



UTOPIA Level 2 Master MegaCore

Function User Guide



101 Innovation Drive
San Jose, CA 95134
www.altera.com

MegaCore Version: 9.0
Document Date: March 2009

Copyright © 2009 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

Chapter 1. About This MegaCore Function

New Features	1-1
Release Information	1-1
Device Family Support	1-1
Features	1-2
General Description	1-2
Atlantic Interface	1-3
OpenCore Plus Evaluation	1-3
Performance and Resource Utilization	1-4

Chapter 2. Getting Started

Design Flow	2-1
UTOPIA Level 2 Master Walkthrough	2-2
Create a New Quartus II Project	2-2
Launch IP Toolbench	2-3
Step 1: Parameterize	2-5
Step 2: Set Up Simulation	2-7
Step 3: Generate	2-8
Simulate the Design	2-10
Testbench with the ModelSim Simulator	2-11
Testbench with NativeLink	2-12
Compile the Design	2-13
Program a Device	2-13
Set Up Licensing	2-14

Chapter 3. Functional Description

OpenCore Plus Time-Out Behavior	3-1
Parameters	3-2
Signals	3-3
Interfaces	3-8
UTOPIA Transmit Interface	3-8
Local Transmit Interface	3-9
Atlantic Slave Transmit Interface	3-10
UTOPIA Receive Interface	3-11
Local Receive Interface	3-11
Atlantic Slave Receive Interface	3-12
Atlantic Interface	3-13
Compatibility	3-14
Timing	3-14
Address Translation	3-15
MegaCore Verification	3-16

Additional Information


Revision History	1-1
How to Contact Altera	1-1
Typographic Conventions	1-1

Release Information

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

Table 1–1. UTOPIA Level 2 Master Release Information

Item	Description
Version	9.0
Release Date	March 2009
Ordering Code	IP-UTOPIA2MS
Product ID	0017
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 Master 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
- Single-PHY (SPHY) operation, with both octet- and cell-level handshaking
- MPHY operation, with single clav signal
- Logical-to-physical address translation via programmable look-up table
- Parity generation and detection
- 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 Master MegaCore function is designed for use in asynchronous transfer mode (ATM) layer devices that transfer data to and from physical layer (PHY) devices using the standard UTOPIA bus.

The UTOPIA Level 2 Master 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 with up to 31 PHY devices.

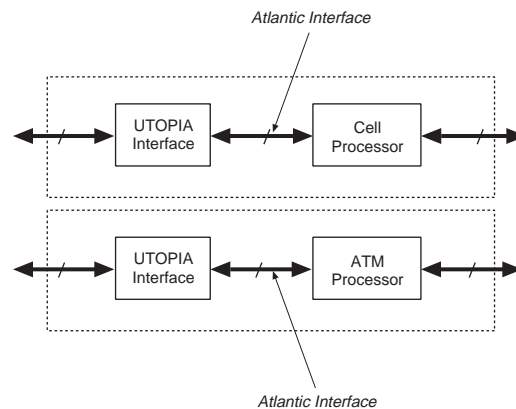
The transmitter polls the PHYs in a round-robin fashion to determine which are ready to receive data transfers and outputs the poll status to the ATM layer. The transmitter accepts cells from the ATM layer and sends them to the PHY devices via the UTOPIA bus interface. There is an option to generate parity information for the UTOPIA bus.

The receiver polls the PHYs in a round-robin fashion to determine which are ready to send data transfers and outputs the poll status to the ATM layer. The receiver accepts cells from the PHY devices via the UTOPIA bus interface and sends them to the ATM layer. There is an option to check for parity errors on 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. The MegaCore function supports a slave Atlantic interface only.

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 Master, 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 8.1:

- Cyclone II

■ Stratix III

The UTOPIA level 2 master runs above 50 MHz in all Cyclone, Cyclone II, Cyclone III, Stratix, Stratix II, and Stratix III devices. All figures are MPHY and use the complete address range and RAM for the translation tables.



The performance for Stratix IV devices is similar to Stratix III devices.



Use the LogicLock™ incremental design capability for greater performance.

Table 1-3. Performance—Cyclone II Device

Parameters				LEs	Memory Blocks
Data Flow Direction	UTOPIA Width (local bits)	Local Cell Size	Mode		
Receiver	8	53	MPHY using parity_generate and pipeline_user_interface	270	1
	16	54	MPHY using parity_generate	287	1
Transmitter	8	53	MPHY using parity_generate and pipeline_user_interface	269	1
	16	54	MPHY using parity_generate	294	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	53	MPHY using parity_generate and pipeline_user_interface	187	1
	16	54	MPHY using parity_generate	196	1
Transmitter	8	53	MPHY using parity_generate and pipeline_user_interface	180	1
	16	54	MPHY using parity_generate	191	1

Design Flow

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

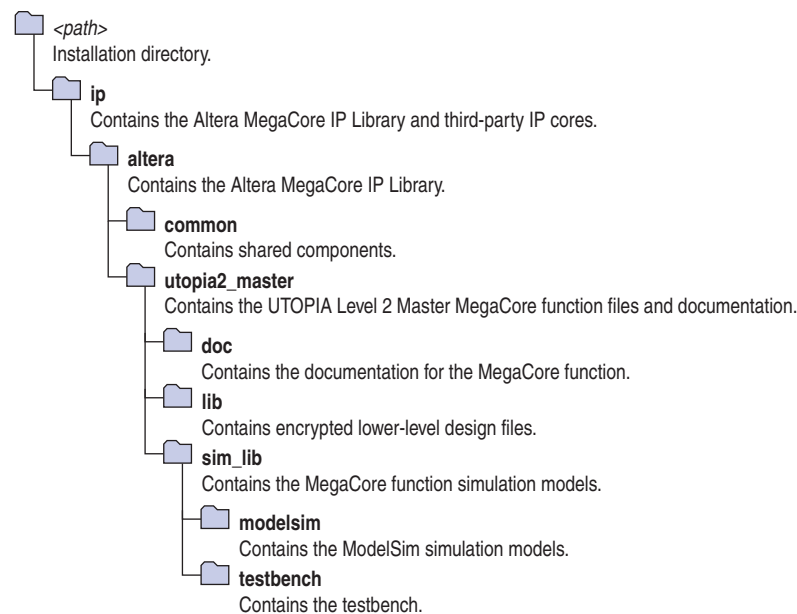
1. Obtain and install the UTOPIA Level 2 Master MegaCore function.

The UTOPIA Level 2 Master MegaCore function 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 and Linux Workstations*.

Figure 2–1 on page 2–1 shows the directory structure after you install the UTOPIA Level 2 Master 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



2. Create a custom variation of the UTOPIA Level 2 Master 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 Master MegaCore function.

After you have purchased a license for the UTOPIA Level 2 Master 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 Master Walkthrough

This walkthrough explains how to create a UTOPIA Level 2 Master MegaCore function using the Altera UTOPIA Level 2 Master IP Toolbench and the Quartus II software. When you finish generating a custom variation of the UTOPIA Level 2 Master 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\utopia2m_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 (see [Figure 2-2](#)).


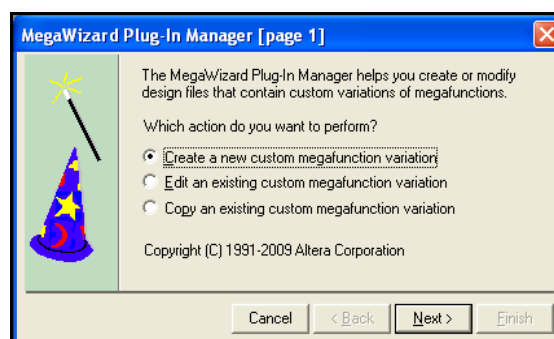
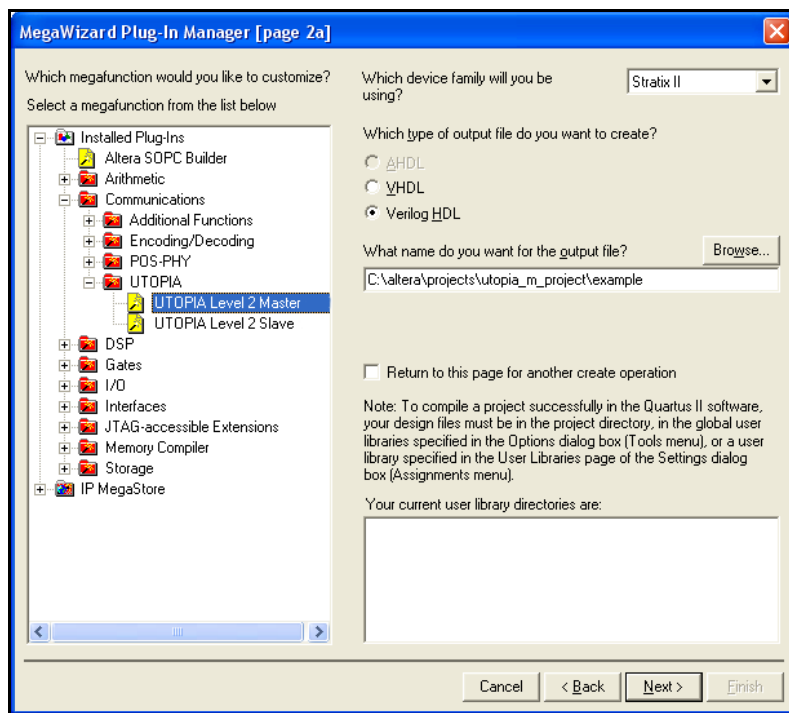
 Refer to Quartus II Help for more information on how to use the MegaWizard Plug-In Manager.

Figure 2-2. 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 Master**.
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 2-3](#) shows the wizard after you have made these settings.

Figure 2-3. Select the MegaCore Function



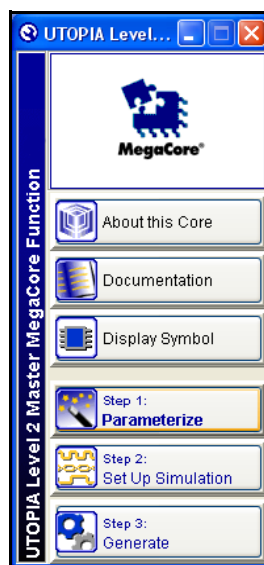
6. Click **Next** to launch IP Toolbench.

Step 1: Parameterize

To parameterize your MegaCore function, follow these steps:

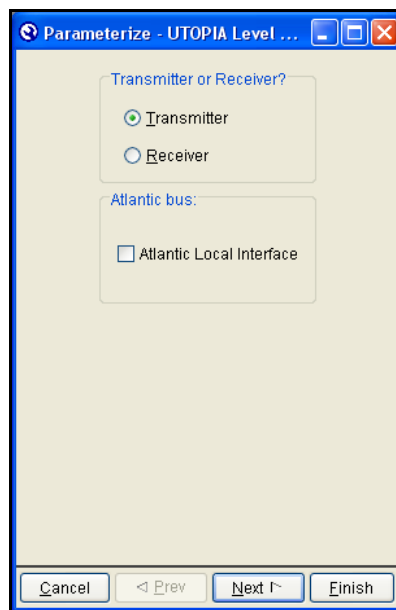
1. Click **Step 1: Parameterize** in IP Toolbench (see [Figure 2-4](#) on page 2-5).

Figure 2-4. IP Toolbench—Parameterize

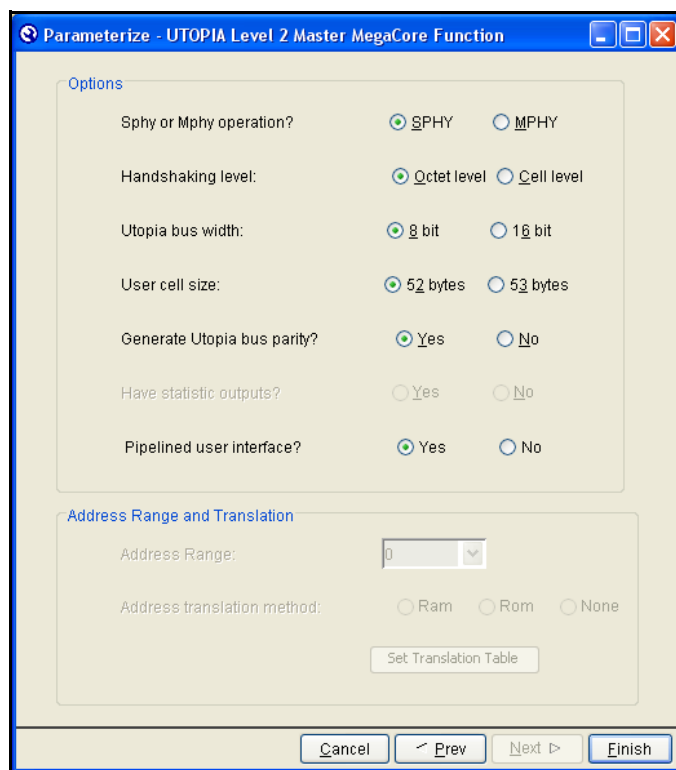


2. Select whether you wish to create a transmitter or receiver (see [Figure 2-5](#)). If you require an Atlantic™ local interface, turn on **Atlantic Local Interface**. Click **Next**.

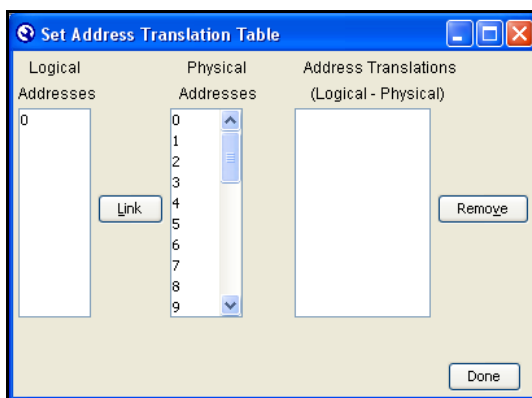
Figure 2-5. Select a Transmitter or Receiver



3. Choose the parameters that define the specific UTOPIA master MegaCore function you wish to implement (see [Figure 2-6](#)). 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.

Figure 2-6. Select the Parameters

4. When you select the multi-physical layer (MPHY) mode, you can implement an address translation table in ROM or RAM.
 - a. To implement the address translation table in ROM, click **Set Address Translation Table**. The Set Address Translation Table dialog box appears.
 - b. Click on a logical address and a physical address and then click **Link** (see [Figure 2-7 on page 2-6](#)). Continue until all logical addresses have been linked to the required physical addresses. Click **Done**.

Figure 2-7. Setting the Address Translation Tables

5. Click **Finish** in the Parameterize window when you are done.

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.

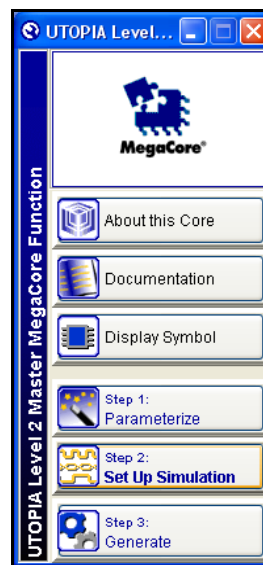


You may only use these simulation model output files for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis will create a nonfunctional design.

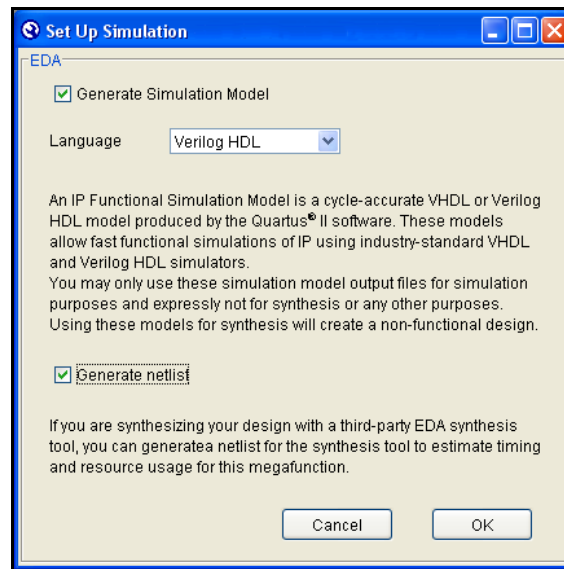
To generate an IP functional simulation model for your MegaCore function, follow these steps:

1. Click **Step 2: Set Up Simulation** in IP Toolbench (see [Figure 2-8 on page 2-7](#)).

Figure 2-8. IP Toolbench—Set Up Simulation



2. Turn on **Generate Simulation Model** (see [Figure 2-9](#)).

Figure 2–9. 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 2–10](#)).

Figure 2-10. IP Toolbench—Generate

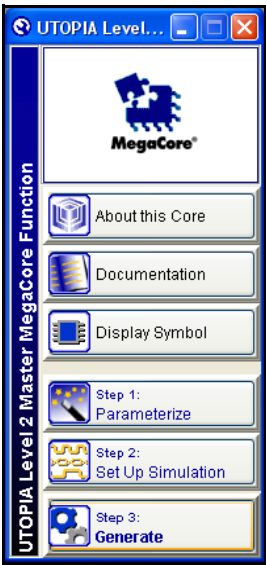


Figure 2-11 on page 2-9 shows the generation report.

Figure 2-11. Generation Report

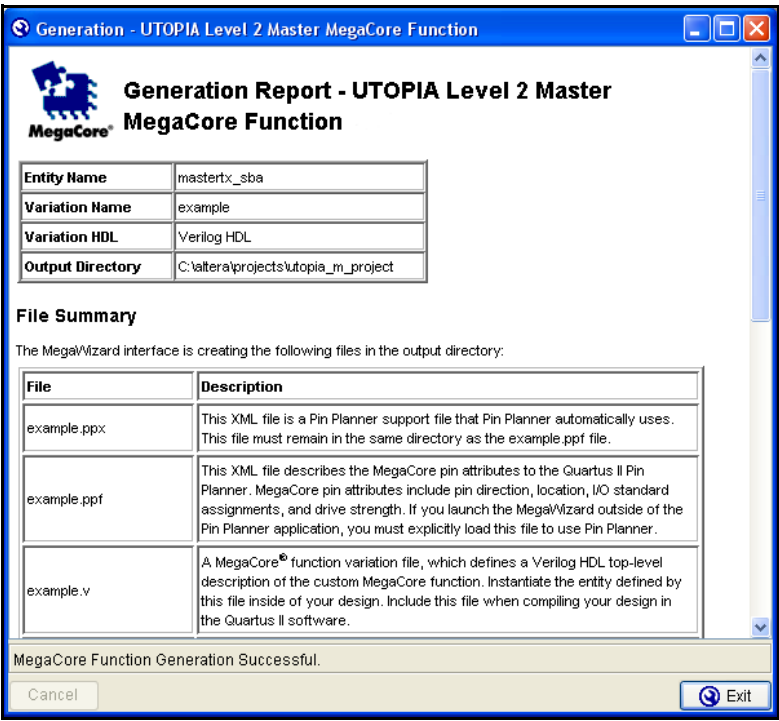


Table 2-1 describes the generated files and other files that may be in your project directory. The names and types of files specified in the IP Toolbench report vary based on whether you created your design with VHDL or Verilog HDL.

Table 2-1. Generated Files *(Note 1)*

Extension <i>(2)</i>	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> .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.

1. 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.

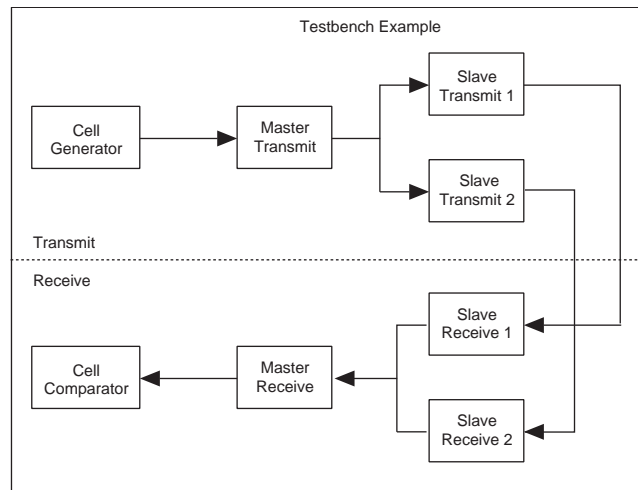


For more information on IP functional simulation models, refer to “*Simulate the Design*” on page 2-10 and the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the *Quartus II Handbook*.

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 2-12 on page 2-11](#)). 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 2-12. 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 master 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_master\sim_lib\testbench\<hdl>** directory, where **<hdl>** is **verilog** or **vhdl**, with the project name and top-level entity name of **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 2-13 on page 2-13](#)). To enter the files described in the table, browse to the files in your project.

Table 2-2. NativeLink Test Bench Settings

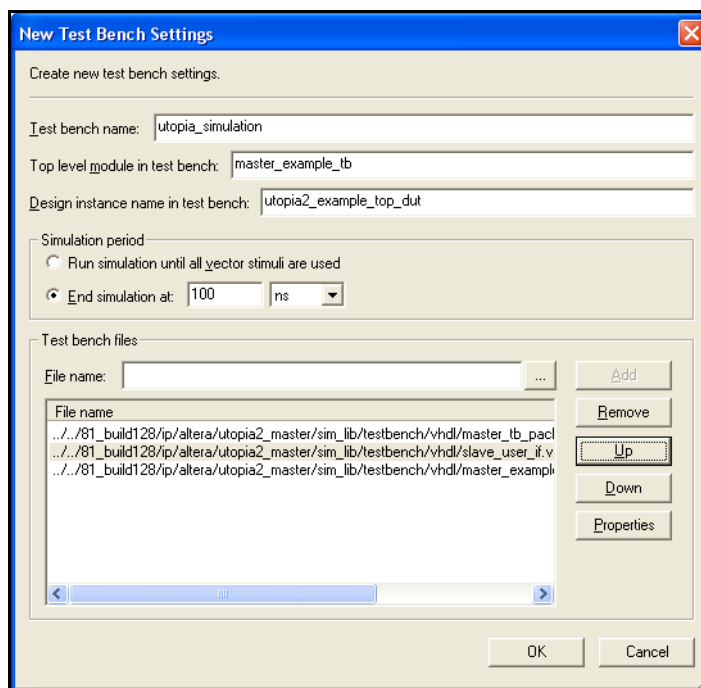
Parameter	Setting / File Name
Test bench name	<any name>
Test bench entity	master_example_tb
Instance	utopia2_example_top_dut
Run for	100 ns
Test bench files (1)	master_tb_pack.vhd
	slave_user_if.vhd
	master_example_tb.vhd

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 correctly order the files.

Figure 2-13 on page 2-13 shows the example testbench settings.

Figure 2-13. 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 Master 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 Master 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 Master, 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*.

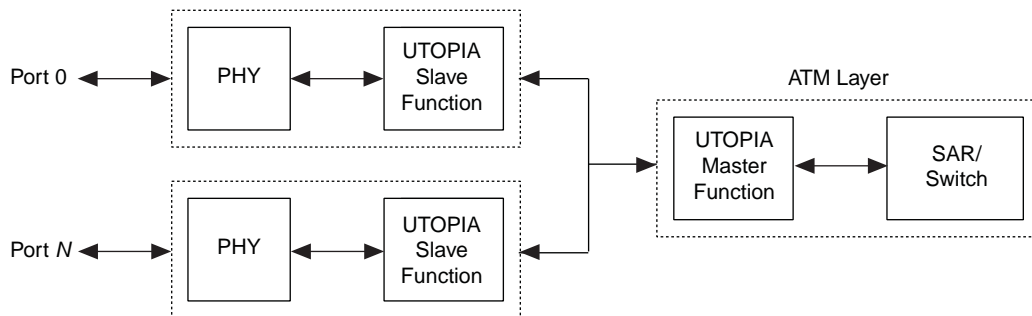
Set Up Licensing

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 Master 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 atm_rx_port_load input goes low
 - The atm_rx_data output goes low
 - The atm_rx_valid output goes low
 - The rx_clav input goes low
 - The rx_enb output goes high

- For the transmit interface:
 - The atm_tx_port_load input goes low
 - The atm_tx_enb output goes low
 - The atm_tx_valid input goes low
 - The tx_clav input goes low
 - The tx_enb output goes high
 - The tx_data output goes low
- For the Atlantic™ receive interface:
 - The atm_rx_req_adr_load input goes low
 - The atm_rx_data output goes low
 - The atm_rx_val output goes low
 - The rx_clav input goes low
 - The rx_enb output goes high
- For the Atlantic transmit interface:
 - The atm_tx_dav output goes low
 - The atm_tx_ena input goes low
 - The tx_clav input goes low
 - The tx_enb output goes high
 - The 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 Master 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 Master Parameters (Part 1 of 2)

Parameter	Values	Description
sphy_mode	Yes or no	Determines whether operation is in UTOPIA 1 compatibility mode (that is, interfacing with a single physical layer (SPHY)).
octet_handshake	Yes or no	When in SPHY mode, this parameter determines whether the interface supports octet- or cell-level handshaking.
bus_width	8 or 16	Width of the UTOPIA and local data bus. The local data bus width is always the same as the UTOPIA bus width.
53/54_byte_cells	Yes or no	Determines whether operation is with cells of 53 bytes (54 bytes in 16-bit mode), or cells of 52 bytes on the local interface side. In 52-byte cells the user defined (UDF) field has been removed.
parity_check	Yes or no	Determines whether UTOPIA bus parity is checked. This parameter controls the creation of parity logic in the receive direction.

Table 3-1. UTOPIA Level 2 Master Parameters (Part 2 of 2)

Parameter	Values	Description
parity_generate	Yes or no	Determines whether UTOPIA bus parity is generated. This parameter controls the creation of parity logic in the transmit direction.
pipeline_user_interface	Yes or no	Determines whether operation is in pipelined or non-pipelined mode. (1)
address_range	0 to 30	Determines the range of logical addresses that is polled.
address_translation	Yes or no	Determines whether logical-to-physical address translation is performed. If address translation is not enabled, logical addresses are mapped directly to physical addresses.

Note to Table 3-1:

(1) 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

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

Figure 3-2. Signal Block Diagram

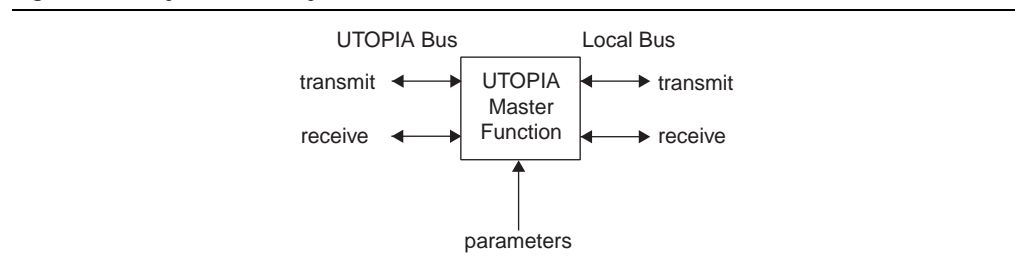


Table 3-2 shows the UTOPIA transmit interface signal definitions.

Table 3-2. UTOPIA Transmit Interface

Signal	Direction	Description
tx_data[15/7:0]	Output	Transmit data. When in 8-bit mode, tx_data[7:0] is used.
tx_soc	Output	Start of cell indicator. Active-high signal asserted when tx_data contains the first word of the cell.
tx_enb	Output	Enable. Active-low signal asserted when tx_data contains valid data.
tx_clav	Input	Cell available. Active-high signal asserted from the PHY devices when they are ready to receive another cell.
tx_prty	Output	Parity. tx_prty is the parity bit over tx_data[7:0] in 8-bit mode, and tx_data[15:0] in 16-bit mode.
tx_addr[4:0]	Output	Address. Five-bit-wide address used in multi-PHY (MPHY) mode to poll and select the various connected PHY devices.
tx_clk_in	Input	Transmit clock. All signals on the UTOPIA and local transmit interface are synchronous to it.

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

Table 3-3. Local Transmit Interface

Signal	Direction	Description
atm_tx_data[15/7:0]	Input	Data input for transmit data. When in 8-bit mode, atm_tx_data[7:0] is used.
atm_tx_soc	Input	Active-high signal asserted when atm_tx_data contains the first word of the cell.
atm_tx_valid	Input	Active-high signal asserted when valid cell data is present on atm_tx_data.
atm_tx_enb	Output	Active-high signal asserted when ready to accept cell data.
atm_tx_port_stat[30:0]	Output	Port status of polled PHYs. Represents the tx_clav status of the corresponding PHYs.
atm_tx_port[4:0]	Input	Port address for next cell transfer.
atm_tx_port_load	Input	Active-high enable pulse. Loads the port address for the next cell transfer.
atm_tx_port_wait	Output	Active-high signal. Indicates that the local interface must wait before loading the next port address with atm_tx_port.

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

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

Signal	Direction	Description
atm_tx_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 atm_tx_dat. The data is sent most significant bit (MSB) first and all valid bits are contiguous with the MSB.
atm_tx_sop	Input	Start of packet signal. atm_tx_sop is used to delineate the packet boundaries on the atm_tx_dat bus. When atm_tx_sop is high, the start of the packet is present on the atm_tx_dat bus. atm_tx_sop is required to be present at the beginning of every packet.
atm_tx_eop	Input	End of packet signal. atm_tx_eop is used to delineate the packet boundaries on the atm_tx_dat bus. When atm_tx_eop is high, the end of the packet is present on the atm_tx_dat bus. atm_tx_eop is required to be present at the end of every packet.
atm_tx_ena	Input	Enable signal. atm_tx_ena is driven by a master interface, and used to control the flow of data across the interface. atm_tx_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.

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

Signal	Direction	Description
atm_tx_dav	Output	Data available signal. If atm_tx_dav is high, the first-in first-out (FIFO) buffer has enough space for another word to be written. If more than one word of data is written after atm_tx_dav transitions low, data is lost.
atm_tx_adr_stat[30:0]	Output	Port status of polled PHYs. Represents the tx_clav status of the corresponding PHYs.
atm_tx_adr[4:0]	Input	Port address for next cell transfer.

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

Table 3-5. Transmit Configuration Interface

Signal	Direction	Description
tx_phy_mode[1:0]	Input	Bit 0: 0 = MPHY mode; 1 = SPHY mode. Bit 1: (SPHY mode only) 0 = octet-level handshaking, 1 = cell-level handshaking.
atm_tx_pipe_mode	Input	0 = Non-pipelined mode; 1 = pipelined mode.
tx_width	Input	0 = 8-bit UTOPIA transmit and local bus width. 1 = 16-bit UTOPIA transmit and local bus width.
tx_user_bytes	Input	Defines the size of the cells at the local transmit interface. 8-bit local transmit bus: 0 = 52 bytes; 1 = 53 bytes. 16-bit local transmit bus: 0 = 52 bytes; 1 = 54 bytes.
tx_parity_generate	Input	Enables parity generation on the UTOPIA bus when high.
tx_addr_range[4:0]	Input	The range of transmit ports polled in MPHY mode is from 0 to tx_addr_range.
tx_tt_en	Input	Enables the logical-to-physical translation table.
tx_tt_mode	Input	Selects the logical-to-physical translation table: 0 = RAM; 1 = ROM.
tx_tt_write	Input	Write enable for logical-to-physical translation table (active high).
tx_tt_addr[4:0]	Input	Address for logical-to-physical translation table.
tx_cell_adjust[3:0]	Input	Cell size adjustment. Default is 0, other modes are not supported.
tx_tt_data[4:0]	Input	Data for logical-to-physical translation table.
reset	Input	Active-low system reset. Can be asserted asynchronously, but must be deasserted synchronously to tx_clk_in.

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

Table 3-6. UTOPIA Receive Interface (Part 1 of 2)

Signal	Direction	Description
rx_data[15/7:0]	Input	Receive data. When in 8-bit mode, rx_data[7:0] is used.
rx_soc	Input	Start of cell indicator. Active-high signal asserted when rx_data contains the first word of the cell.
rx_enb	Output	Enable. Active-low signal asserted to enable the PHY device to drive data on rx_data.

Table 3–6. UTOPIA Receive Interface (Part 2 of 2)

Signal	Direction	Description
rx_clav	Input	Cell available. Active-high signal asserted from the PHY devices when they are ready to send another cell.
rx_clk_in	Input	Receive clock. All signals on the UTOPIA/local receive interface are synchronous to it.
rx_prty	Input	Parity. <code>rx_prty</code> is the parity bit over <code>rx_data[7:0]</code> in 8-bit mode, and <code>rx_data[15:0]</code> in 16-bit mode.
rx_addr[4:0]	Output	Address. Five-bit-wide address used in MPHY mode to poll and select the various connected PHY devices.
rx_prty_pulse	Output	Active-high one-cycle pulse, indicating a parity error has been detected on <code>rx_prty</code> input.
rx_cell_pulse	Output	Active-high one-cycle pulse, indicating a cell has been received.
rx_cell_err_pulse	Output	Active-high one-cycle pulse, indicating an illegal-length cell has been received.

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

Table 3–1. Local Receive Interface

Signal	Direction	Description
atm_rx_data[15/7:0]	Output	Data output for receive data.
atm_rx_soc	Output	Asserted when <code>atm_rx_data</code> contains the first word of the cell.
atm_rx_valid	Output	Asserted when valid cell data is present on <code>atm_rx_data</code> .
atm_rx_enb	Input	Asserted when ready to accept cell data.
atm_rx_port_stat[30:0]	Output	Port status of polled PHYs. Represents the <code>rx_clav</code> status of the corresponding PHYs.
atm_rx_port[4:0]	Input	Port address for the next cell transfer.
atm_rx_port_load	Input	Active-high enable pulse, which loads the port address for the next cell transfer.
atm_rx_port_wait	Output	Active-high signal asserted to indicate that the local interface must wait before loading the next port address with <code>atm_rx_port_load</code> .

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

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

Signal	Direction	Description
atm_rx_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 <code>atm_rx_dat</code> . The data is sent most significant bit (MSB) first and all valid bits are contiguous with the MSB.
atm_rx_sop	Output	Start of packet signal. <code>atm_rx_sop</code> is used to delineate the packet boundaries on the <code>atm_rx_dat</code> bus. When <code>atm_rx_sop</code> is high, the start of the packet is present on the <code>dat</code> bus. <code>atm_rx_sop</code> is required to be present at the beginning of every packet.

Table 3–2. Atlantic Receive Interface (Part 2 of 2)

Signal	Direction	Description
atm_rx_eop	Output	<p>End of packet signal.</p> <p>atm_rx_eop is used to delineate the packet boundaries on the atm_rx_dat bus. When atm_rx_eop is high, the end of the packet is present on the atm_rx_dat bus.</p> <p>atm_rx_eop is required to be present at the end of every packet.</p>
atm_rx_val	Output	<p>Data valid signal.</p> <p>atm_rx_val indicates the validity of the data signals.</p> <p>atm_rx_val is updated on every clock edge where atm_rx_ena is sampled asserted, and holds its current value along with the atm_rx_dat bus where ena is sampled deasserted.</p> <p>When atm_rx_val is asserted, the Atlantic data interface signals are valid.</p> <p>When atm_rx_val is deasserted, the Atlantic data interface signals are invalid and must be disregarded.</p> <p>To determine whether new data has been received, the master must qualify the atm_rx_val signal with the previous state of the atm_rx_ena signal.</p>
atm_rx_ena	Input	<p>Enable signal.</p> <p>atm_rx_ena is driven by an Atlantic master interface, and used to control the flow of data across the interface.</p> <p>atm_rx_ena behaves as a read enable from Atlantic master to Atlantic slave.</p> <p>When atm_rx_ena is sampled asserted, the Atlantic data interface signals contain new data during the following clock edge. atm_rx_val indicates the validity of the data.</p> <p>The Atlantic data interface signals get new data on every clock cycle, if atm_rx_ena is asserted.</p>
atm_rx_dav	Output	<p>Data available signal.</p> <p>If atm_rx_dav is high, the slave FIFO buffer has at least one word available to be read.</p>
atm_rx_adr_stat[30:0]	Output	Port status of polled PHYs. Represents the rx_clav status of the corresponding PHYs.
atm_rx_req_adr[4:0]	Input	Port address for the next cell transfer.
atm_rx_req_adr_load	Input	Active-high enable pulse, which loads the port address for the next cell transfer.
atm_rx_req_adr_wait	Output	Active-high signal asserted to indicate that the local interface must wait before loading the next port address with atm_rx_req_adr_load.
atm_rx_adr[4:0]	Output	<p>Address Bus. atm_rx_adr carries the associated address information for each packet in multi-port implementations.</p> <p>The atm_rx_adr bus is valid at the same time as the atm_rx_dat bus and remains constant throughout a complete packet.</p>

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

Table 3-3. Receive Configuration Interface

Signal	Direction	Description
rx_phy_mode[1:0]	Input	Bit 0: 0 = MPHY mode; 1 = SPHY mode. Bit 1: (SPHY mode only) 0 = octet-level handshaking, 1 = cell-level handshaking.
atm_rx_pipe_mode	Input	0 = Non-pipelined mode; 1 = pipelined mode.
rx_width	Input	0 = 8-bit UTOPIA receive and local bus width. 1 = 16-bit UTOPIA receive and local bus width.
rx_user_bytes	Input	Defines the size of the cells at the local receive interface. 8-bit local receive bus: 0 = 52 bytes; 1 = 53 bytes. 16-bit local receive bus: 0 = 52 bytes; 1 = 54 bytes.
rx_parity_check	Input	Enables parity checking on the UTOPIA bus when high.
rx_addr_range[4:0]	Input	The range of receive ports polled in MPHY mode is from 0 to rx_addr_range.
rx_cell_adjust[3:0]	Input	Cell size adjustment. Default is 0, other modes are not supported.
rx_tt_en	Input	Enables the logical-to-physical translation table.
rx_tt_mode	Input	Selects the logical-to-physical translation table: 0 = RAM; 1 = ROM.
rx_tt_write	Input	Write enable for logical-to-physical translation table (active high).
rx_tt_addr[4:0]	Input	Address for logical-to-physical translation table.
rx_tt_data[4:0]	Input	Data for logical-to-physical translation table.
reset	Input	Active-low system reset. Can be asserted asynchronously, but must be deasserted synchronously to rx_clk_in.

Interfaces

This section describes the following interfaces:

- UTOPIA transmit
- Local transmit
- Atlantic slave transmit
- UTOPIA receive
- Local receive
- Atlantic slave receive
- Atlantic Interface

UTOPIA Transmit Interface

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

In MPHY mode, the master polls the various slaves using `tx_addr`. It polls the logical addresses ranging from address 0 to the programmed value of `tx_addr_range`. The slaves respond by driving their `tx_clav` output to allow the master to determine whether the slaves can accept a cell. The `atm_tx_port_stat` and `atm_tx_port_stat` outputs store the polling result for each slave address, which allows the local transmit interface to determine which slaves can accept cells. 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*.



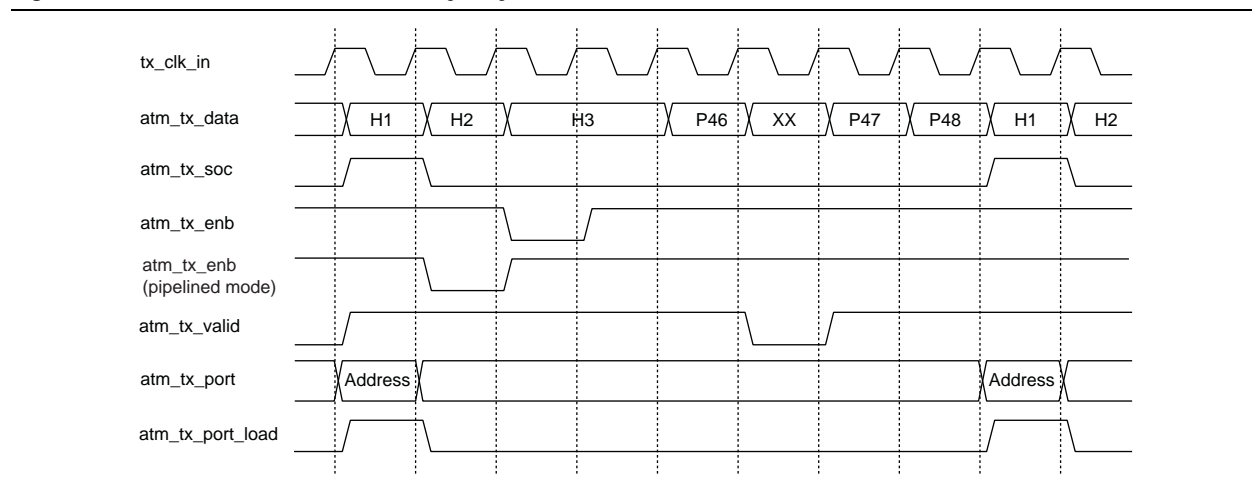
The UTOPIA Level 2, Version 1.0 specification is available from www.atmforum.com

In SPHY mode, the slave indicates that it can accept a cell by asserting the `tx_clav` output. 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*.

Local Transmit Interface

The local transmit interface provides the ability to transmit ATM cell data along with its associated port (slave) address, ready for transmission on the UTOPIA bus. To avoid stalling the UTOPIA transmit interface, a new ATM cell transmission should be initiated only when the relevant bit of `atm_tx_port_stat` is active. The data path (`atm_tx_data`) can operate in either 8- or 16-bit mode, but it always has the same bus width as the UTOPIA transmit interface. The cell size can be 52, 53, or 54 bytes long. The local transmit interface indicates that it can accept data on `atm_tx_data` by asserting `atm_tx_enb` high. Before you transmit a cell, you must load a new port using `atm_tx_port_load`. With the cell's first data word, you should assert `atm_tx_soc` high. The master asserts the `atm_tx_port_wait` signal for a number of clock cycles after `atm_tx_port_load` until the cell starts transmission on the UTOPIA interface. While it is high, you must not load the port for the next cell. You can load a new port, and start the cell transfer simultaneously (see [Figure 3-1](#)).

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

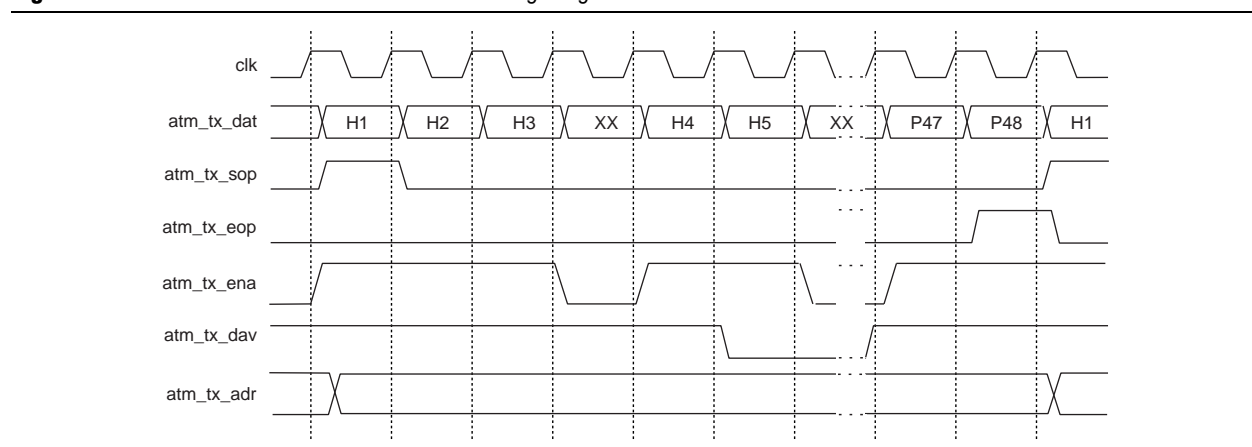
Figure 3-1. Local Transmit Interface Timing Diagram

Atlantic Slave Transmit Interface

The Atlantic slave transmit interface provides the ability to transmit ATM cell data along with its associated port (slave) address, ready for transmission on the UTOPIA bus. To avoid stalling the UTOPIA transmit interface, a new ATM cell transmission should be initiated only when the relevant bit of `atm_tx_adr_stat` is active. The data path (`atm_tx_dat`) can operate in either 8- or 16-bit mode, but it always has the same bus width as the UTOPIA transmit interface. The cell size can be 52, 53, or 54 bytes long. The Atlantic slave transmit interface indicates that it can accept data on `atm_tx_dat` by asserting `atm_tx_dav` high. With the cell's first data word, you should assert `atm_tx_sop` high.

The Atlantic slave controls the transfer of data across this interface by asserting `atm_tx_dav` high when it is ready to accept data. Data is transferred when `atm_tx_ena` is high. To avoid the FIFO buffer overflowing, the Atlantic master must stop transferring data the clock cycle after `dav` transitions low.

Figure 3-2 shows the Atlantic slave transmit interface timing diagram.

Figure 3-2. Atlantic Slave Transmit Interface Timing Diagram

UTOPIA Receive Interface

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

In MPHY mode, the master polls the various slaves using the `rx_addr` bus. It polls the logical addresses ranging from address 0 to the programmed value of `rx_addr_range`. The slaves respond by driving their `rx_clav` output to allow the master to determine whether they have any cells ready for transfer. The `atm_rx_port_stat` and `atm_rx_adr_stat` output stores the polling result for each slave address to allow the local transmit interface to determine which slaves can accept cells. 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.

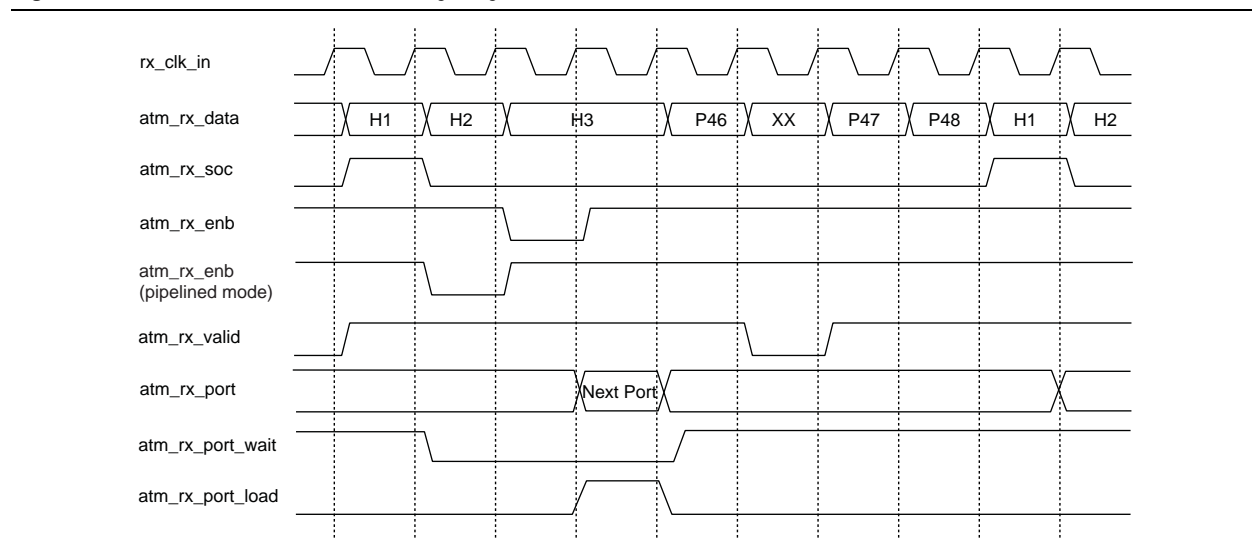
In SPHY mode, the slave indicates that it has a cell ready for transfer by asserting the `rx_clav` output. The master subsequently initiates the transfer of a cell from the slave by asserting `rx_enb` low. This behavior is described in section 3 of the UTOPIA Level 2, Version 1.0 specification.

Local Receive Interface

The local receive interface provides the ability to receive ATM cell data from its associated port (slave) address across the UTOPIA bus. To avoid stalling the UTOPIA receive interface, a new ATM cell request should be initiated only when the relevant bit of `atm_rx_port_stat` is active. The data path (`atm_rx_data`) can operate in either 8- or 16-bit mode, but it always has the same bus width as the UTOPIA receive interface. The cell size can be 52, 53, or 54 bytes long. Before a cell is received, a new port must be loaded using `atm_rx_port_load`. This starts a cell transfer across the UTOPIA bus when a cell is available. The UTOPIA master asserts the `atm_rx_port_wait` signal for a number of clock cycles after `atm_rx_port_load` until the cell starts being received across the UTOPIA interface. While it is high, the port for the next cell should not be loaded. The local receive interface indicates that cell data is available on `atm_rx_data` by asserting `atm_rx_valid` high; `atm_rx_soc` is asserted high when the first data word of the cell is on `atm_rx_data`.

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

Figure 3–3 shows the local receive interface timing diagram.

Figure 3-3. Local Receive Interface Timing Diagram

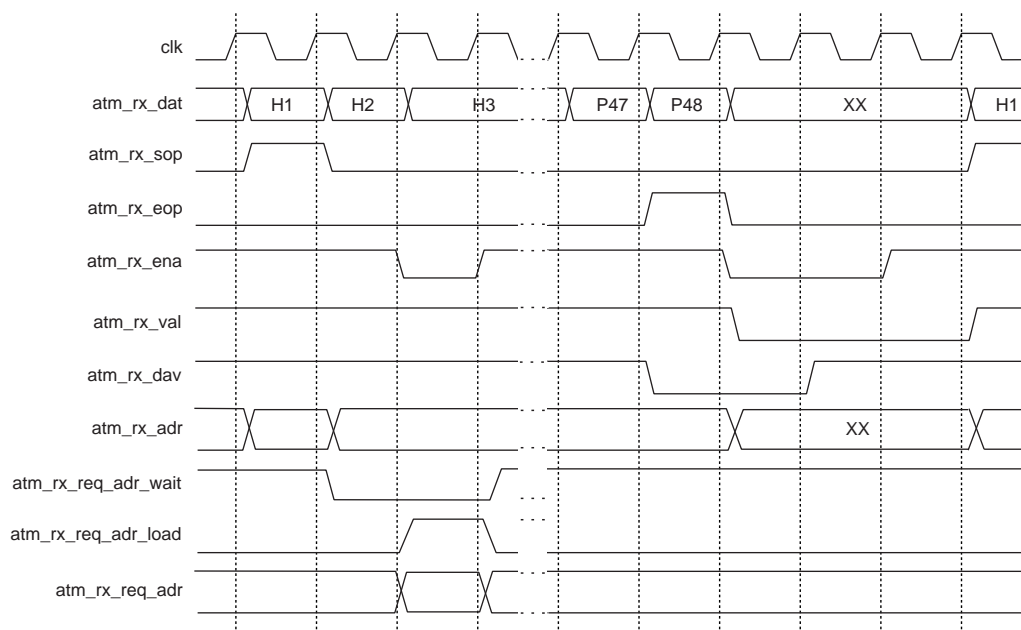
Atlantic Slave Receive Interface

The Atlantic slave receive interface provides the ability to receive ATM cell data from its associated port (slave) address across the UTOPIA bus. To avoid stalling the UTOPIA receive interface, a new ATM cell request should be initiated only when the relevant bit of `atm_rx_adr_stat` is active. The data path (`atm_rx_dat`) can operate in either 8- or 16-bit mode, but it always has the same bus width as the UTOPIA receive interface. The cell size can be 52, 53, or 54 bytes long. Before a cell is received, a new port must be loaded using `atm_rx_req_adr_load`. This starts a cell transfer across the UTOPIA bus when a cell is available. The UTOPIA master asserts the `atm_rx_req_adr_wait` signal for a number of clock cycles after `atm_rx_adr_load` until the cell starts being received across the UTOPIA interface. While it is high, the port for the next cell should not be loaded.

The Atlantic slave indicates that it has data available by asserting `atm_rx_dav` high. The Atlantic master controls the transfer of data across this interface by asserting `atm_rx_ena` high when it is ready to accept data. When `atm_rx_ena` is sample asserted, the data interface signals contain new data at the following clock edge. `atm_rx_val` indicates the validity of the data. `atm_rx_sop` is asserted high when the first data word of the cell is on `atm_rx_dat`.

Figure 3-4 shows the Atlantic slave receive interface timing diagram.

Figure 3-4. Atlantic Slave Receive Interface Timing Diagram



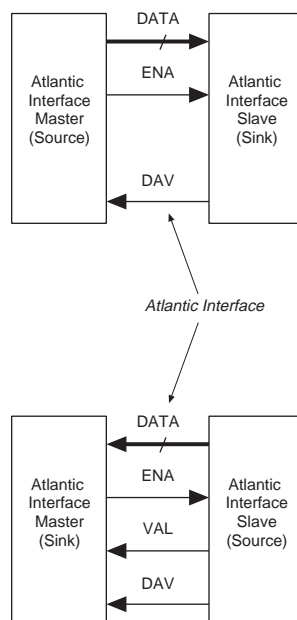
Atlantic Interface

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

- 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 3-5 on page 3-14 shows the following four Atlantic interface control options:

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

Figure 3–5. Atlantic Interface Control Options**Compatibility**

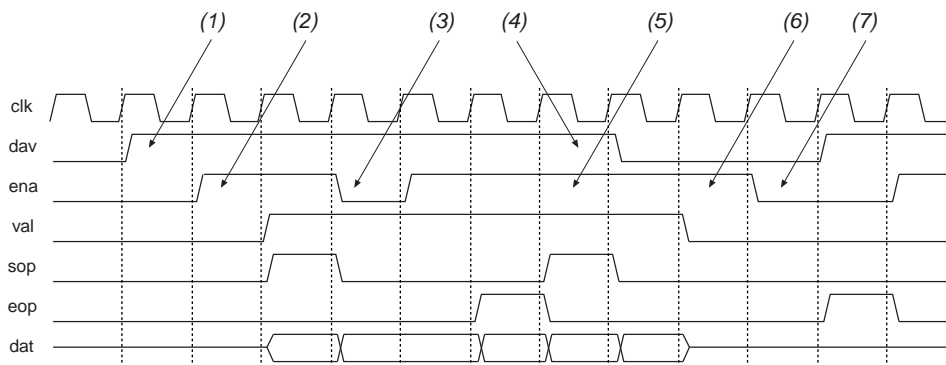
To ensure that individual implementations of an Atlantic interface are compatible they must have the following:

- 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 3–6 on page 3–15 shows the timing of the Atlantic interface when in master mode. Figure 3–7 on page 3–15 shows the timing of the Atlantic interface when in slave mode.

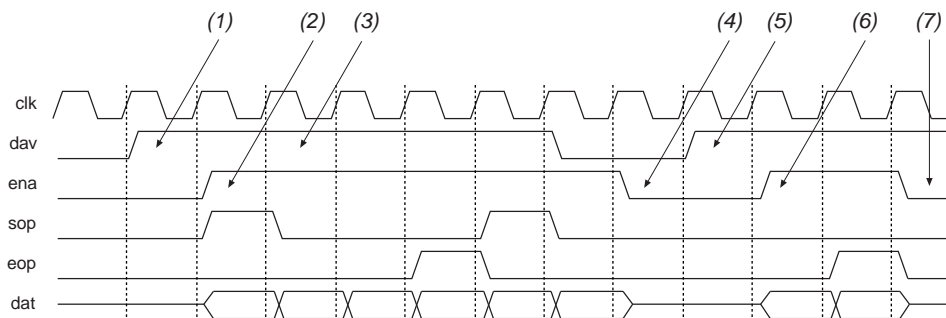
Figure 3-6. Atlantic Interface Timing—Slave Source



Notes to Figure 3-6:

- (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.
- (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 re-asserted.

Figure 3-7. Atlantic Interface Timing—Slave Sink



Notes to Figure 3-7:

- (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.
- (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.

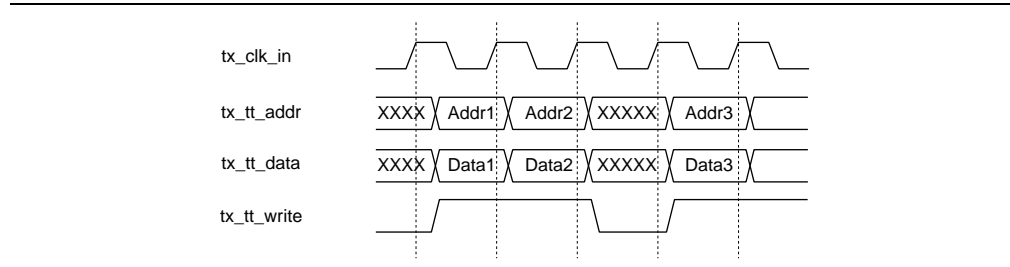
Address Translation

The receiver and the transmitter have the option of using a logical-to-physical address translation table. The master, when in MPHY mode, always polls logical addresses from zero up to address_range. The address_translation signal can be used if the master must poll physical addresses that are not contiguous, or do not start at zero.

When you select `address_translation`, IP Toolbench allows you to map any logical address to any physical address.

Additionally, the address translation table can be implemented either in ROM or in RAM. If you know what the mappings are, IP Toolbench allows the table to be set up and generates ROM. However, if you want to specify the mappings after using IP Toolbench (for example, if the slave addresses change regularly), implementing the address translation table in RAM is an option. When implementing the address translation table in RAM, a waveform must be generated that loads data into the address translation table (see Figure 3-8).

Figure 3-8. Translation Table Programming Interface Timing Diagram



MegaCore 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 tables.
December 2006	6.1	<ul style="list-style-type: none"> Updated the release information and device family support tables Updated the performance tables Added a procedure for running a testbench simulation with NativeLink

How to Contact Altera

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

Contact <i>(Note 1)</i>	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) (Software Licensing)	Email	nacomp@altera.com
	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
<i>Italic Type with Initial Capital Letters</i>	Indicates document titles. For example: <i>AN 519: Stratix IV Design Guidelines</i> .
<i>Italic type</i>	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.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	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.