaration for the core instance in the example design.
cal_block_v1_4_1.v[hd]	(Virtex-4 devices only) The current GT11 Calibration Block instance in the wrapper file.
gt11_speed_controller.v[hd]	(Virtex-4 devices only) A state machine to reprogram the GT11 for different speeds, instanced in the wrapper file.
gtp_speed_controller.v[hd]	(Virtex-5 devices only) A state machine to reprogram the GTP for different speeds, instanced in the wrapper file.
fc_al_fifo.v[hd]	A loopback FIFO design instanced in the top level (above).
rx_client_fifo.v[hd]	A FIFO design instanced in the loopback FIFO (above).
tx_client_fifo.v[hd]	A FIFO design instanced in the loopback FIFO (above).
ext_stats.v[hd]	A Statistics block instanced in the top level (above).
ext_stats_counter.v[hd]	A counter design instanced in the Statistics block (above).
<component_name>_top.ucf	The User Constraints File (UCF) for the core and the example design.

Back to Top

<component name>/doc

The doc directory contains the FC-AL PDF documentation provided with the core.

Table 3-4: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
fibre_channel_al_ds518.pdf	FC-AL data sheet
fibre_channel_al_gsg220.pdf	FC-AL Getting Started Guide
fibre_channel_al_ug211.pdf	FC-AL User Guide

Back to Top

<component name>/implement

The implement directory contains the core implementation script files.

Table 3-5: **Implement Directory**

Name	Description
<project_dir>/<component_name>/implement	
xst.prj	The XST project file for the example design; enumerates all the HDL files that need to be synthesized.
xst.scr	The XST script file for the example design.
implement.sh	A Linux shell script that processes the example design through the Xilinx tool flow.
implement.bat	A Windows batch file that processes the example design through the Xilinx tool flow. See "Implementation Script," page 25 for more information.

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 files through the Xilinx Implementation tools. After this is complete, the directory contains two files for timing simulations.

Table 3-6: **Results Directory**

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

Back to Top

<component name>/simulation

The simulation directory and its sub-directories contain the files necessary to test a Verilog/VHDL implementation of the example design.

Table 3-7: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
demo_tb.vhd	The Verilog/VHDL demonstration test fixture for the FC-Al core.

Back to Top

simulation/functional

The functional directory contains the files necessary to test a VHDL functional implementation of the example design.

Table 3-8: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_mti.do	A ModelSim macro file that compiles the Verilog/VHDL sources then runs the simulation to completion.
wave_mti.do	A ModelSim macro file that opens a wave window and adds interesting signals to it; called by the simulate_mti.do macro file.
simulate_ncsim.sh	A Linux script file that compiles the VHDL sources with NCSIM then runs the simulation to completion.
wave_ncsim.sv	A NCSIM macro file that opens a waveform window and adds interesting signals to it; called by the simulate_ncsim.sh script.

Back to Top

simulation/timing

The timing directory contains the files necessary to test a VHDL timing-based implementation of the example design.

Table 3-9: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/timing	
simulate_mti.do	A ModelSim macro file that compiles the Verilog/VHDL sources then runs the simulation to completion.

Table 3-9: **Functional Directory *(Continued)***

Name	Description
wave_mti.do	A ModelSim macro file that opens a wave window and adds interesting signals to it; called by the simulate_mti.do macro.
simulate_ncsim.sh	A Linux script file that compiles the Verilog/VHDL sources with NCSIM then runs the simulation to completion.
wave_ncsim.sv	A NCSIM macro file that opens a waveform window and adds interesting signals to it. It is called by the simulate_ncsim.sh script.

Back to Top

Implementing the Example Design

After the core is 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.

To implement the example design:

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

  Windows

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

  Linux

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

This command starts 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 CORE Generator. 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 either a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator or a SWIFT-compliant simulator.

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

Virtex-4 and Virtex-II Pro Devices

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

Verilog Simulation

To run a Verilog functional simulation of the example design:

1. Launch the ModelSim simulator 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:

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

Timing Simulation

Gate-level simulation is supported through the dynamic generation of a gate-level simulation model in ISE.

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 Verification Design Guide*.

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

Verilog Simulation

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

1. Launch the ModelSim simulator 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 the ModelSim simulator 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. . 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.

Implementation and Test Scripts

Implementation Script

The implement directory is only produced when CORE Generator is 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.

Windows

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

Linux

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

The `/results` subdirectory is created in the `/implement` directory, where all implementation files and tool reports are located.

The implement script performs the following steps.

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

Test Scripts for Timing Simulation

There are two test scripts for Timing Simulation.

ModelSim

The first 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`

NCSIM

The second test script is for NCSIM and automates the timing-based simulation of the test bench.

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

Timing Simulation

Both test scripts perform the following tasks:

1. Compiles the gate level netlist.
2. Compiles the example design files.
3. Compiles the demonstration test bench.

4. Starts a simulation of the test bench including timing information.
5. Opens a Wave window and adds some signals of interest to it (wave_mti.do/wave_ncsim.sv).
6. Runs the simulation to completion.

Test Scripts for Functional Simulation

There are two test scripts for Functional Simulation.

ModelSim

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

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

NCSIM

The second test script is for NCSIM and automates the functional simulation of the test bench.

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

Timing Simulation

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

The demonstration test bench performs the following steps. Figure 3-4, page 30 shows the architecture of the demonstration test bench.

1. Resets the core.
2. Stimulates the Receiver with IDLE codes.
3. Sets AL_TIME to 12.5 µs.
4. Sets the Port Name to 0x1000000A35002243.
5. Sets a Preferred AL_PA of 0x02 and setup the core as an N Port to obey the Access Fairness algorithm and support positional mapping in the login sequence (LILP and LIRP frames are used).
6. Enables the MGT transceiver and set the speed. The first run-through is at the higher of the two speeds and the second is run at the lower speed.
7. Waits for the Loss of Sync state machine to reach Synchronized state.
8. The core begins to transmit LIPF7 codes.

- Waits for a while and transmits LIPF7 to core.
- Waits for core to transmit IDLE codes.
- Waits for a while and transmits IDLE to core.

9. Transmits 5 LISM frames to core with a higher Port Name and AL_PA = 0xEF.
   - The core repeatedly transmits its own LISM frame.
   - Re-transmits up to 200 LISM frames back to core until core transmits ARBF0 codes.
10. Transmits IDLE codes to core.
11. Transmits ARBF0 codes to core and waits for core to stop transmitting ARBF0 codes.
12. The core now begins the LIFA/LIHA/LIPA/LISA sequence, followed by the LILP/LIRP sequence of frames, and the test bench responds as required.
    - The test bench receives the AL_PA of 0x04 during the LISA phase.
    - The core assumes the AL_PA of 0x01.
13. Waits for the core to transmit the CLS code.
    - Transmits the CLS code to core.
14. Waits for the core to enter MONITORING state.
15. Writes the value 4 to the LocalBBCredit register in the core.
16. Transmits ARB0404 to the core.
    - Waits for ARB0404 to be re-transmitted by core.
17. Transmits OPN0104 to core.
18. Transmits 4 RRDY codes to core.
    - Waits for core to transmit 4 RRDY codes.
19. Transmits 4 frames to core. The third frame has an intentional CRC error and is not re-transmitted by core (via loopback FIFO) but all other frames are re-transmitted.
    - Transmits 3 RRDY codes to core to return credit for frames received by test bench.
20. Transmits CLS code to core.
21. Sets the `RequestClose` bit in the AL Circuit Control register in the core.
    - Waits for the core to transmit CLS
22. Checks the statistics - the third frame has the CRC error.
23. Finally, change the speed of the core to the slower of the two speeds and repeat the entire process from the start.
24. End the simulation.

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 by the user. Note that alterations are not supported, as stated in the LogiCore IP License agreement.

Example Design Architecture

The follow sections outline the Fibre Channel Arbitrated Loop example design architecture, comprised of the HDL wrapper and the demonstration test bench.

Example HDL Wrapper

The example HDL wrapper contains the following:

- The 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.

Figure 3-3 illustrates the example HDL wrapper.

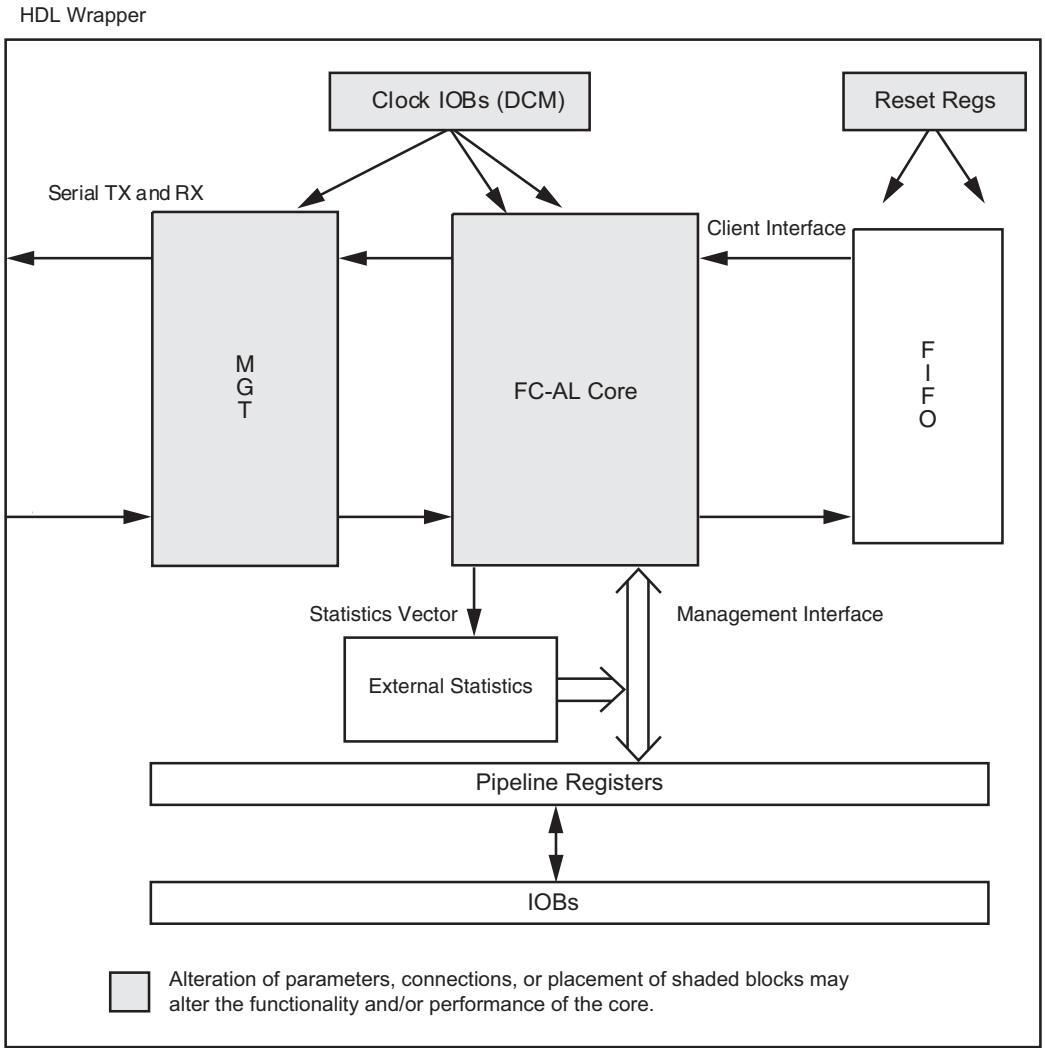

Figure 3-3: **Example HDL Wrapper for FC-AL Core**

Demonstration Test Bench Architecture

The demonstration test bench is a simple VHDL or Verilog verification tool that exercises the example design and the core. 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.

All accesses to the Configuration Registers are performed through the Management interface, as driven by the Management Interface Bus Functional Model (BFM) in the test bench. The Configuration Status Vector is used to detect when the core is in Synchronized state.

Figure 3-4 illustrates the demonstration test bench.

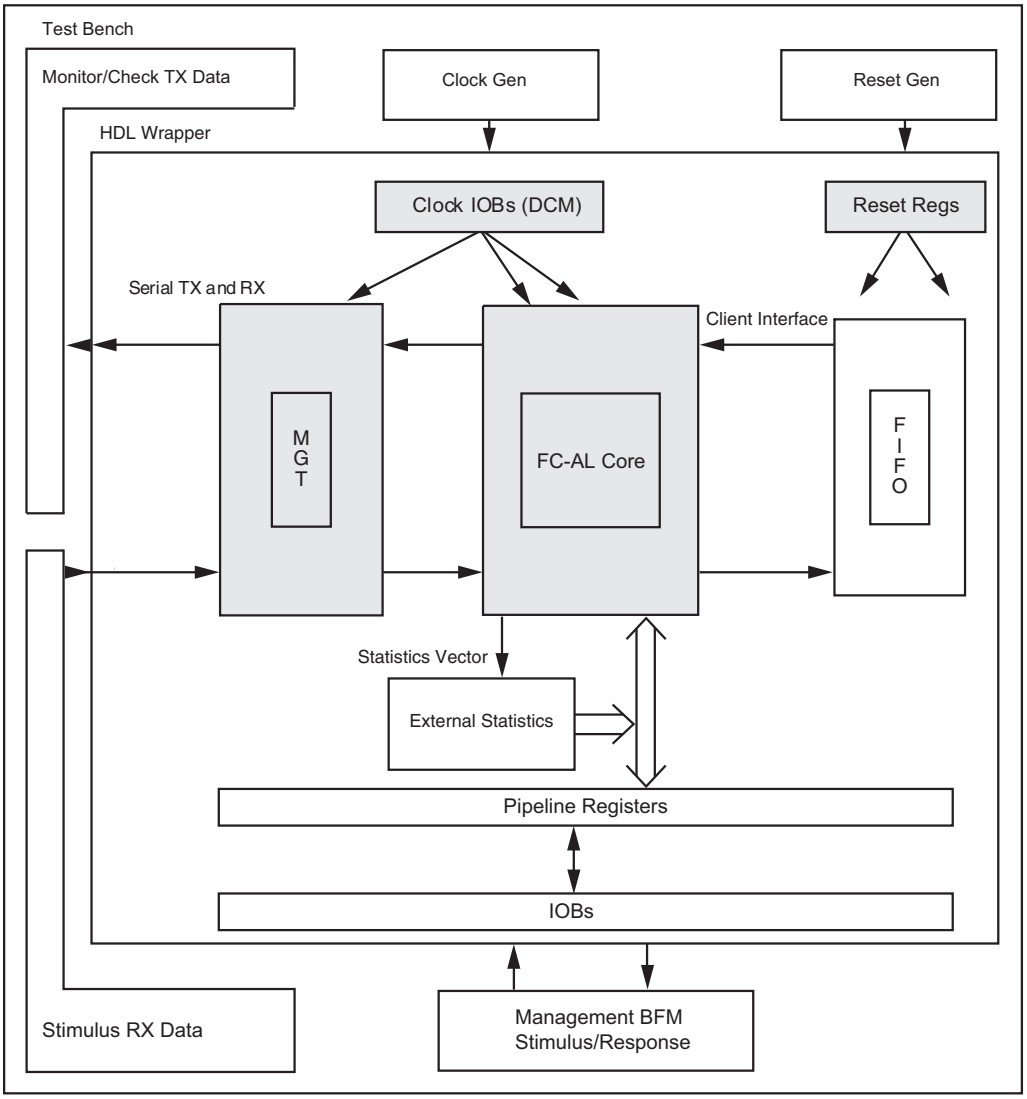

Figure 3-4: **Demonstration Test Bench for FC-AL Core**