UTOPIA Level 2 Slave MegaCore
Function User Guide

Contents

vi

Release Information

Table 1–1 provides information about this release of the Altera® UTOPIA Level 2 Slave MegaCore® function.

Table 1–1. UTOPIA Level 2 Slave Release Information

Item	Description
Version	9.0
Release Date	March 2009
Ordering Code	IP-UTOPIA2SL
Product ID	0016
Vendor ID	6AF7

For more information about this release, refer to the *MegaCore IP Library Release Notes and Errata*.

Altera verifies that the current version of the Quartus® II software compiles the previous version of each MegaCore function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release."

Device Family Support

MegaCore functions provide either full or preliminary support for target Altera device families:

■ *Full support* means the MegaCore function meets all functional and timing requirements for the device family and may be used in production designs.

■ *Preliminary support* means the MegaCore function meets all functional requirements, but may still be undergoing timing analysis for the device family; it may be used in production designs with caution.

Table 1–2 shows the level of support offered by the UTOPIA Level 2 Slave MegaCore function to each Altera device family.

Table 1–2. Device Family Support (Part 1 of 2)

Device Family	Support
Arria™ GX	Full
Cyclone®	Full
Cyclone II	Full
Cyclone III	Full
HardCopy® II	Full
Stratix®	Full

Table 1–2. Device Family Support (Part 2 of 2)

Device Family	Support
Stratix II	Full
Stratix II GX	Full
Stratix III	Full
Stratix IV	Preliminary
Stratix GX	Full
Other device families	No support

Features

■ Conforms to the UTOPIA Level 2, Version 1.0 specification

■ 8- or 16-bit UTOPIA bus operation and local bus widths of 8 or 16 bits

■ Single-PHY (SPHY) operation, with both octet- and cell-level handshaking

■ Multi-PHY (MPHY) operation, with single clav signal

■ Parity generation and detection

■ Optional cell discard on parity error detection

■ Internal 4-cell first-in first-out (FIFO) buffers supported for both transmit and receive

■ Atlantic™ interface—packet-based interface that is compatible with other Altera cell and packet MegaCore functions

■ Easy-to-use IP Toolbench interface

■ IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators

■ Support for OpenCore Plus evaluation

General Description

The Altera UTOPIA Level 2 Slave MegaCore function is designed for use in physical layer (PHY) devices that transfer data to and from asynchronous transfer mode (ATM) devices using the standard UTOPIA bus.

The UTOPIA Level 2 Slave MegaCore function comprises a separate transmitter and receiver; both support SPHY and MPHY operation modes. SPHY mode supports octet- or cell-level handshake; MPHY mode supports cell-level handshake.

The transmitter is polled by the ATM layer to determine whether it is ready to receive data transfers. The transmitter accepts cells from the ATM layer via the UTOPIA bus interface, and sends them to the PHY devices. It detects and discards cells that are too short, and discards excess bytes from cells that are too long. It also checks for parity errors on the UTOPIA bus, and there is an option to discard cells with detected parity errors.

The receiver is polled by the ATM layer to determine whether it is ready to transfer data. The receiver accepts cells from the PHY, and sends them to the ATM layer device via the UTOPIA bus interface. There is an option to generate parity information for the UTOPIA bus.

Atlantic Interface

The Atlantic interface allows a consistent interface between all Altera cell and packet MegaCore functions. The Atlantic interface is only designed to support a point-to-point connection. You must choose whether you use the local or Atlantic interface. Figure 1–1 shows examples of the Atlantic interface.

Figure 1–1. Atlantic Interface
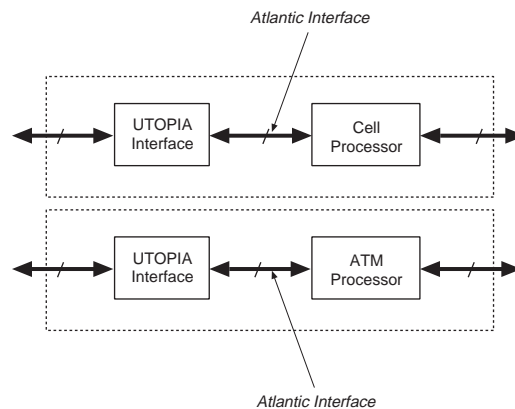
For more information on the Atlantic interface, refer to *FS 13: Atlantic Interface*.

OpenCore Plus Evaluation

With Altera's free OpenCore Plus evaluation feature, you can perform the following actions:

■ Simulate the behavior of a megafunction (Altera MegaCore function or AMPPSM megafunction) within your system

■ Verify the functionality of your design, as well as evaluate its size and speed quickly and easily

■ Generate time-limited device programming files for designs that include megafunctions

■ Program a device and verify your design in hardware

You only need to purchase a license for the megafunction when you are completely satisfied with its functionality and performance, and want to take your design to production.

For more information on OpenCore Plus hardware evaluation using the UTOPIA Level 2 Slave, see "OpenCore Plus Time-Out Behavior" on page 3–1 and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Performance and Resource Utilization

Table 1–3 and 1–4 show the resource utilization for the following devices using the Quartus® II software version 8.0:

- Cyclone II

- Stratix III

The UTOPIA level 2 slave runs above 50 MHz in all Cyclone, Cyclone II, Cyclone III, Stratix, Stratix II, and Stratix III devices.

Table 1–3. Performance—Cyclone II Device

Parameters				LEs	Memory Blocks
Data Flow Direction	**UTOPIA Width (local bits)**	**Local Cell Size**	**Mode**		
Receiver	8	52	MPHY using `parity_generate` and `discard_on_error`	190	1
	16	54	MPHY using `parity_generate` and `pipeline_user_interface`	173	1
Transmitter	8	52	MPHY using `parity_check` and `discard_on_error`	228	1
	16	54	MPHY using `parity_check` and `pipeline_user_interface`	228	1

Table 1–4. Performance—Stratix III Device

Parameters				ALUTs	Memory Blocks
Data Flow Direction	**UTOPIA Width (local bits)**	**Local Cell Size**	**Mode**		
Receiver	8	52	MPHY using `parity_generate` and `discard_on_error`	117	1
	16	54	MPHY using `parity_generate` and `pipeline_user_interface`	114	1
Transmitter	8	52	MPHY using `parity_check` and `discard_on_error`	129	1
	16	54	MPHY using `parity_check` and `pipeline_user_interface`	135	1

Design Flow

To evaluate the UTOPIA Level 2 Slave MegaCore function using the OpenCore Plus feature, include these steps in your design flow:

1. Obtain and install the UTOPIA Level 2 Slave MegaCore function.

   The UTOPIA Level 2 Slave MegaCore is part of the MegaCore IP Library, which is distributed with the Quartus II software and downloadable from the Altera website, www.altera.com.

   For system requirements and installation instructions, refer to *Quartus II Installation & Licensing for Windows Workstations*.

   Figure 2–1 shows the directory structure after you install the UTOPIA Level 2 Slave MegaCore function, where *<path>* is the installation directory. The default installation directory on Windows is **c:\altera\90**; on Linux it is **/opt/altera90**.

Figure 2–1. Directory Structure


```
<path>
Installation directory.
   ip
   Contains the Altera MegaCore IP Library and third-party IP cores.
      altera
      Contains the Altera MegaCore IP Library.
         common
         Contains shared components.
         utopia2_slave
         Contains the UTOPIA Level 2 Slave MegaCore function files and documentation.
            doc
            Contains the documentation for the MegaCore function.
            lib
            Contains encrypted lower-level design files.
            sim_lib
            Contains the MegaCore function simulation models.
               modelsim
               Contains the ModelSim simulation models.
               testbench
               Contains the testbench.
```

2. Create a custom variation of the UTOPIA Level 2 Slave MegaCore function using IP Toolbench.

   IP Toolbench is a toolbar from which you quickly and easily view documentation, specify parameters, and generate all of the files necessary for integrating the parameterized MegaCore function into your design.

3. Implement the rest of your design using the design entry method of your choice.

4. Use the IP Toolbench-generated IP functional simulation model to verify the operation of your design.

For more information on IP functional simulation models, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the *Quartus II Handbook.*.

5. Use the Quartus II software to compile your design.

☞ You can also generate an OpenCore Plus time-limited programming file, which you can use to verify the operation of your design in hardware.

6. Purchase a license for the UTOPIA Level 2 Slave MegaCore function.

After you have purchased a license for the UTOPIA Level 2 Slave MegaCore function, follow these additional steps:

1. Set up licensing.

2. Generate a programming file for the Altera® device(s) on your board.

3. Program the Altera device(s) with the completed design.

4. Perform design verification.

UTOPIA Level 2 Slave Walkthrough

This walkthrough explains how to create a UTOPIA Level 2 Slave MegaCore function using the Altera UTOPIA Level 2 Slave IP Toolbench and the Quartus II software. When you finish generating a UTOPIA Level 2 Slave MegaCore function, you can incorporate it into your overall project.

This walkthrough involves the following steps:

■ Create a New Quartus II Project

■ Launch IP Toolbench

■ Step 1: Parameterize

■ Step 2: Set Up Simulation

■ Step 3: Generate

Create a New Quartus II Project

You need to create a new Quartus II project with the **New Project Wizard**, which specifies the working directory for the project, assigns the project name, and designates the name of the top-level design entity.

To create a new project follow these steps:

1. Choose **Programs > Altera > Quartus II** *<version>* (Windows Start menu) to run the Quartus II software. You can also use the Quartus II Web Edition software.

2. Choose **New Project Wizard** (File menu).

3. Click **Next** in the **New Project Wizard Introduction** (the introduction does not display if you turned it off previously).

4. In the **New Project Wizard: Directory, Name, Top-Level Entity** page, enter the following information:

    a. Specify the working directory for your project. For example, this walkthrough uses the **c:\altera\projects\utopia2s_project** directory.

    ☞ The Quartus II software automatically specifies a top-level design entity that has the same name as the project. This walkthrough assumes that the names are the same.

    b. Specify the name of the project. This walkthrough uses **example** for the project name.

5. Click **Next** to close this page and display the **New Project Wizard: Add Files** page.

    ☞ When you specify a directory that does not already exist, a message asks if the specified directory should be created. Click **Yes** to create the directory.

6. Click **Next** to close this page and display the **New Project Wizard: Family & Device Settings** page.

7. On the **New Project Wizard: Family & Device Settings** page, choose the target device family in the Family list.

8. The remaining pages in the **New Project Wizard** are optional. Click **Finish** to complete the Quartus II project.

You have finished creating your new Quartus II project.

Launch IP Toolbench

To launch IP Toolbench in the Quartus II software, follow these steps:

1. Start the MegaWizard® Plug-In Manager by choosing **MegaWizard Plug-In Manager** (Tools menu). The **MegaWizard Plug-In Manager** dialog box displays.

    ☞ Refer to Quartus II Help for more information on how to use the MegaWizard Plug-In Manager.

2. Specify that you want to create a new custom megafunction variation and click **Next**.

3. Expand the **Communications** > **UTOPIA** directory then click **UTOPIA Level 2 Slave**.

4. Select the output file type for your design; the wizard supports VHDL and Verilog HDL.

5. The MegaWizard Plug-In Manager shows the project path that you specified in the **New Project Wizard**. Append a variation name for the MegaCore function output files *<project path>\<variation name>*. Figure 2–2 on page 2–4 shows the wizard after you have made these settings.

Figure 2–2. Select the MegaCore Function
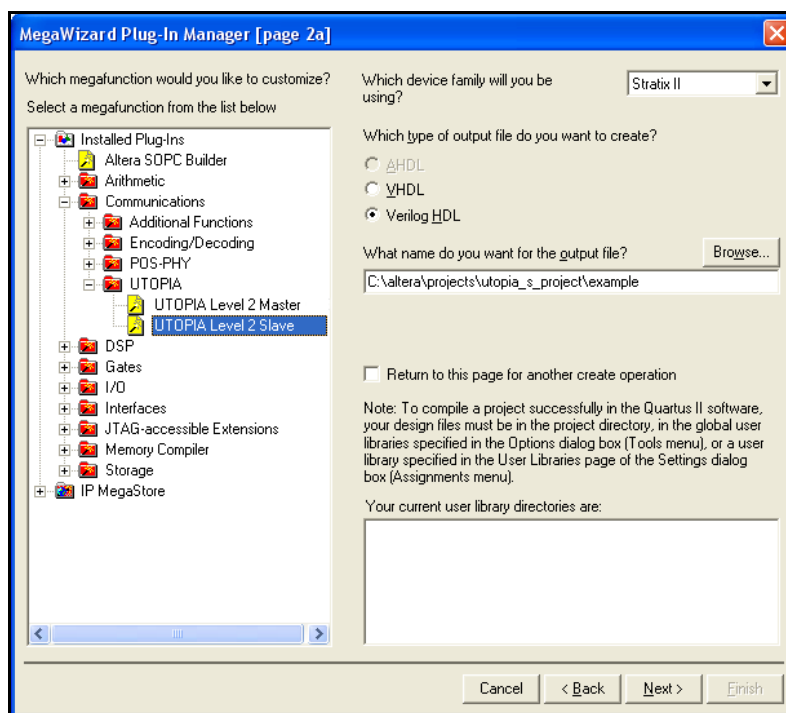
6. Click Next to launch IP Toolbench.

Step 1: Parameterize

To parameterize your MegaCore function, follow these steps:

1. Click **Step 1: Parameterize** in IP Toolbench (see).

Figure 2–3. IP Toolbench—Parameterize
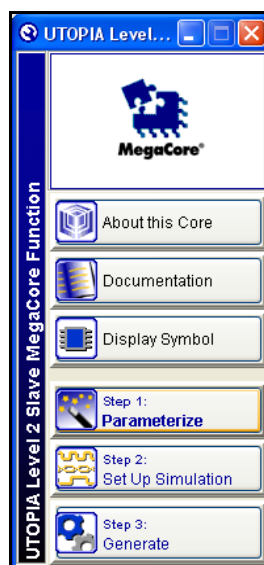
2. Select whether you wish to create a transmitter or receiver (see Figure 2–4). If you require an Atlantic™ local interface, turn on the Atlantic Local Interface check box. Click Next.

Figure 2–4. Select a Transmitter or Receiver

3. Choose the parameters that define the specific UTOPIA slave MegaCore function you wish to implement (see Figure 2–5). See Table 3–1 on page 3–2 for a description of the parameters. IP Toolbench allows you to select only legal combinations of parameters. Click Finish when you are done.

Figure 2–5. Select the Parameters
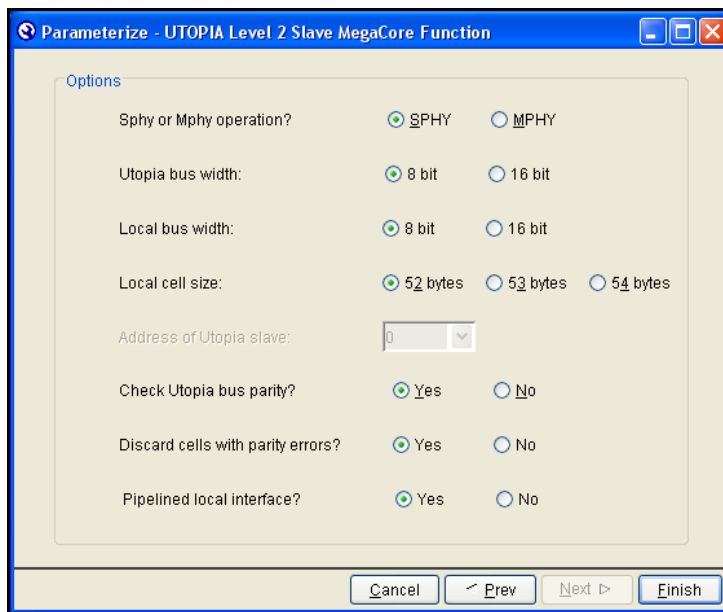
Step 2: Set Up Simulation

An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software. It allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.

> **CAUTION**
> You may only use these simulation model output files for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis will create a nonfunctional design.

To generate an IP functional simulation model for your MegaCore function, follow these steps:

1. Click **Step 2: Set Up Simulation** in IP Toolbench (see Figure 2–6).

Figure 2–6. IP Toolbench—Set Up Simulation
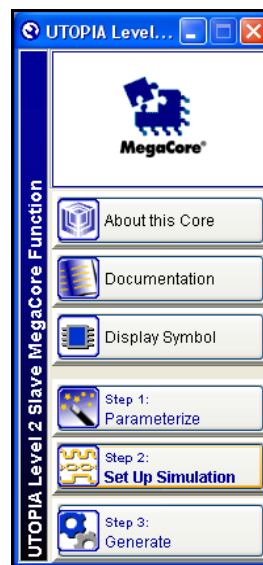
2. Turn on **Generate Simulation Model** (see Figure 2–7).

Figure 2–7. Generate Simulation Model
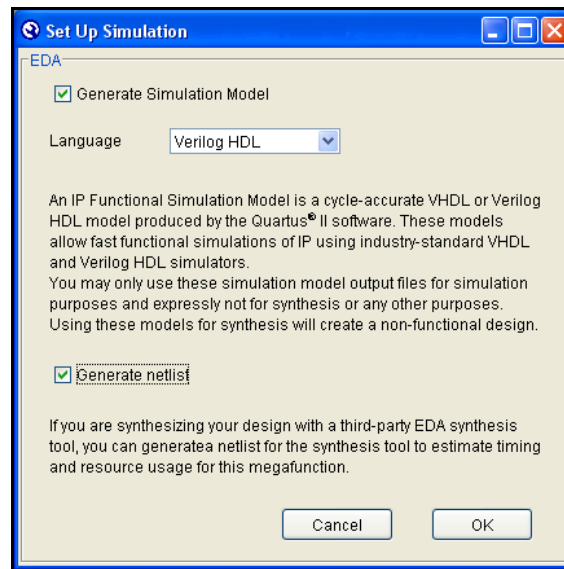
3. Choose the language in the Language list.

4. Some third-party synthesis tools can use a netlist that contains only the structure of the MegaCore function, but not detailed logic, to optimize performance of the design that contains the MegaCore function. If your synthesis tool supports this feature, turn on **Generate netlist**.

5. Click **OK**.

Step 3: Generate

To generate your MegaCore function, follow these steps:

1. Click **Step 3: Generate** in the IP Toolbench (see Figure 2–8).

Figure 2–8. IP Toolbench—Generate

Figure 2–9 on page 2–8 shows the generation report.

Figure 2–9. Generation Report

Table 2–1 describes the generated files and other files that may be in your project directory. The names and types of files specified in the IP Toolbench report vary based on whether you created your design with VHDL or Verilog HDL.

Table 2–1. Generated Files *(Note 1)*

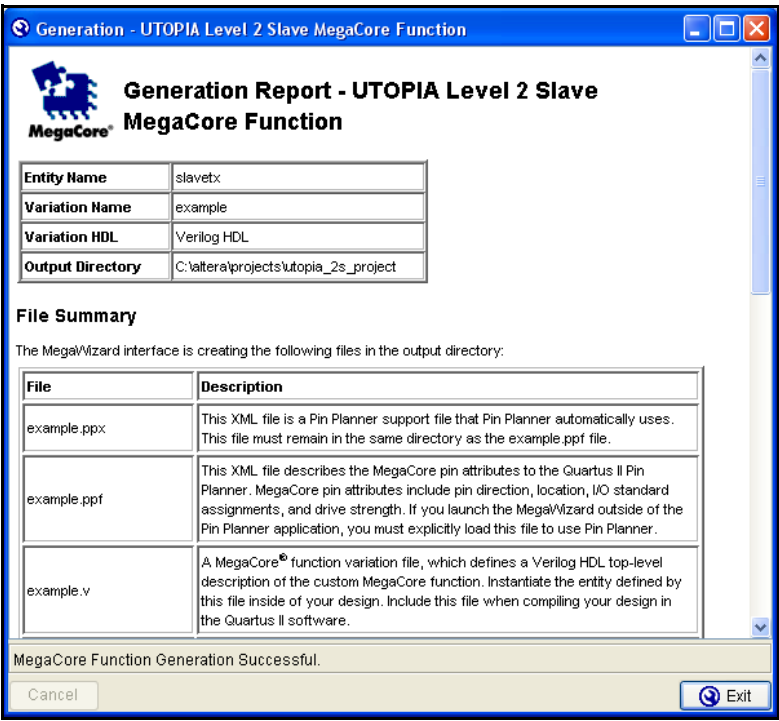| Extension *(2)*	Description
*<variation name>***syn.v** or *<variation name>***syn.vhd**	A timing and resource netlist for use in some third-party synthesis tools.
*<variation name>***.bsf**	Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.
*<variation name>***.html**	MegaCore function report file.
*<variation name>***.ppf**	This XML file describes the MegaCore function pin attributes to the Quartus II Pin Planner. MegaCore function pin attributes include pin direction, location, I/O standard assignments, and drive strength. If you launch the MegaWizard outside of the Pin Planner application, you must explicitly load this file to use Pin Planner.
*<variation name>***.ppx**	This XML file is a Pin Planner support file that Pin Planner automatically uses. This file must remain in the same directory as the *<variation name>***.ppf** file.
*<variation name>***.vhd** or *<variation name>***.v**	A MegaCore function variation file, which defines a VHDL or Verilog HDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside your design. Include this file when compiling your design in the Quartus II software.
*<variation name>***.vo** or *<variation name>***.vho**	VHDL or Verilog HDL IP functional simulation model.

Notes to Table 2–1:

(1) These files are variation dependent, some may be absent or their names may change.

(2) *<variation name>* is a prefix variation name supplied automatically by IP Toolbench.

2. After you review the generation report, click **Exit** to close IP Toolbench and click **Yes** on the **Quartus II IP Files** message.

☞ The Quartus II IP File (**.qip**) is a file generated by the MegaWizard interface or SOPC Builder that contains information about a generated IP core. You are prompted to add this **.qip** file to the current Quartus II project at the time of file generation. In most cases, the **.qip** file contains all of the necessary assignments and information required to process the core or system in the Quartus II compiler. Generally, a single **.qip** file is generated for each MegaCore function and for each SOPC Builder system. However, some more complex SOPC Builder components generate a separate **.qip** file, so the system **.qip** file references the component **.qip** file.

You can now integrate your custom MegaCore function variation into your design and simulate and compile.

Simulate the Design

You can simulate your design using the IP Toolbench-generated VHDL and Verilog HDL IP functional simulation models.

✒ For more information on IP functional simulation models, refer to "Simulate the Design" on page 2–9 and the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the *Quartus II Handbook.*.

Altera also provides a fixed-example VHDL testbench that you can use to simulate an example UTOPIA level 2 system. It includes one UTOPIA level 2 master and two slaves (see Figure 2–10 on page 2–10). You can use the testbench as a basis for your own design. The testbench can be used with the IP functional simulation models. The testbench and associated files are located in the testbench directory. The testbench uses the following UTOPIA level 2 parameters:

■ UTOPIA bus width = 16

■ Master user cell size = 54

■ User cell size = 54

■ User bus width = 16

■ Generate parity

■ UTOPIA clock period = 20 ns

■ PHY clock period = 20 ns

■ MPHY mode

■ Number of slaves = 2

Figure 2–10. Testbench
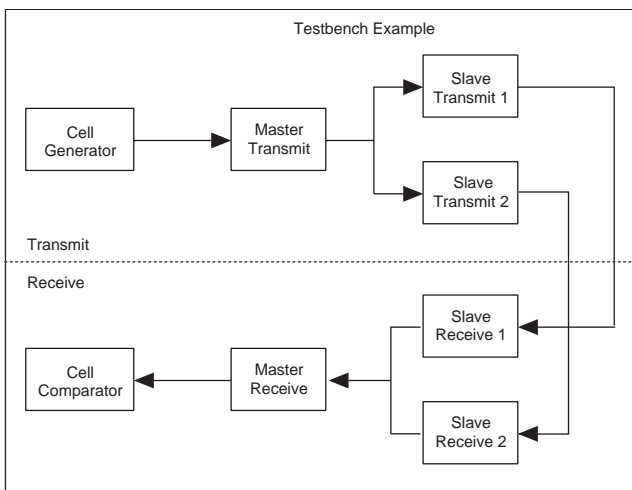
Testbench with the ModelSim Simulator

To use the example testbench with IP functional simulation models in the ModelSim simulator, follow these steps:

☞ The testbench includes pre-generated VHDL IP functional simulation models.

1. Start the ModelSim simulator.

2. Change the directory to the **sim_lib\modelsim** directory.

3. For VHDL, type the following command:

   ```
   do compile_utopia2_fixed_example_testbench_for_vhdl.tcl↵
   ```

☞ To simulate a Verilog HDL UTOPIA level 2 slave with the VHDL testbench, use a simulator with VHDL and Verilog HDL co-simulation support with the following command:

```
do compile_utopia2_fixed_example_testbench_for_verilog.tcl↵
```

Testbench with NativeLink

To use the testbench with NativeLink, follow these steps:

1. Using the New Project Wizard in the Quartus II software, create a new project in the **utopia2_slave\sim_lib\testbench**<*hdl*> directory, where <*hdl*> is **verilog** or **vhdl**, with the project name and top-level design entity **utopia2_example_top**.

2. Check that the absolute path to your third-party simulation tool is set. Set the path from **EDA Tool Options** in the **Options** dialog box (Tools menu).

3. Add the UTOPIA level 2 master and UTOPIA level 2 slave libraries:

   a. On the Assignments menu click **Settings**.

   b. Under **Category** click **Libraries**

   c. In **Project library name** click **...**

   d. Browse to **\utopia2_master\lib** and click **Open**.

   e. Click **Add**.

   f. In **Project library name** click **...**

   g. Browse to **\utopia2_slave\lib** and click **Open**.

   h. Click **Add**.

   i. Click **OK**.

4. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.

   ☞ If the analysis and elaboration is not successful, fix the error before moving to the next step.

5. On the Assignments menu, click **Settings**. The Settings window appears. Expand **EDA Tool Settings** and select **Simulation**.

6. In **Tool name**, select a simulator tool from the list.

   In EDA Netlist Writer options, select the required language (VHDL or Verilog) from the list for **Format for output netlist**.

   In NativeLink settings, select the **Compile test bench** option and then click the **Test Benches** button. The Test Benches window appears.

7. In the New Test Bench Settings window, enter the information described in Table 2–2 (see also Figure 2–11 on page 2–12). To enter the files described in the table, browse to the files in your project.

Table 2–2. NativeLink Test Bench Settings

Parameter	Setting / File Name
Test bench name	*<any name>*
Test bench entity	master_example_tb
Instance	utopia2_example_top_dut
Run for	100 ns
Test bench files *(1)*	**master_tb_pack.vhd**
	slave_user_if.vhd
	master_example_tb.vhd

Notes to Table 2–2:

(1) The files must be in the order shown, from the top to bottom, which is the order of compilation. Use the **Up** and **Down** buttons in the New Test Bench Settings window to order the files.

Figure 2–11 on page 2–12 shows the example testbench settings.

Figure 2–11. Example of New Test Bench Settings for NativeLink
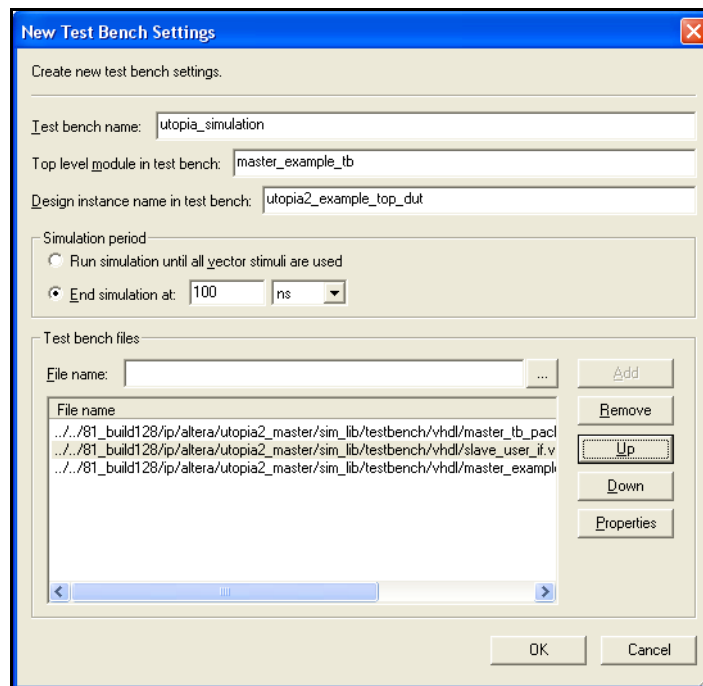
8. When you have entered the required information for your new testbench, click **OK** in the New Test Bench Settings window.

9. Click **OK** in the Test Benches window and then click **OK** in the Settings window.

10. On the Tools menu, point to **Run EDA Simulation Tool** and click **EDA RTL Simulation**. The simulation now begins with your chosen simulation tool.

Compile the Design

You can use the Quartus II software to compile your design. Refer to Quartus II Help for instructions on compiling your design.

Program a Device

After you have compiled your design, program your targeted Altera device and verify your design in hardware.

With Altera's free OpenCore Plus evaluation feature, you can evaluate the UTOPIA Level 2 Slave MegaCore function before you purchase a license. OpenCore Plus evaluation allows you to generate an IP functional simulation model, and produce a time-limited programming file.

For more information on IP functional simulation models, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the *Quartus II Handbook.*

You can simulate the UTOPIA Level 2 Slave MegaCore function in your design and perform a time-limited evaluation of your design in hardware.

For more information on OpenCore Plus hardware evaluation using the UTOPIA Level 2 Slave, see "OpenCore Plus Evaluation" on page 1–3, "OpenCore Plus Time-Out Behavior" on page 3–1, and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Set Up Licensing

You need to purchase a license for the MegaCore function only when you are completely satisfied with its functionality and performance and want to take your design to production.

After you purchase a license for UTOPIA Level 2 Slave MegaCore function, you can request a license file from the Altera website at www.altera.com/licensing and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have Internet access, contact your local Altera representative.

Figure 3–1 shows the UTOPIA MegaCore® function block diagram.

Figure 3–1. UTOPIA MegaCore Function Block Diagram
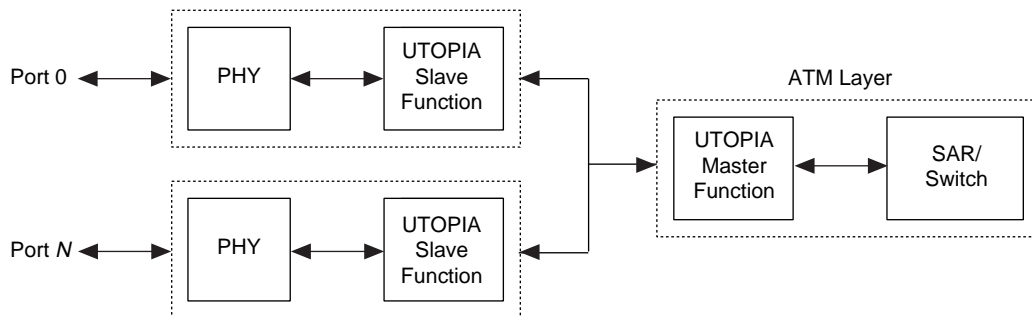
OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation can support the following two operation modes:

■ Untethered—the design runs for a limited time.

■ Tethered—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely.

All megafunctions in a device time out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior may be masked by the time-out behavior of the other megafunctions.

☞ For MegaCore functions, the untethered timeout is one hour; the tethered timeout value is indefinite.

Your design stops working after the hardware evaluation time expires and the following events occur:

■ For the receive interface:

    ■ The `phy_rx_valid` input goes low

    ■ The `phy_rx_enb` output goes low

    ■ The `phy_rx_valid` input goes low

    ■ The `rx_data` output goes low

- For the transmit interface:

    - The `phy_tx_enb` input goes low

    - The `phy_tx_soc` output goes low

    - The `phy_tx_valid` output goes low

    - The `phy_tx_clav` output goes low

    - The `phy_tx_data` output goes low

- For the Atlantic™ receive interface:

    - The `phy_rx_ena` input goes low

    - The `phy_rx_dav` output goes low

    - The `rx_data` output goes low

- For the Atlantic transmit interface:

    - The `phy_tx_ena` input goes low

    - The `phy_tx_dav` output goes low

    - 'The `phy_tx_val` output goes low

    - The `phy_tx_sop` output goes low

    - The `phy_tx_data` output goes low

For more information on OpenCore Plus hardware evaluation, see "OpenCore Plus Evaluation" on page 1–3 and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Parameters

Table 3–1 shows the UTOPIA Level 2 Slave MegaCore function parameters, which can only be set in IP Toolbench (see "Step 1: Parameterize" on page 2–4).

Table 3–1. UTOPIA Level 2 Slave Parameters (Part 1 of 2)

Parameter	Values	Type	Description
`sphy_mode`	Yes or no	Fixed or programmable	Determines whether operation is in UTOPIA 1 compatibility mode (that is, interfacing with a single physical layer (SPHY)).
`utopia_bus_width`	8 or 16	Fixed or programmable	Width of the UTOPIA data bus.
`user_bus_width`	8 or 16	Fixed or programmable	Width of the local data bus.
`52/53/54_byte_cells`	52, 53, or 54	Fixed or programmable	Determines whether operation is with cells of 52, 53 *(1)*, or 54 bytes on the local interface side. In 52-byte cells, the user defined (UDF) field has been removed.
`slave_address`	0 to 30	Fixed or programmable	Determines the address of the UTOPIA slave.
`parity_check`	Yes or no	Fixed or programmable	Determines whether UTOPIA bus parity is checked. This parameter controls the creation of parity logic in the transmit direction.

Table 3–1. UTOPIA Level 2 Slave Parameters (Part 2 of 2)

Parameter	Values	Type	Description
`parity_generate`	Yes or no	Fixed or programmable	Determines whether UTOPIA bus parity is generated. This parameter controls the creation of parity logic in the receive direction.
`discard_on_error`	Yes or no	Fixed or programmable	Determines whether cells with errors are discarded on reception. Cells that are received with parity errors on the UTOPIA transmit interface are discarded.
`pipeline_user_interface`	Yes or no	Fixed	Determines whether operation is in pipelined or non-pipelined mode. *(2)*

Notes to Table 3–1:

(1) 53-byte cells only supported in 8-bit local bus mode.

(2) Pipelined is the recommended mode; although the non-pipelined mode is also provided.

Signals

The MegaCore function uses the following signals:

■ Input—Standard input-only signal

■ Output—Standard output-only signal

■ Tri-state—Tri-state input/output signal

Figure 3–2 shows the MegaCore function's signal block diagram.

Figure 3–2. Signal Block Diagram
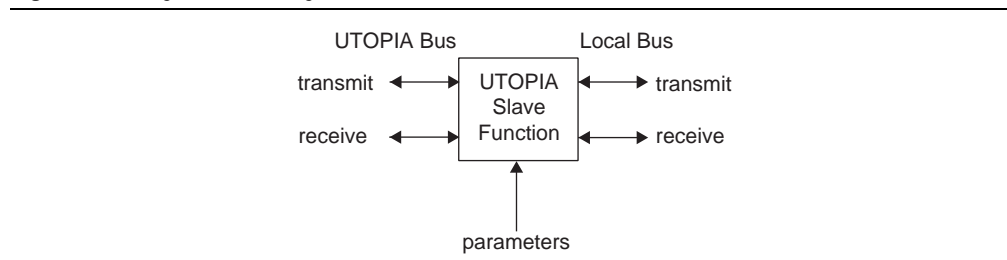
Table 3–2 shows the UTOPIA transmit interface signal definitions.

Table 3–2. UTOPIA Transmit Interface

Signal	Direction	Description
`tx_data[15/7:0]`	Input	Transmit data. When in 8-bit mode, `tx_data[7:0]` is used.
`tx_soc`	Input	Start of cell indicator. Active-high signal asserted when tx_data contains the first word of the cell.
`tx_enb`	Input	Enable. Active-low signal asserted when `tx_data` contains valid data.
`tx_clav`	Output	Cell available. Active-high signal asserted when the slave interface is ready to receive another cell.
`tx_clav_enb`	Output	Output enable signal for `tx_clav`. When high, the `tx_clav` output buffer should be enabled. When low, the output buffer should be tri-stated.
`tx_prty`	Input	Parity. The parity bit over `tx_data[7:0]` in 8-bit mode, and `tx_data[15:0]` in 16-bit mode (optional).

Table 3–2. UTOPIA Transmit Interface

Signal	Direction	Description
`tx_addr[4:0]`	Input	Address. Five-bit-wide address used in multi-PHY (MPHY) mode to poll and select the correct PHY device.
`tx_prty_pulse`	Output	Active-high 1 cycle pulse, indicating a parity error has been detected.
`tx_cell_pulse`	Output	Active-high 1 cycle pulse, indicating a cell has been received.
`tx_cell_err_pulse`	Output	Active-high 1 cycle pulse, indicating an illegal length cell has been received.
`tx_cell_disc_pulse`	Output	Active-high 1 cycle pulse, indicating a cell has been discarded due to the detection of a parity error or an incorrect length (short) cell.
`tx_clk`	Input	Transmit clock. All signals on this interface are synchronous to it.

Table 3–3 shows the local transmit interface signal definitions.

Table 3–3. Local Transmit Interface

Signal	Direction	Description
`phy_tx_data[15/7:0]`	Output	Data output for transmit data. When in 8-bit mode, `phy_tx_data[7:0]` is used.
`phy_tx_soc`	Output	Active-high signal asserted when `phy_tx_data` contains the first word of the cell.
`phy_tx_valid`	Output	Active-high signal asserted when valid cell data is present on `phy_tx_data`.
`phy_tx_enb`	Input	Active-high signal asserted when the local interface is ready to accept cell data.
`phy_tx_fifo_full`	Output	Active-high signal asserted when the receive first-in first-out (FIFO) buffer is full.
`phy_tx_clav`	Output	Active high signal asserted when there is at least one more complete cell in the FIFO buffer. Valid four cycles after the start of a cell.
`phy_tx_clk`	Input	Local transmit interface clock. All signals on this interface are synchronous to it.

Table 3–4 shows the Atlantic transmit interface signal definitions.

Table 3–4. Atlantic Transmit Interface (Part 1 of 2)

Signal	Direction	Description
`phy_tx_dat[15/7:0]`	Output	Data bus. This bus carries the packet octets that are transferred across the interface. The data is transmitted in big endian order on `phy_tx_dat`. The data is sent most significant bit (MSB) first and all valid bits are contiguous with the MSB.
`phy_tx_sop`	Output	Start of packet signal. `phy_tx_sop` is used to delineate the packet boundaries on the `phy_tx_dat` bus. When `phy_tx_sop` is high, the start of the packet is present on the `dat` bus. `phy_tx_sop` is required to be present at the beginning of every packet.

Table 3–4. Atlantic Transmit Interface (Part 2 of 2)

Signal	Direction	Description
`phy_tx_eop`	Output	End of packet signal. `phy_tx_eop` is used to delineate the packet boundaries on the `phy_tx_dat` bus. When `phy_tx_eop` is high, the end of the packet is present on the `phy_tx_dat` bus. `phy_tx_eop` is required to be present at the end of every packet.
`phy_tx_err`	Output	Error indicator signal. `phy_tx_err` is used to indicate that the current packet is aborted and should be discarded. `phy_tx_err` can be asserted at any time during the current packet, but when asserted it can only be de-asserted on the clock cycle after `phy_tx_eop` is asserted.
`phy_tx_val`	Output	Data valid signal. `phy_tx_val` indicates the validity of the data signals. `phy_tx_val` is updated on every clock edge where `phy_tx_ena` is sampled asserted, and holds its current value along with the `phy_tx_dat` bus where `ena` is sampled deasserted. When `phy_tx_val` is asserted, the Atlantic data interface signals are valid. When `phy_tx_val` is deasserted, the Atlantic data interface signals are invalid and must be disregarded. To determine whether new data has been received, the master must qualify the `phy_tx_val` signal with the previous state of the `phy_tx_ena` signal.
`phy_tx_ena`	Input	Enable signal. `phy_tx_ena` is driven by a master interface, and used to control the flow of data across the interface. `phy_tx_ena` behaves as a read enable from master to slave. When `phy_tx_ena` is sampled asserted, the Atlantic data interface signals contain new data during the following clock edge. `phy_tx_val` indicates the validity of the data. The Atlantic data interface signals get new data on every clock cycle, if `phy_tx_ena` is asserted.
`phy_tx_dav`	Output	Data available signal. If `phy_tx_dav` is high, the slave FIFO buffer has at least one cell available to be read, or the data can be read up to an end of packet without risk of underflow.

Table 3–5 shows the transmit configuration interface signal definitions.

Table 3–5. Transmit Configuration Interface (Part 1 of 2)

Signal	Direction	Description
`tx_phy_mode`	Input	0 = MPHY mode; 1 = SPHY mode.
`phy_tx_pipe_mode`	Input	0 = Non-pipelined mode; 1 = pipelined mode.
`tx_ut_width`	Input	1 = 16-bit UTOPIA transmit bus width. 0 = 8-bit UTOPIA transmit bus width.

Table 3–5. Transmit Configuration Interface (Part 2 of 2)

Signal	Direction	Description
`tx_user_width`	Input	1 = 16-bit local transmit bus width.
		0 = 8-bit local transmit bus width.
`tx_user_bytes[1:0]`	Input	Defines the size of the cells at the local transmit interface.
		8-bit local transmit bus: 00 = 52 bytes; 01 = 53 bytes; 1x = 54 bytes.
		16-bit local transmit bus: 0x = 52 bytes; 1x = 54 bytes.
`tx_address[4:0]`	Input	Transmit port address in MPHY mode.
`tx_discard_on_error`	Input	0 = Transmit cells not discarded on detecting a parity error.
		1 = Transmit cells discarded on detecting a parity error.
`tx_parity_check`	Input	Enables parity checking on the UTOPIA bus when high.
`reset`	Input	Active-low system reset. Can be asserted asynchronously, but must be deasserted synchronously to `tx_clk`.

Table 3–6 shows the UTOPIA receive interface signal definitions.

Table 3–6. UTOPIA Receive Interface

Signal	Direction	Description
`rx_data[15/7:0]`	Output	Receive data. When in 8-bit mode, `rx_data[7:0]` is used.
`rx_soc`	Output	Start of cell indicator. Active-high signal asserted when `rx_data` contains the first word of the cell.
`rx_prty`	Output	Parity. The parity bit over `rx_data[7:0]` in 8-bit mode, and `rx_data[15:0]` in 16-bit mode (optional).
`rx_bus_enb`	Output	Output enable signal for the `rx_data`, `rx_soc`, and `rx_prty` signals. When high, the output buffers should be enabled. When low, the output buffers should be tri-stated.
`rx_enb`	Input	Enable. Active-low signal asserted to enable the PHY device to drive data on `rx_data`.
`rx_clav`	Output	Cell available. Active-high signal asserted when another cell is ready to be sent.
`rx_clav_enb`	Output	Output enable signal for `rx_clav`. When high, the `rx_clav` output buffer should be enabled. When low, the output buffer should be tri-stated.
`rx_addr[4:0]`	Input	Address. Five-bit-wide address used in MPHY mode to poll and select the correct PHY device.
`rx_clk`	Input	Receive clock. All signals on this interface are synchronous to it.

Table 3–1 shows the local receive interface signal definitions.

Table 3–1. Local Receive Interface (Part 1 of 2)

Signal	Direction	Description
`phy_rx_data[15/7:0]`	Input	Data input for receive data. When in 8-bit mode, `phy_rx_data[7:0]` is used.
`phy_rx_soc`	Input	Active-high signal asserted when `phy_rx_data` contains the first word of the cell.
`phy_rx_valid`	Input	Active-high signal asserted when valid cell data is present on `phy_rx_data`.

Table 3–1. Local Receive Interface (Part 2 of 2)

Signal	Direction	Description
`phy_rx_enb`	Output	Active-high signal asserted when the local interface is ready to accept cell data. This signal goes low when the receive FIFO buffer is full.
`phy_rx_clav`	Output	Active high signal asserted when there is space for at least one more complete cell in the FIFO buffer. Valid four cycles after the start of a cell.
`phy_rx_clk`	Input	Local receive interface clock. All signals on this interface are synchronous to it.

Table 3–2 shows the Atlantic receive interface signal definitions.

Table 3–2. Atlantic Receive Interface

Signal	Direction	Description
`phy_rx_dat[15/7:0]`	Input	Data bus. This bus carries the packet octets that are transferred across the interface. The data is transmitted in big endian order on `phy_rx_dat`. The data is sent most significant bit (MSB) first and all valid bits are contiguous with the MSB.
`phy_rx_sop`	Input	Start of packet signal. `phy_rx_sop` is used to delineate the packet boundaries on the `phy_rx_dat` bus. When `phy_rx_sop` is high, the start of the packet is present on the `phy_rx_dat` bus. `phy_rx_sop` is required to be present at the beginning of every packet.
`phy_rx_eop`	Input	End of packet signal. `phy_rx_eop` is used to delineate the packet boundaries on the `phy_rx_dat` bus. When `phy_rx_eop` is high, the end of the packet is present on the `phy_rx_dat` bus. `phy_rx_eop` is required to be present at the end of every packet.
`phy_rx_err`	Input	Error indicator signal. `phy_rx_err` is used to indicate that the current packet is aborted and should be discarded. `phy_rx_err` can be asserted at any time during the current packet, but when asserted it can only be de-asserted on the clock cycle after `phy_rx_eop` is asserted. Conditions that can cause `phy_rx_err` to be set can be, but are not limited to, FIFO overflow and abort sequence detection.
`phy_rx_ena`	Input	Enable signal. `phy_rx_ena` is driven by a master interface, and used to control the flow of data across the interface. `phy_rx_ena` behaves as a write enable from master to slave. When `ena` is sampled asserted, the Atlantic data interface signals are valid and are transferred across the interface on the following rising edge of `clk`. The Atlantic data interface signals get new data on every clock cycle, if `ena` is asserted.
`phy_rx_dav`	Output	Data available signal. If `phy_rx_dav` is high, the slave FIFO has enough space for another one cell to be written.

Table 3–3 shows the receive configuration interface signal definitions.

Table 3–3. Receive Configuration Interface

Signal	Direction	Description
`rx_parity_generate`	Input	Enables parity generation on the UTOPIA bus when high.
`rx_phy_mode`	Input	0 = MPHY mode; 1 = SPHY mode.
`phy_rx_pipe_mode`	Input	0 = Non-pipelined mode; 1 = pipelined mode.
`rx_ut_width`	Input	1 = 16-bit UTOPIA receive bus width. 0 = 8-bit UTOPIA receive bus width.
`rx_user_width`	Input	1 = 16-bit local receive bus width. 0 = 8-bit local receive bus width.
`rx_user_bytes[1:0]`	Input	Defines the size of the cells at the local receive interface. 8-bit local receive bus: 00 = 52 bytes; 01 = 53 bytes; 1x = 54 bytes. 16-bit local receive bus: 0x = 52 bytes; 1x = 54 bytes.
`rx_address[4:0]`	Input	Receive port address in MPHY mode.
`reset`	Input	Active-low system reset. Can be asserted asynchronously, but must be deasserted synchronously to `rx_clk`.

Interfaces

This section describes the following interfaces:

■ UTOPIA transmit

■ Local transmit

■ UTOPIA receive

■ Local receive

■ Atlantic Interface

UTOPIA Transmit Interface

The UTOPIA transmit interface receives ATM cell data from a UTOPIA level 2 master transmitter. The interface supports 8- or 16-bit transmit data buses, and SPHY or MPHY modes of operation. Any cells that are too short are discarded, and any cells that are too long have their excess bytes discarded. If `discard_on_error` is enabled, any parity errors detected on `tx_data` causes the cell to be discarded. The interface contains several statistic 'pulse' outputs, which can be used to count parity errors, cells received, illegal length cells, and discarded cells.

In MPHY mode, a UTOPIA master polls the various UTOPIA slaves using `tx_addr`. The slaves respond by driving their `tx_clav` output to allow the master to determine whether they can accept a cell transfer. The master then selects a slave and transfers a complete cell. This behavior is described in section 4.2 of the UTOPIA Level 2, Version 1.0 specification.

The UTOPIA Level 2, Version 1.0 specification is available from **w**www.atmforum.com

In SPHY mode, the slave indicates that it can accept a cell transfer by asserting `tx_clav`. The master subsequently transmits a cell to the slave by asserting `tx_enb` low. This behavior is described in section 3 of the *UTOPIA Level 2, Version 1.0* specification.

The received ATM cell data is written into the rate-matching four-cell deep transmit FIFO buffer ready to be accessed by the local transmit interface.

Local Transmit Interface

The local transmit interface provides access to read the received ATM cell data from the transmit FIFO buffer in a variety of different cell formats. The data path (`phy_tx_data`) can operate either in 8- or 16-bit mode. The cell size can be 52, 53, or 54 bytes long. The local transmit interface indicates that cell data is available on `phy_tx_data` by asserting `phy_tx_valid` high. `phy_tx_soc` is also asserted high with the first data word of the cell.

The local side controls the transfer of data across this interface by asserting `phy_tx_enb` high when it is ready to accept data. In non-pipelined mode, data is transferred when both `phy_tx_enb` and `phy_tx_valid` are high. In pipelined mode, data is transferred when `phy_tx_valid` is high and the previous value of `phy_tx_enb` is high.

If further cells remain in the transmit FIFO buffer, back-to-back cells may be transferred by keeping `phy_tx_enb` high (see Figure 3–1).

Figure 3–1. Local Transmit Interface Timing Diagram
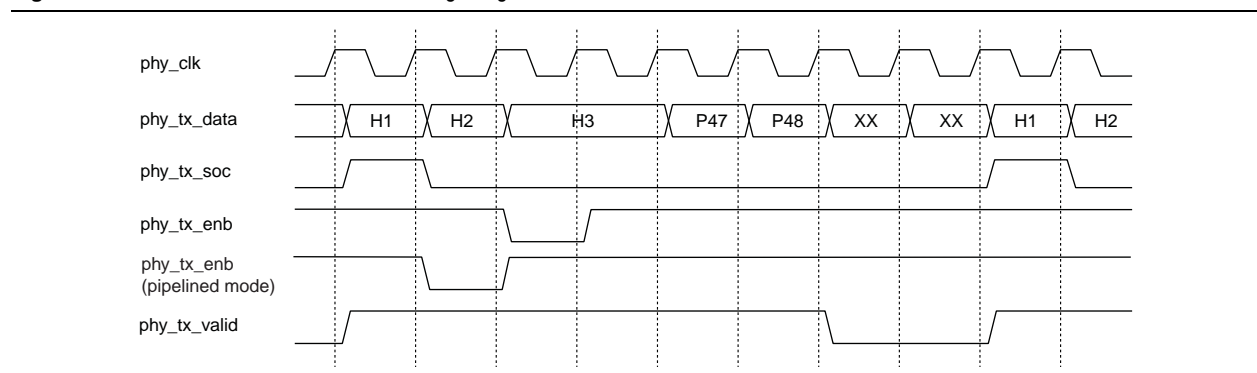
UTOPIA Receive Interface

The UTOPIA receive interface transmits ATM cell data to a UTOPIA level 2 master receiver. The interface supports 8- or 16-bit receive data buses, and SPHY or MPHY modes of operation.

In MPHY mode, a UTOPIA level 2 master polls the various UTOPIA level 2 slaves using `rx_addr`. The slaves respond by driving their `rx_clav` output, to allow the master to determine whether they have any cells ready for transfer. The master then selects a slave and transfers a complete cell. This behavior is described in section 4.2 of the *UTOPIA Level 2, Version 1.0 specification*.

In SPHY mode, the slave indicates that it has a cell ready for transfer by asserting `rx_clav`. The master subsequently initiates the transfer of a cell from the slave by asserting `rx_enb` low. This behavior is described in section 3 of the *UTOPIA Level 2, Version 1.0 specification*.

Local Receive Interface

The local receive interface provides access to fill the receive FIFO buffer with ATM cell data that is ready to be transmitted on the UTOPIA bus. The data path (`phy_rx_data`) can operate either in 8- or 16-bit mode. The cell size can be 52, 53, or 54 bytes long. The local receive interface is notified that cell data is available on `phy_rx_data` by asserting `phy_rx_valid` high. `phy_rx_soc` should also be asserted high with the first data word of the cell.

The local receive interface controls the transfer of data across this interface by asserting `phy_rx_enb` high when it is ready to accept data. In non-pipelined mode, data is transferred when both `phy_rx_enb` and `phy_rx_valid` are high. In pipelined mode, data is transferred when `phy_rx_valid` is high and the previous value of `phy_rx_enb` is high.

If further space for cells remains in the receive FIFO buffer (i.e., `phy_rx_enb` is high), back-to-back cells may be transferred by keeping `phy_rx_valid` high and supplying continuous cell data (see Figure 3–2).

Figure 3–2. Local Receive Interface Timing Diagram
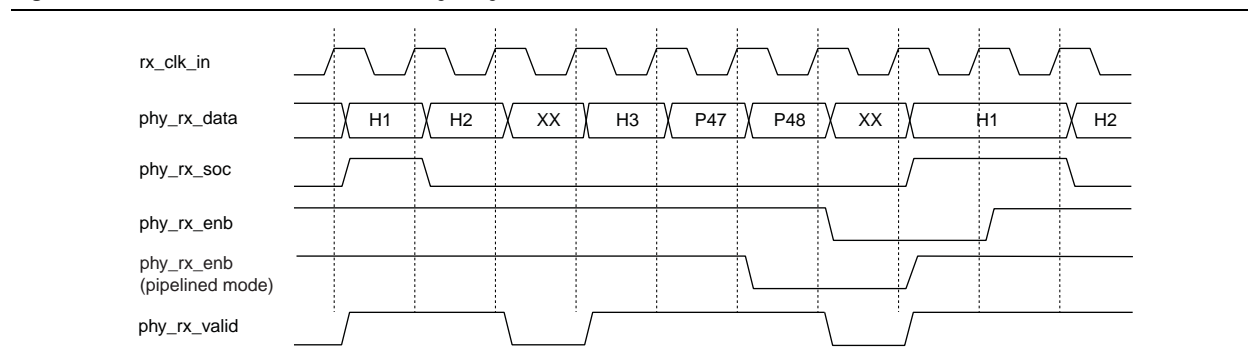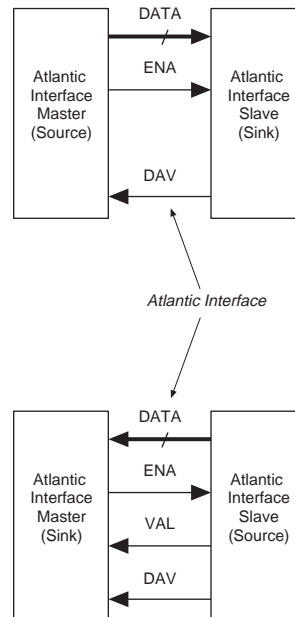

Atlantic Interface

On the UTOPIA Level 2 MegaCore function, the Atlantic interface is configured to be a slave control interface.

- A slave sink interface responds to write commands from a master source interface and behaves like a synchronous FIFO controller.

- A master sink interface generates read commands to a slave source interface and behaves like a synchronous FIFO controller.

Figure 3–3 shows the following four Atlantic interface control options:

- Master source
- Slave sink
- Master sink
- Slave source

**Figure 3–3.** Atlantic Interface Control Options

Compatibility

To ensure that individual implementations of an Atlantic interface are compatible they must have the following:

■ The same data bus width

■ Compatible data directions (data source connecting to data sink)

■ Compatible control interfaces (master interface connecting to slave interface)

■ Compatible FIFO threshold levels (slave sink can overflow, and slave source can operate inefficiently if thresholds are incorrectly set)

Timing

Figure 3–4 shows the timing of the Atlantic interface when in master mode. Figure 3–5 shows the timing of the Atlantic interface when in slave mode.

Figure 3–4. Atlantic Interface Timing—Slave Source
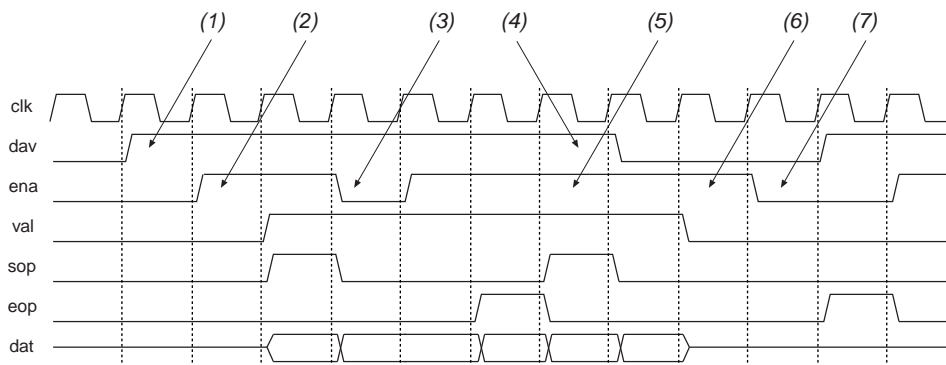
Notes to Figure 3–4:

(1) Slave (source) indicates that data is available (at least one word).

(2) Master (sink) begins reading data.

(3) Master (sink) decides to stop reading the data for one clock cycle. val remains asserted.

(4) Slave (source) indicates that it has less than one word available. The master (sink) cannot tell which, nor can it deassert ENA quickly enough to prevent a potential underflow, but this is desirable if it is also to be able to transfer back to back packets.

(5) Master (sink) continues to read data, validates data with val.

(6) Slave (source) cannot supply any more data, so deasserts val.

(7) Master (sink) goes idle until dav is reasserted.

Figure 3–5. Atlantic Interface Timing—Slave Sink
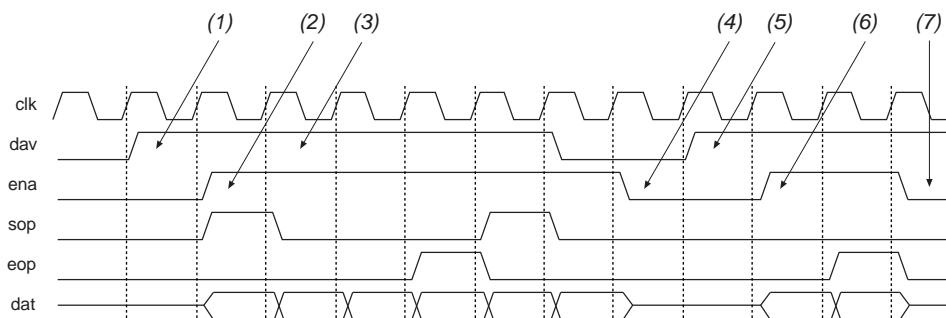
Notes to Figure 3–5:

(1) Slave (sink) indicates it has space for at least two words.

(2) Master (source) begins writing data to the slave (sink).

(3) Slave (sink) indicates it does not have space for two words. Master (source) must stop sending data, on the next clock cycle to ensure that the slave (sink) FIFO does not overflow. By setting the slave (sink) threshold to a suitable value, the requirement on the master (source) to stop immediately is removed. Setting a small threshold can be used to counter for pipeline delays; setting a large threshold may be useful if the master (source) is capable of transferring data in bursts.

(4) Master (source) stops sending data.

(5) Slave (sink) indicates it has space for at least two words.

(6) Master (source) begins writing data to the slave (sink).

(7) Slave (sink) indicates it still has space, but the master (source) has run out of data.

MegaCore Function Verification

The MegaCore function verification includes an automated regression test suite that tests all possible interfaces and sets of parameters (but not all combinations). The transmitter and receiver are tested separately and have random data and random packet sizes sent through them, including cells with parity errors and short and long cells.

Hardware verification on the UTOPIA master and slave MegaCore functions was performed using a Cobalt UTOPIA emulator on a custom UTOPIA test board. The Cobalt UTOPIA emulator verifies all the supported UTOPIA bus modes against an independent source and monitors the response to various error conditions. The hardware verification generates and monitors a UTOPIA data stream from both master to slave, and slave to master.

The hardware verification generates and monitors the following cell error conditions and displays any error count and errors including cell number and byte number:

- Long cells
- Short cells
- Parity errors

Revision History

The following table shows the revision history for this user guide.

Date	Version	Changes Made
March 2009	9.0	No changes.
November 2008	8.1	No changes.
May 2008	8.0	Added support for Stratix® IV devices.
October 2007	7.2	No changes.
May 2007	7.1	Updated the device family support.
December 2006	7.0	Updated the release information and device family support.
December 2006	6.1	■ Updated the release information and device family support ■ Updated the performance tables ■ Added a procedure for running a testbench simulation using NativeLink

How to Contact Altera

For the most up-to-date information about Altera® products, see the following table.

Contact *(Note 1)*	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Altera literature services	Email	literature@altera.com
Non-technical support (General)	Email	nacomp@altera.com
(Software Licensing)	Email	authorization@altera.com

Note:

(1)   You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The following table shows the typographic conventions that this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicates command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, and software utility names. For example, **\qdesigns** directory, **d:** drive, and **chiptrip.gdf** file.

Visual Cue	Meaning
Italic Type with Initial Capital Letters	Indicates document titles. For example: *AN 519: Stratix IV Design Guidelines.*
Italic type	Indicates variables. For example, *n* + 1.
	Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>*.**pof** file.
Initial Capital Letters	Indicates keyboard keys and menu names. For example, Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. Active-low signals are denoted by suffix `n`. Example: `resetn`.
	Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.
	Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
1., 2., 3., and a., b., c., and so on.	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■	Bullets indicate a list of items when the sequence of the items is not important.
☞	The hand points to information that requires special attention.
⚠ CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
⚠ WARNING	A warning calls attention to a condition or possible situation that can cause you injury.
↵	The angled arrow instructs you to press the enter key.
👣	The feet direct you to more information about a particular topic.