

This chapter describes the example designs and the traffic generator.

Two independent example designs are created during generation with the MegaWizard Plug-In Manager. These example designs illustrate how to instantiate and connect the memory interface for both synthesis and simulation flows.

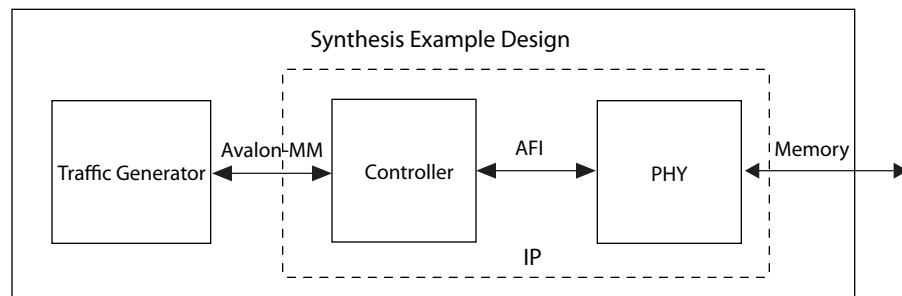
The two example designs are completely independent, and contain independent RTL files and other project files; they should be compiled or simulated separately, and the files should not be mixed. Nonetheless, the designs are related, as the simulation example design builds upon the design of the synthesis example design.

Synthesis Example Design

The synthesis example design contains the following major blocks, as shown in [Figure 7-1](#):

- A traffic generator, which is a synthesizable Avalon-MM example driver that implements a pseudo-random pattern of reads and writes to a parameterized number of addresses. The traffic generator also monitors the data read from the memory to ensure it matches the written data and asserts a failure otherwise.
- An instance of the UniPHY memory interface, which includes a memory controller that moderates between the Avalon-MM interface and the AFI interface, and the UniPHY, which serves as an interface between the memory controller and external memory devices to perform read and write operations.


Figure 7-1. Synthesis Example Design



© 2011 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. 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. 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.



You can obtain the synthesis example design by generating your IP core using the MegaWizard Plug-In Manager flow. The files related to the synthesis example design reside at `<variation_name>_example_design/example_project`. The synthesis example design includes a Quartus II project file (`<variation_name>_example_design/example_project/<variation_name>_example.qpf`). The Quartus II project file can be compiled in the Quartus II software, and can be run on hardware.

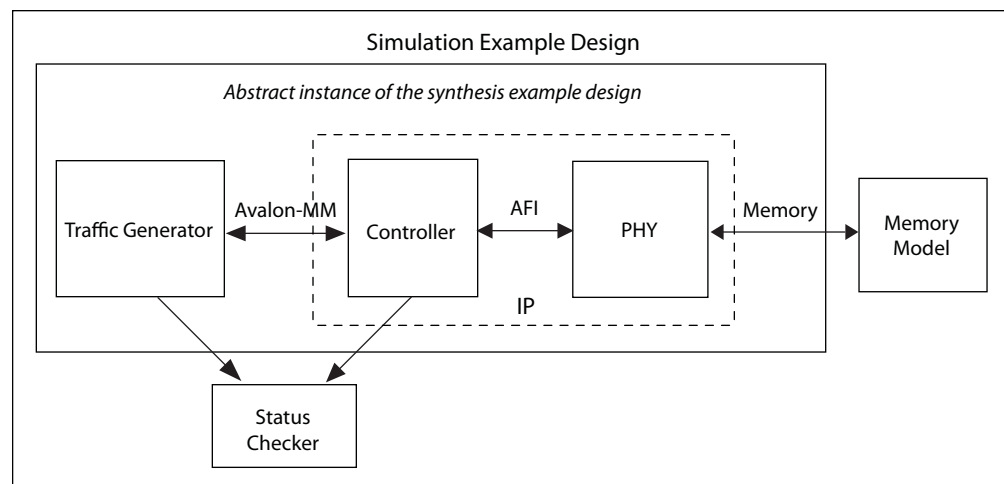
 If one or more of the **PLL Sharing Mode**, **DLL Sharing Mode**, or **OCT Sharing Mode** parameters are set to any value other than **No Sharing**, the synthesis example design will contain two traffic generator/memory interface instances. The two traffic generator/memory interface instances are related only by shared PLL/DLL/OCT connections as defined by the parameter settings. The traffic generator/memory interface instances demonstrate how you can make such connections in your own designs.

Simulation Example Design

The simulation example design contains the following major blocks, as shown in [Figure 7-2](#):

- An instance of the synthesis example design. As described in the previous section, the synthesis example design contains a traffic generator and an instance of the UniPHY memory interface. These blocks default to abstract simulation models where appropriate for rapid simulation.
- A memory model, which acts as a generic model that adheres to the memory protocol specifications. Frequently, memory vendors provide simulation models for specific memory components that you can download from their websites.
- A status checker, which monitors the status signals from the UniPHY IP and the traffic generator, to signal an overall pass or fail condition.

Figure 7-2. Simulation Example Design



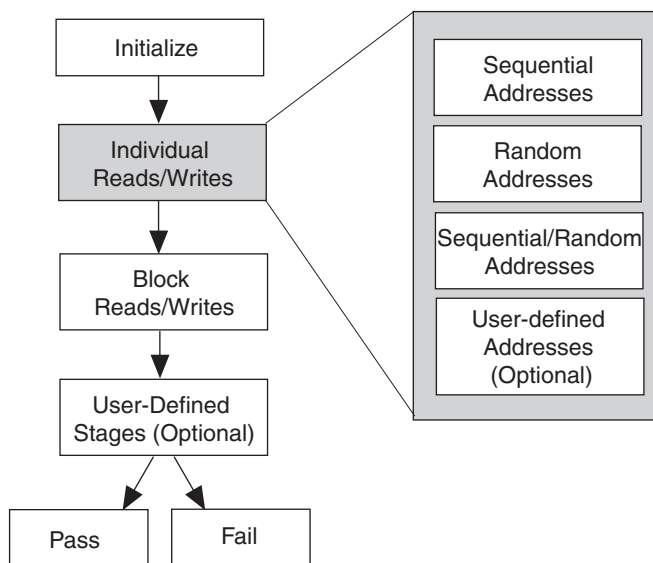
You can obtain the simulation example design by generating your IP core with the MegaWizard Plug-In Manager. The files related to the simulation example design reside at `<variation_name>_example_design/simulation`. After obtaining the files generated by the MegaWizard Plug-In Manager, you must still generate the simulation example design RTL for your desired HDL language. The file `<variation_name>_example_design/simulation/README.txt` contains details about how to generate the IP and to run the simulation in ModelSim-AE/SE.

Traffic Generator and BIST Engine

The traffic generator and built-in self test (BIST) engine for Avalon-MM memory interfaces generates Avalon-MM traffic on an Avalon-MM master interface. The traffic generator creates read and write traffic, stores the expected read responses internally, and compares the expected responses to the read responses as they arrive. If all reads report their expected response, the pass signal is asserted; however, if any read responds with unexpected data a fail signal occurs.

Each operation generated by the traffic generator is a single write or block of writes followed by a single read or block of reads to the same addresses, which allows the driver to precisely determine the data that should be expected when the read data is returned by the memory interface. The traffic generator comprises a traffic generation block, the Avalon-MM interface and a read comparison block. The traffic generation block generates addresses and write data, which are then sent out over the Avalon-MM interface. The read comparison block compares the read data received from the Avalon-MM interface to the write data from the traffic generator. If at any time the data received is not the expected data, the read comparison block records the failure, finishes reading all the data, and then signals that there is a failure and the traffic generator enters a fail state. If all patterns have been generated and compared successfully, the traffic generator enters a pass state.

Figure 7-3. Example Driver Operations



Within the traffic generator, there are the following main states:

- Generation of individual read and writes
- Generation of block read and writes
- The pass state
- The fail state

Within each of the generation states there are the following substates:

- Sequential address generation
- Random address generation
- Mixed sequential and random address generation

For each of the states and substates, the order and number of operations generated for each substate is parameterizable—you can decide how many of each address pattern to generate, or can disable certain patterns entirely if you want. The sequential and random interleave substate takes in additions to the number of operations to generate. An additional parameter specifies the ratio of sequential to random addresses to generate randomly.

Read and Write Generation

The traffic generator block can generate individual or block reads and writes.

Individual Read and Write Generation

During the traffic generator's individual read and write generation state, the traffic generation block generates individual write followed by individual read Avalon-MM transactions, where the address for the transactions is chosen according to the specific substate. The width of the Avalon-MM interface is a global parameter for the driver, but each substate can have a parameterizable range of burst lengths for each operation.

Block Read and Write Generation

During the traffic generator's block read and write generation state, the traffic generator block generates a parameterizable number of write operations followed by the same number of read operations. The specific addresses generated for the blocks are chosen by the specific substates. The burst length of each block operation can be parameterized by a range of acceptable burst lengths.

Address and Burst Length Generation

The traffic generator block can perform sequential or random addressing.

Sequential Addressing

The sequential addressing substate defines a traffic pattern where addresses are chosen in sequential order starting from a user definable address. The number of operations in this substate is parameterizable.

Random Addressing

The random addressing substate defines a traffic pattern where addresses are chosen randomly over a parameterizable range. The number of operations in this substate is parameterizable.

Sequential and Random Interleaved Addressing

The sequential and random interleaved addressing substate defines a traffic pattern where addresses are chosen to be either sequential or random based on a parameterizable ratio. The acceptable address range is parameterizable as is the number of operations to perform in this substate.

Traffic Generator Signals

Table 7-1 lists the signals used by the traffic generator.

Table 7-1. Traffic Generator Signals

Signal	Width	Signal Type
clk		
reset_n		
avl_ready		avl_ready
avl_write_req		avl_write_req
avl_read_req		avl_read_req
avl_addr	24	avl_addr
avl_size	3	avl_size
avl_wdata	72	avl_wdata
avl_rdata	72	avl_rdata
avl_rdata_valid		avl_rdata_valid
pnf_per_bit		pnf_per_bit
pnf_per_bit_persist		pnf_per_bit_persist
pass		pass
fail		fail
test_complete		test_complete



For information about the Avalon signals and the Avalon interface, refer to [Avalon Interface Specifications](#).

Traffic Generator Add-Ons

Some optional components that can be useful for verifying aspects of the controller and PHY operation are generated in conjunction with certain user-specified options. These add-on components are self-contained, and are not part of the controller or PHY, nor the traffic generator.

User Refresh Generator

The user refresh generator sends refresh requests to the memory controller when user refresh is enabled. The memory controller returns an acknowledgement signal and then issues the refresh command to the memory device.

The user refresh generator is created when you turn on **Enable User Refresh** on the **Controller Settings** tab of the parameter editor.

Traffic Generator Timeout Counter

The traffic generator timeout counter uses the Avalon interface clock.

When a test fails due to driver failure or timeout, the `fail` signal is asserted. When a test has failed, the traffic generator must be reset with the `reset_n` signal.

Creating and Connecting the UniPHY Memory Interface and the Traffic Generator in Qsys

The traffic generator can be used in Qsys as a stand-alone component for use within a larger system. This section explains how to instantiate and configure the example driver, and includes tips on configuring a UniPHY memory interface which can apply in many general situations.

Creating the Qsys System

To create the system in Qsys, perform the following steps:

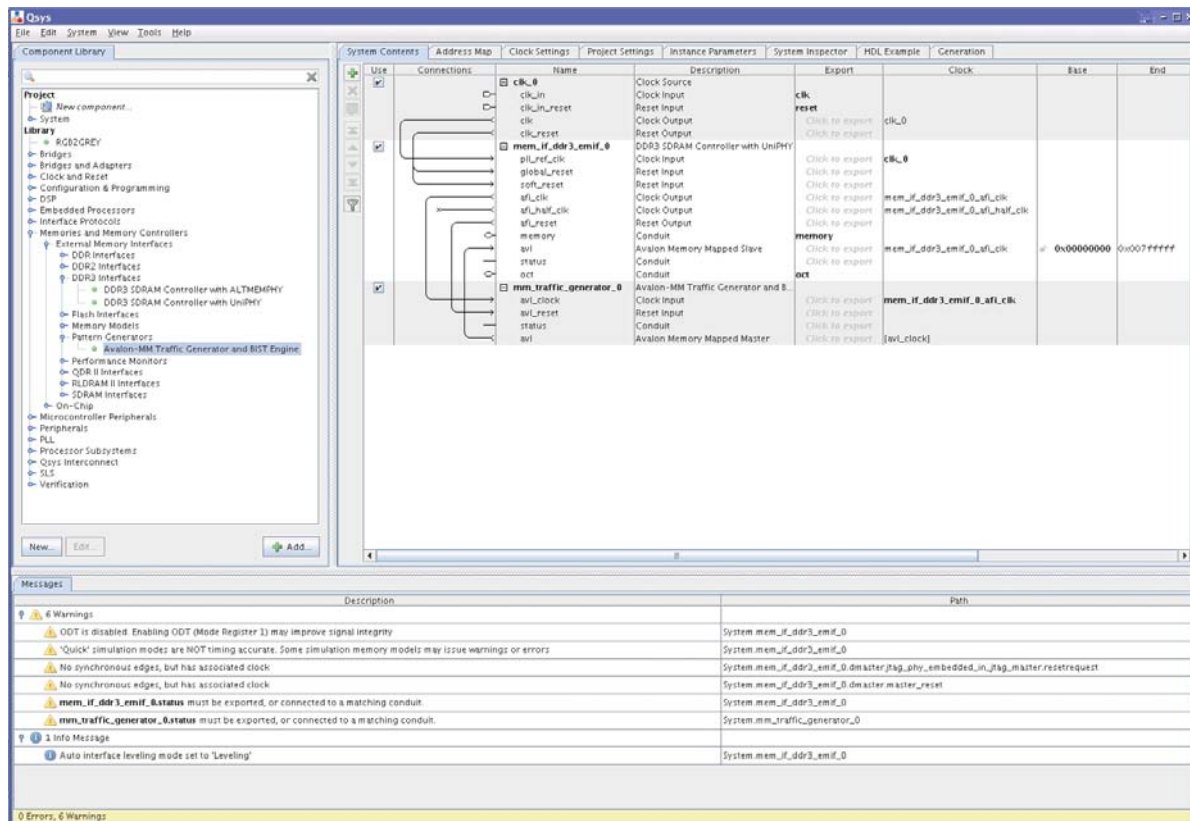
1. Start Qsys.
2. On the **Project Settings** tab, select the required device from the **Device Family** list.
3. In the **Component Library**, choose a UniPHY memory interface to instantiate. For example, under **Library > Memories and Memory Controllers > External Memory Interfaces**, select **DDR3 SDRAM Controller with UniPHY**.
4. Configure the parameters for your instantiation of the memory interface.
5. In the Component Library, find the example driver and instantiate it in the system. For example, under **Library > Memories and Memory Controllers > Pattern Generators**, select **Avalon-MM Traffic Generator and BIST Engine**.
6. Configure the parameters for your instantiation of the example driver.



The Avalon specification is that Avalon-MM master interfaces issue byte addresses, while Avalon-MM slave interfaces accept word addresses. The default for the Avalon-MM Traffic Generator and BIST Engine is to issue word addresses. When using Qsys or SOPC Builder, you must enable the **Generate per byte address** setting in the traffic generator.

7. Connect the interfaces as illustrated in Figure 7-4. At this point, you can generate synthesis RTL, Verilog or VHDL simulation RTL, or a simulation testbench system.

Figure 7-4. Qsys System



Notes on Using UniPHY IP in Qsys

This section includes notes and tips on using the UniPHY IP in Qsys.

- The address ranges shown for the Avalon-MM slave interface on the UniPHY component should be interpreted as byte addresses that an Avalon-MM master would address, despite the fact that this range is modified by configuring the word addresses width of the Avalon-MM slave interface on the UniPHY controller.
- The `afi_clk` clock source is the associated clock to the Avalon-MM slave interface on the memory controller. This is the ideal clock source to use for all IP components connected on the same Avalon network. Using another clock would cause Qsys to automatically instantiate clock-crossing logic, potentially degrading performance.
- The `afi_clk` clock rate is determined by the **Rate on Avalon-MM interface** setting on the UniPHY **PHY Settings** tab. The `afi_half_clk` clock interface has a rate which is further halved. For example, if **Rate on Avalon-MM interface** is set to **Half**, the `afi_clk` rate is half of the memory clock frequency, and the `afi_half_clk` is one quarter of the memory clock frequency.

- The `global_reset` input interface can be used to reset the UniPHY memory interface and the PLL contained therein. The `soft_reset` input interface can be used to reset the UniPHY memory interface but allow the PLL to remain locked. You can use the `soft_reset` input to reset the memory but to maintain the AFI clock output to other components in the system.
- Do not connect a reset request from a system component (such as a Nois II processor) to the UniPHY `global_reset_n` port. Doing so would reset the UniPHY PLL, which would propagate as a reset condition on `afi_reset` back to the requester; the resulting reset loop could freeze the system.
- Qsys generates an interconnect fabric for each Avalon network. The interconnect fabric is capable of burst and width adaptation. If your UniPHY memory controller is configured with an Avalon interface data width which is wider than an Avalon-MM master interface connected to it, you must enable the byte enable signal on the Avalon-MM slave interface, by checking the **Enable Avalon-MM byte-enable signal** checkbox on the **Controller Settings** tab in the parameter editor.
- If you have a point-to-point connection from an Avalon-MM master to the Avalon-MM slave interface on the memory controller, and if the Avalon data width and burst length settings match, then the Avalon interface data widths may be multiples of either a power of two or nine. Otherwise, you must enable **Generate power-of-2 data bus widths for Qsys or SOPC Builder** on the **Controller Settings** tab of the parameter editor.

Document Revision History

Table 7-2 lists the revision history for this document.

Table 7-2. Document Revision History

Date	Version	Changes
November 2011	1.1	<ul style="list-style-type: none"> ■ Added Synthesis Example Design and Simulation Example Design sections. ■ Added Creating and Connecting the UniPHY Memory Interface and the Traffic Generator in Qsys. ■ Revised Example Driver section as Traffic Generator and BIST Engine.