Qsys System Design Tutorial

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

TU-01006

Document last updated for Altera Complete Design Suite version: 11.0
Document publication date: April 2011

Subscribe

Contents

1. Introduction

This tutorial introduces you to the Qsys system integration tool available with the Quartus® II software. This tutorial shows you how to design a system that uses various test patterns to test an external memory device. This tutorial guides you through system requirement analysis, hardware design tasks, and evaluation of the system performance, with emphasis on architecting the system. Upon completion you understand the Qsys development flow, and you can design your own systems.

Software and Hardware Requirements

This tutorial requires the following software:

- Altera® Quartus II software version 11.0 or later.

  For system requirements and installation instructions, refer to *Altera Software Installation and Licensing*.

- Nios® II EDS version 11.0 or later.
- **tt_qsys_design.zip** design example files, available from the Qsys Tutorial Design Example web page. The design example files include project files set up for select Altera development boards.

You can build the system in this tutorial for any Altera development board or your own custom board if it meets the following requirements:

- The board must have an Altera Arria®, Cyclone®, or Stratix® series FPGA.
- The FPGA must contain a minimum of 12 K logic elements (LEs) or adaptive lookup tables (ALUTs).
- The FPGA must contain a minimum of 150 Kbits of embedded memory.
- The board must have a JTAG connection to the FPGA that provides a communications link back to the host so that you can monitor the memory test progress.
- The board must contain a memory that the design tests. For example, any memory that has a Qsys-based controller with an Avalon® Memory-Mapped (Avalon-MM) slave interface.

To complete this tutorial for development boards other than those already set up with the design example files, refer to the board documentation for the clock frequencies and pin-out descriptions. For Altera development boards, you can find this information in the relevant reference manual. Altera provides hardware projects with porting steps for various other development boards.

Overview

The Qsys system you build in this tutorial tests a synchronous dynamic random access memory (SDRAM). The final system contains the SDRAM controller, and instantiates a Nios II processor and some embedded peripherals in a hierarchical subsystem. You complete the Qsys system by adding various Qsys components that generate test data, access memory, and verify the returned data.

The final system contains the following components:

- Processor subsystem based on the Nios II/e core (included with the Altera Complete Design Suite)
- SDRAM controller (included with the Altera Complete Design Suite)
- Pseudo-random binary sequence (PRBS) pattern generator and checker
- Custom pattern generator and checker
- Pattern select multiplexer and demultiplexer
- Pattern writer and reader
- Memory test controller

You can use this final system on hardware without a license. With Altera's free OpenCore Plus evaluation feature, you can perform the following actions:

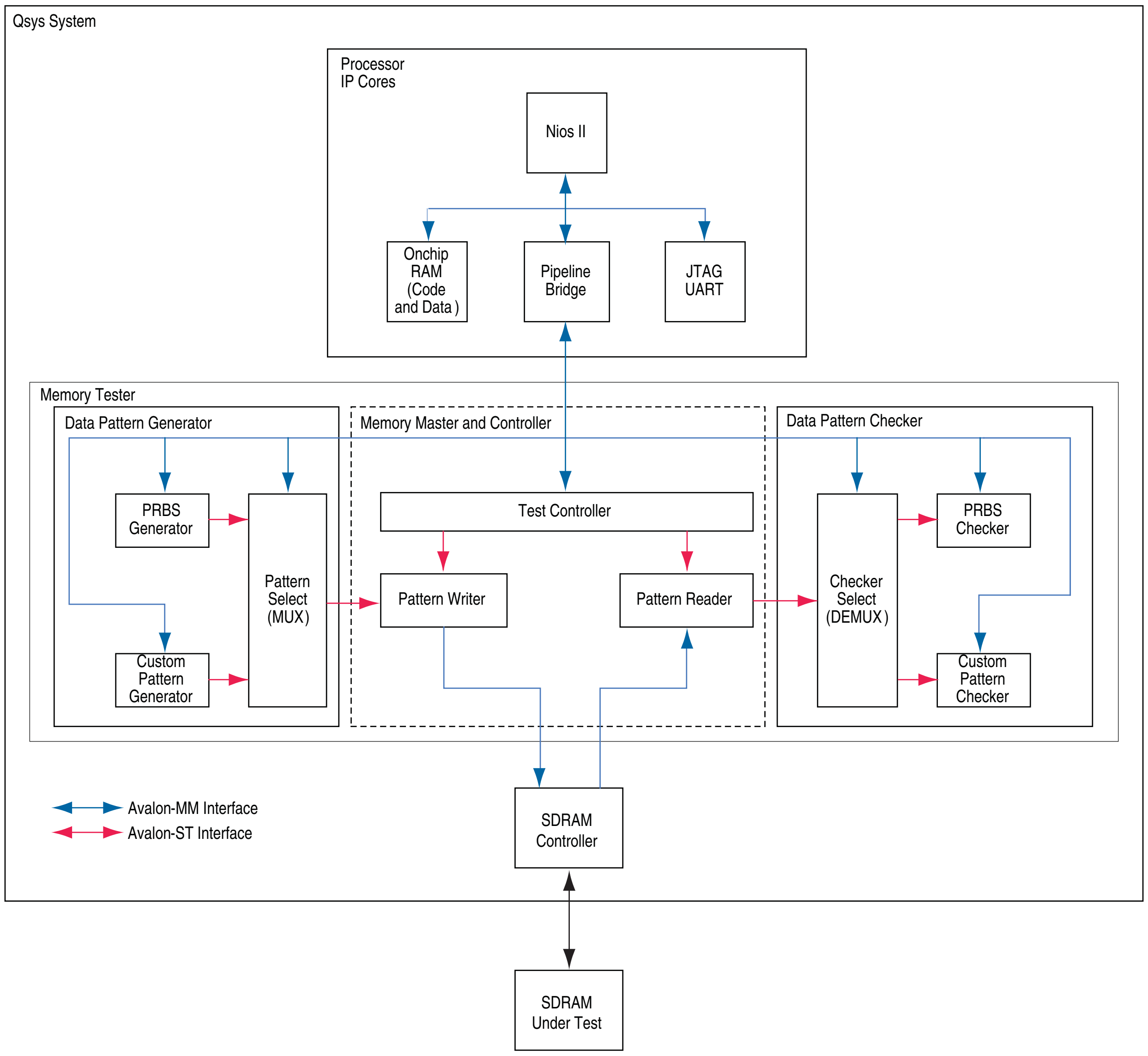
- Simulate the behavior of the system and verify its functionality
- Generate time-limited device programming files for your designs
- Program a device and verify your design in hardware

The design example files comprise components that are free to use in any design. The Nios II 'e' processor core and the DDR SDRAM IP core are free to use with a Quartus II subscription license. The design files for different development kit boards use different DDR SDRAM controllers, to match the memory device available on the development kit.

- For more information about OpenCore Plus, refer to the *AN320: OpenCore Plus Evaluation of Megafunctions*.

Figure 1–1 shows the complete top-level system of the design example. Figure 1–1 shows the components in the memory tester system as one Qsys system, with the three major design functions grouped with dotted lines. This tutorial shows using hierarchy—instantiating the data pattern generator and data pattern checker components into separate systems, which you then include in the memory tester system. Hierarchy allows you to instantiate a system as a component in a higher-level system.

Figure 1–1. Top-Level System Architecture



Downloading and Installing the Design Example Files

To download and install the design example files for the tutorial, follow these steps:

1. Download the Qsys Tutorial Design Example (.zip) files from the Qsys Tutorial Design Example web page.

2. Extract the contents of the archive file to a directory on your computer. Do not use spaces in the directory path name.

Opening the Tutorial Project

The design example files for this tutorial provide the required custom IP design blocks and project files to use as a starting point and include a partially completed Quartus II project and Qsys system. The design example files include the following complete projects:

- Assigned Quartus II project I/O pin assignments and specified Synopsys Design Constraints (**.sdc**) timing assignments.
- Parameterized Nios II processor core to communicate with the host PC that controls the memory test system you develop.
- Parameterized DDR SDRAM controller to use the memory on the development board.

To open the tutorial project, follow these steps:

1. Open the Quartus II software.
2. Open the Quartus II Project File (**.qpf)** for your board:
   a. On the File menu, click **Open Project.**
   b. Browse to the **tt_qsys_design\quartus_ii_projects_for_boards\**<*development_board*>\ directory.
   c. Select the relevant board-specific **.qpf** file, and click **Open**.

The custom memory test components for the design are Verilog HDL components with an accompanying Hardware Component Description File (**_hw.tcl**) that describes the interfaces and parameterization of each component. These files are in the **tt_qsys_design\memory_tester_ip** directory. To view these components in Qsys, on the **Component Library** tab expand **Memory Test Microcores**. An IP Index (**.ipx**) file provides a reference to the **memory_tester_ip** directory that contains these memory test components.

2. Creating Qsys Systems

This chapter shows you how to instantiate, parameterize, and connect components to create Qsys systems.

In this chapter you create Qsys systems for the following design blocks that Figure 1–1 on page 1–3 shows:

- Data pattern generator
- Data pattern checker

☞ If you are familiar with the procedure for creating Qsys systems, you may skip this chapter and go to Chapter 3, Assembling Hierarchical Systems. The tutorial design files include the completed systems from this chapter.

The data pattern checker generates high-speed streaming data, which performs either as a PRBS or as a soft programmable sequence, for example, “walking ones”. The design sends the data with an Avalon-Streaming (Avalon-ST) connection to the pattern writer of the memory master and control logic.

The data pattern generator writes the data to memory based on commands issued to it by the controller logic. When the design writes the data to memory, the pattern reader logic reads the contents back and send them to the data pattern verification logic.

The data pattern checker accepts the data read back by the pattern reader from an Avalon-ST connection. The design verifies the data pattern to ensure that the pattern it writes to memory is identical to the data that it reads back.

☞ As you add components and make connections in the system, error and warning messages in the Qsys **Messages** tab indicate steps that you must perform before the system is compete. Some of the error messages are not resolved immediately, and are resolved in later steps of the procedures.

Creating the Data Pattern Generator

In this section, you create a data pattern generator system, which includes two components to generate test patterns and a third component to multiplex the data that a processor controls.You configure the data pattern generator to match the width of the memory interface. Because the data pattern generator can provide a full word of data every clock cycle, configuring the components to match the memory width provides sufficient bandwidth to access the memory rapidly.

☞ Before you create this Qsys system, ensure you download and install the tutorial files and open the Quartus II project (“Downloading and Installing the Design Example Files” on page 1–3 and “Opening the Tutorial Project” on page 1–4)

Creating a New Qsys System with a Clock Source

To create a new Qsys system and set up the clock source, follow these steps:

1. In the Quartus II software, on the Tools menu, click **Qsys**.
2. In Qsys, on the File menu, click **New System**. Qsys opens and displays a new empty system. In the **System Contents** tab, Qsys shows a clock source instance, **clk_0**.
3. To open the clock source settings, right-click **clk_0** and click **Edit** or double-click on the instance.
4. Turn off **Clock frequency is known** to indicate that, when created, the higher-level hierarchical system that instantiates this subsystem provides the clock frequency.
5. Click **Finish**.
6. Save and name the system:
   a. On the File menu, click **Save As**.
   b. Type file name `pattern_generator_system` and click **Save**. Ensure you use the exact system name described in these steps, because the tutorial scripts are configured to use this name.

Adding a Pipeline Bridge

The components that make up this system include several Avalon-MM slave interfaces. To allow a higher-level system to access all the Avalon-MM slave interfaces by reading and writing to a single slave interface, you consolidate the slave interfaces behind an Avalon-MM pipeline bridge and export a single Avalon-MM slave interface out of this system. The bridge also adds a level of pipelining, which can improve timing performance. To add the pipeline bridge, follow these steps:

1. On the **Component Library** tab, expand **Bridges and Adapters**, and then expand **Memory Mapped**. Alternatively, you can type `bridge` in the search box to filter the list and show only the bridge components. You should click **X** next to the search box, to clear the search filtering.
2. Click **Avalon-MM Pipeline Bridge** component and click **Add**. Alternatively, you can double-click on **Avalon-MM Pipeline Bridge**. The parameter editor opens.
3. In the parameter editor, for the **Address width** enter `11` to accommodate the span of the memory-mapped components in this system.
4. Click **Finish**. The default bridge is added to your system with the instance name **mm_bridge_0**.
5. Set the **mm_bridge_0** clock domain to **clk_0**:
   - In the **Clock** column for the **mm_bridge_0** `clk` interface, select **clk_0** from the drop-down list.
   - Alternatively, you can make the connection in the **Connections** column. Click to fill in the connection dot between the **clk_0** `clk` output and the **mm_bridge_0** `clk` input.
   - Alternatively, you can right-click on **mm_bridge_0** `clk` input, point to **mm_bridge_0.clk Connections**, and select **clk_0.clk**.
6. Export the **mm_bridge_0** `s0` interface with the name slave. Click in the **Export** column and type `slave`.

Adding a Custom Pattern Generator

You can configure the custom pattern generator to generate multiple test patterns. The component is programmed with the pattern data and a pattern length. When the end of the pattern is reached the custom pattern generator cycles back to the first element of the pattern. This component generates the following patterns:

- Walking ones
- Walking zeros
- Low frequency
- Alternating low frequency
- High frequency
- Alternating high frequency
- Synchronous PRBS

The synchronous PRBS pattern is the longest pattern containing 256 elements before repeating. The width of the memory dictates the walking ones or zeros pattern lengths. For example, when testing a 32-bit memory the walking ones or zeros pattern is 32 elements in length before repeating. The high and low frequency patterns contain only two elements before repeating.

This custom pattern generator contains three interfaces, two of which control the generated pattern. The interface controls the behavior of the custom pattern generated. The processor accesses the `pattern_access` interface, which is write only, to program the elements of the custom pattern that are sent to the pattern writer core. The `st_pattern_output` is the streaming source interface that sends data to the pattern writer core. To add the custom pattern generator, follow these steps:

1. On the **Component Library** tab, under **Project** expand **Memory Test Microcores**, and double-click **Custom Pattern Generator**. The parameter editor appears.
2. To accept the default parameters, click **Finish** in the parameter editor.
3. On the **System Contents** tab, rename the instance to `custom_pattern_generator`:
   a. In the **Name** column, right-click on **custom_pattern_generator_0**, and select **Rename**.
   b. Remove the `_0` characters from the name.
4. Set the **custom_pattern_generator** clock domain to **clk_0**.
5. Connect the **custom_pattern_generator** `csr` interface to the **mm_bridge_0** `m0` interface:
   - In the **Connections**, column click to fill in the connection dot between the **custom_pattern_generator** `csr` interface and the **mm_bridge_0** `m0` interface.
   - Alternatively, you can right-click on `customer_pattern_generator.csr` interface, point to **customer_pattern_generator.csr Connections**, and **mm_bridge_0.m0**.
6. Connect the **custom_pattern_generator** `pattern_access` interfaces to the **mm_bridge_0** `m0` interface.

7. Assign the **custom_pattern_generator** `csr` interface to a base address of 0400:
   a. In the **Base** column, double-click on the address `0x00000000`.
   b. Enter `400` for the base address, which is in hexadecimal format.

   You assign a base address just higher than the end address of the `pattern_access` interface to avoid conflicting with the address space of the `pattern_access` interface

8. Keep the assignment of the **custom_pattern_generator** `pattern_access` interface to a base address of 0000.

Adding a PRBS Pattern Generator

The output of the PRBS pattern generator is a statically-defined PRBS pattern. You can specify the pattern length before the pattern repeats in the parameter editor. The pattern length is defined by 2^(data width) – 1. For example, a 32-bit PRBS pattern generator repeats the pattern after it sends 4,294,967,295 elements. You set the width of the PRBS generator based on the (local) data width of the memory on your board.

The PRBS pattern generator has two interfaces. The `csr` interface controls the behavior of the PRBS pattern generated. The `st_pattern_output` streaming source interface sends data to the pattern writer component. To add the PRBS pattern generator, follow these steps:

1. Double-click **PRBS Pattern Generator** from the **Memory Test Microcores** group. The parameter editor appears.
2. To accept the default parameters, click **Finish** in the parameter editor.
3. Rename the instance to `prbs_pattern_generator`.
4. Set the **prbs_pattern_generator** clock domain to **clk_0**.
5. Connect the **prbs_pattern_generator** `csr` interface to the **mm_bridge_0** `m0` interface.
6. Assign the **prbs_pattern_generator** `csr` interface to a base address of 0x0420 (which is a base address just higher than the end address of the **custom_pattern_generator** `csr` interface at base address 0x0400).

Adding a Two-to-One Streaming Multiplexer

Because the system has two pattern sources, and the pattern writer component accepts data only from one streaming source, you add a two-to-one streaming multiplexer between the pattern generators and the pattern writer. The two-to-one streaming soft programmable multiplexer IP core allows the processor to select which pattern to send to the pattern writer component. The component has the following interfaces:

- Two streaming inputs: `st_input_A` and `st_input_B`.
- One streaming output: `st_output`.
- One `csr` slave interface, which the processor controls to select whether input `A` or input `B` is sent to the streaming output.

The custom pattern generator connects to the `A` input; the PRBS pattern generator connects to the `B` input. To add the two-to-one streaming multiplexer, follow these steps:

1. Double-click **Two-to-one Streaming Mux** from the **Memory Test Microcores** group. The parameter editor appears.
2. To accept the default parameters, click **Finish** in the parameter editor.
3. Rename the instance to `two_to_one_st_mux`.
4. Set the **two_to_one_st_mux** clock domain to **clk_0**.
5. Connect the **two_to_one_st_mux** `st_input_A` interface to the **custom_pattern_generator** `st_pattern_output` interface.
6. Connect the **two_to_one_st_mux** `st_input_B` interface to the **prbs_pattern_generator** `st_pattern_output` interface.
7. Connect the **two_to_one_st_mux** `csr` interface to the **mm_bridge_0** `m0` interface.
8. Export the **two_to_one_st_mux** `st_output` interface with the name `st_data_out`.
9. Assign the **two_to_one_st_mux** `csr` interface to a base address of 0x0440 (which is a base address just higher than the end address of the **prbs_pattern_generator** `csr` interface at base address 0x0420).

Verifying the Memory Address Map

To ensure that the memory map of the system you create matches the memory map that other sections of the tutorial use, verify the base addresses in the system. Click the **Address Map** tab, and confirm the entries in your table match the values in Table 2–1. Any red exclamation marks indicate that the address ranges overlap. Correct the address maps to ensure there are no overlapping addresses, and your map matches this tutorial's guidelines.

Table 2–1. Address Map

Component	Address
custom_pattern_generator.csr	0x00000400 – 0x0000040f
custom_pattern_generator.pattern_access	0x00000000 – 0x000003ff
prbs_pattern_generator.csr	0x00000420 – 0x0000043f
two_to_one_st_mux.csr	0x00000440 – 0x00000447

Connecting the Reset Signals and Inserting Adapters

You must connect all the reset signals, which eliminates some of the error messages in the **Messages** tab. Qsys allows multiple reset domains, or one reset signal for the system. In this design, you want to connect all the reset signals with the incoming reset signal, so that you can use the Qsys autoconnect feature. To connect all the reset signals together, on the System menu, select **Create Global Reset Network**.

The remaining error messages on the **Messages** tab relate to the ready latency mismatches between the pattern generators and the multiplexers. To eliminate the mismatch between the streaming source and sink timing characteristics, on the System menu select **Insert Avalon-ST Adapters**, so that Qsys automatically inserts streaming timing adapters into the appropriate datapaths.

Qsys shows no remaining error or warning messages. If you have any error messages in the **Messages** tab, review the procedures to create this system to ensure you did not miss a step. You can view the reset connections and the timing adapters on the **System Contents** tab.

Save the system. On the File menu, click **Save**.

Now you have a system that includes the data pattern generator for the design (Figure 1–1 on page 1–3). The output of the two-to-one streaming multiplexer carries the pattern data from either the custom pattern generator or the PRBS pattern generator to the pattern writer in the full system. The data, from the output of the two-to-one streaming multiplexer, achieves a throughput of one word per clock cycle.

Creating the Data Pattern Checker

In this section you create the data pattern checker system, which is very similar to the data pattern generator system. The system reads back the pattern from the SDRAM and sends it to the pattern checker to verify it against the pattern from the data pattern generator. The pattern reader sends the data to a one-to-two streaming demultiplexer that routes the data to either the custom pattern checker or the PRBS pattern checker. The one-to-two streaming demultiplexer is soft programmable so that the processor can select which pattern checker IP core should verify the data that the pattern reader reads. The custom pattern checker is also soft programmable and is configured to match the same pattern as the custom pattern generator.

Creating a New Qsys System and Setting Up the Clock Source

To create a new Qsys system and set up the clock source, follow these steps:

1. On the File menu, click **New System**. Qsys opens and displays a new empty system. On the **System Contents** tab, Qsys shows a clock source instance, **clk_0**.
2. Double-click on the instance to edit the clock source settings.
3. Turn off **Clock frequency is known** to indicate that, when created, the higher-level hierarchical system that instantiates this subsystem provides the clock frequency.
4. Click **Finish**.
5. Save the pattern checker system:
    a. On the File menu, click **Save As**.
    b. Type file name `pattern_checker_system` and click **Save**.

Adding a Pipeline Bridge

To add a pipeline bridge that consolidates the slave interfaces, follow these steps:

1. On the **Component Library** tab, expand **Bridges and Adapters**, and then expand **Memory Mapped**.

2. Click **Avalon-MM Pipeline Bridge** component and click **Add**. The parameter editor opens.
3. In the parameter editor, for the **Address width** enter `11` to accommodate the span of the memory-mapped components in this system.
4. Click **Finish**. The default instance name is **mm_bridge_0**.
5. Set the **mm_bridge_0** clock domain to **clk_0**.
6. Export the **mm_bridge_0** `s0` interface with the name `slave`.

Adding a One-to-Two Streaming Demultiplexer

The one-to-two streaming demultiplexer performs the opposite operation of the two-to-one streaming multiplexer. It has a streaming input interface, `st_input`, that accepts data from the pattern reader and it has two streaming output interfaces, `st_output_A` and `st_output_B`, that connect to the custom pattern generator and PRBS pattern generator. To allow the processor to program the route the data takes through the component, the system includes a slave interface, `csr`. To add the one-to-two streaming demultiplexer, follow these steps:

1. Double-click **One-to-two Streaming Demux** from the **Memory Test Microcores** group. The parameter editor appears.
2. To accept the default parameters, click **Finish** in the parameter editor.
3. On the **System Contents** tab, rename the instance to `one_to_two_st_demux`.
4. Set the **one_to_two_st_demux** clock domain to **clk_0**.
5. Export the **one_to_two_st_demux** `st_input` interface with the name `st_data_in`.
6. Connect the **one_to_two_st_demux** `csr` interface to the **mm_bridge_0** `m0` interface.
7. Assign the **one_to_two_st_demux** `csr` interface to a base address of 0x0400.

Add a Custom Pattern Checker

The custom pattern checker performs the opposite operation of the custom pattern generator. It has a streaming input interface, `st_pattern_input`, that accepts data from the one-to-two streaming demultiplexer. It has an Avalon-MM slave interface, `csr`, that the processor uses to control the component. It also has a memory mapped slave interface, `pattern_access`, that the processor uses to program the same patterns as the custom pattern generator component. To add the custom pattern checker, follow these steps:

1. Double-click **Custom Pattern Checker** from the **Memory Test Microcores** group. The parameter editor appears.
2. To accept the default parameters, click **Finish** in the parameter editor.
3. Rename the instance to `custom_pattern_checker`.
4. Set the **custom_pattern_checker** clock domain to **clk_0**.
5. Connect the **custom_pattern_checker** `csr` and `pattern_access` interfaces to the **mm_bridge_0** `m0` interface.

6. Connect the **custom_pattern_checker** `st_pattern_input` interface to the **one_to_two_st_demux** `st_output_A` interface.
7. Assign the **custom_pattern_checker** `csr` interface to a base address of 0x0420.
8. Assign the **custom_pattern_checker** `pattern_access` interface to a base address of 0x0000.

Add the PRBS Pattern Checker

The PRBS pattern checker performs the opposite operation of the PRBS pattern generator. It has a memory mapped slave interface, `csr`, that the processor accesses to control the component. It also has a streaming input, `st_pattern_input`, that accepts data from the one-to-two streaming demultiplexer. To add the PRBS pattern checker, follow these steps:

1. Double-click **PRBS Pattern Checker** from the **Memory Test Microcores** group. The parameter editor appears.
2. To accept the default parameters, click **Finish** in the parameter editor.
3. Rename the instance to `prbs_pattern_checker`.
4. Set the **prbs_pattern_checker** clock domain to **clk_0**.
5. Connect the **prbs_pattern_checker** `csr` interface to the **mm_bridge_0** `m0` interface.
6. Connect the **prbs_pattern_checker** `st_pattern_input` interface to the **one_to_two_st_demux** `st_output_B` interface.
7. Assign the **prbs_pattern_checker** `csr` interface to a base address of 0x0440.

Verify the Memory Address Map

To ensure that the memory map of the system you create matches the memory map that other sections of the tutorial use, verify the base addresses in the system. Click the **Address Map** tab, and confirm the entries in your table match the values in Table 2–2.

Table 2–2. Address Map

Component	Address
one_to_two_st_demux.csr	0x00000400 - 0x00000407
custom_pattern_checker.csr	0x00000420 - 0x0000042f
custom_pattern_checker.pattern_access	0x00000000 - 0x000003ff
prbs_pattern_checker.csr	0x00000440 - 0x0000045f

Connecting the Reset Signals

You must connect all the reset signals. To connect all the reset signals together, on the System menu, select **Create Global Reset Network**.

Qsys shows no remaining error or warning messages. If you have any error messages in the **Messages** tab, review the procedures to create this system to ensure you did not miss a step. You can view the reset connections and the timing adapters on the **System Contents** tab.

Save the system. On the File menu, click **Save**.

Now you have a system that verifies the data it reads back from memory. The top-level system sends the data from the pattern reader component via a streaming interface. The data enters the one-to-two streaming demultiplexer, which then routes it to either the custom pattern checker or the PRBS pattern checker.

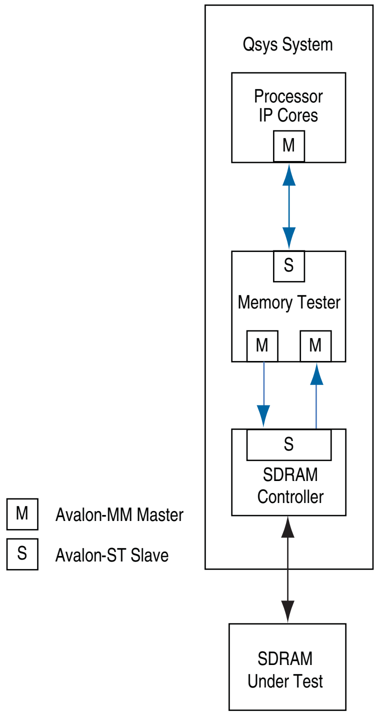
The lower-level subsystems for the memory tester design are complete. You can now move on to Chapter 3, Assembling Hierarchical Systems to use these systems in a hierarchical system design.

3. Assembling Hierarchical Systems

This tutorial describes hierarchical system design in Qsys. Hierarchical systems allow you to create reusable modular system components and also allow you to easily visualize large systems, by breaking large systems into smaller subsystems.

This tutorial uses the systems from Chapter 2, Creating Qsys Systems (or the completed versions of the systems provided with the design files) as hierarchical subsystems in a memory tester system You then instantiate the memory tester system in the top-level system, which also includes a processor system and an SDRAM controller. Figure 3–1 shows the high-level interfaces in the top-level system.

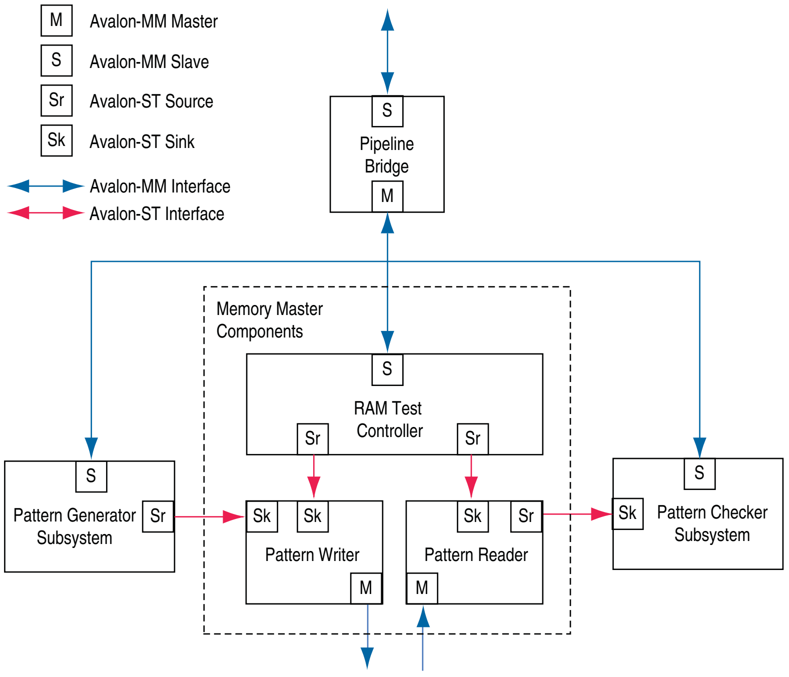
Figure 3–1. System High-level Interfaces



Creating the Hierarchical Memory Tester

Figure 3–2 shows the memory tester interfaces.

Figure 3–2. Hierarchical Memory Tester Interfaces



As Figure 3–2 shows, the memory tester includes the following Qsys subsystems, which you created in Chapter 2, Creating Qsys Systems:

- A data pattern generator—generates and transmits Avalon-ST data to the memory master components.
- A data pattern checker—receives and verifies Avalon-ST data from the memory master components.

If you skipped chapter 2, follow these steps to set up your Quartus II project:

1. Download and install the tutorial files (refer to "Downloading and Installing the Design Example Files" on page 1–3).
2. Copy the two completed systems (**pattern_checker_system.qsys**, and **patter_generator_system.qsys**) from the **tt_qsys_design\completed_subsystems** directory into the appropriate **tt_qsys_design\quartus_ii_projects_for_boards**\<*development_board*> directory for your board.
3. Open the Quartus II project for your development board (refer to "Opening the Tutorial Project" on page 1–4).
4. In the Quartus II software, on the Tools menu, click **Qsys**.

To create the memory tester, follow these steps:

1. In Qsys, on the File menu, click **New System**. Qsys opens and displays a new empty system. In the **System Contents** tab, Qsys shows a clock source instance, **clk_0**.
2. To open the clock source settings, right-click **clk_0** and click **Edit** or double-click on the instance.
3. Turn off **Clock frequency is known** to indicate that, when created, the higher-level hierarchical system that instantiates this subsystem provides the clock frequency.
4. Click **Finish**.
5. Save the system:
   a. On the File menu, click **Save As**.
   b. Type file name `memory_tester_system` and click **Save**.

The memory tester includes several Avalon-MM slave interfaces. However, the memory tester groups them behind an Avalon-MM pipeline bridge that exports a single Avalon-MM slave interface to the top-level system. This technique allows the top-level system to access all the memory-mapped slave ports by reading and writing to a single pipeline bridge slave interface. The bridge also adds a level of pipelining, which can improve timing performance. To add the pipeline bridge, follow these steps:

1. On the **Component Library** tab, expand **Bridges and Adapters**, and then expand **Memory Mapped**. Alternatively, you can type `bridge` in the search box to filter the list and show only the bridge components. You should click **X** next to the search box, to clear the search filtering.
2. Click **Avalon-MM Pipeline Bridge** component and click **Add**. Alternatively, you can double-click on **Avalon-MM Pipeline Bridge**. The parameter editor opens.
3. In the parameter editor, for the **Address width** enter `13` to accommodate the span of the memory-mapped components in this system.
4. Click **Finish**. The default instance name is **mm_bridge_0**.
5. On the **System Contents** tab, set the **mm_bridge_0** clock domain to **clk_0**:
   - In the **Clock** column for the **mm_bridge_0** `clk` interface, select **clk_0** from the drop-down list.
   - Alternatively, you can make the connection in the **Connections** column. Click to fill in the connection dot between the **clk_0** `clk` output and the **mm_bridge_0** `clk` input.
   - Alternatively, you can right-click on **mm_bridge_0** `clk` input, point to **mm_bridge_0.clk Connections**, and select **clk_0.clk**.
6. Export the **mm_bridge_0** `s0` interface with the name slave: click in the **Export** column and type `slave`.

Adding the Data Pattern Generator

The data pattern generator system from Chapter 2, Creating Qsys Systems provides a stream of pattern data via an Avalon-ST source interface. You control the system by accessing the memory locations allocated to each component within the subsystem. The system connect all slave ports to a pipeline bridge, which it then exposes outside of the system. The system contains the following components:

- Pipeline bridge
- Custom pattern generator
- PRBS pattern generator
- Two-to-one streaming multiplexer
- Streaming timing adapters

To add the data pattern generator to the memory tester, follow these steps:

1. On the **Component Library** tab, under **Project** expand **System**, double-click **pattern_generator_system**. The parameter editor appears.
2. Click **Finish**.
3. Rename the instance to `pattern_generator_subsystem`.
   a. In the **Name** column, right-click on **system_0**, and select **Rename**.
   b. Enter `pattern_generator_subsystem` for the instance name.
4. Set the **pattern_generator_subsystem** clock domain to **clk_0**.
5. Connect the **pattern_generator_subsystem** `slave` interface to the **mm_bridge_0** `m0` interface.
6. Connect the **pattern_generator_subsystem** `reset` interface to the **clk_0** `clk_reset` interface.

Because the `reset` interface is exported to reset_0, you cannot right click on the interface to make the connection. You must make the connection in the **Connections** column. Click to fill in the connection dot between **pattern_generator_subsystem** `reset` interface to the **clk_0** `clk_reset` interface.

Adding the Pattern Checker

The pattern checker system from Chapter 2, Creating Qsys Systems validates data that arrives via an Avalon-ST sink interface. You control the system by accessing the memory locations allocated to each component within the subsystem. The system connect all of the slave ports to a pipeline bridge, which it then exposes outside of the system. The system contains the following components:

- Pipeline bridge
- Custom pattern checker
- PRBS pattern checker
- One-to-two demultiplexer

To add the data pattern checker to the memory tester, follow these steps:

1. Double-click **pattern_checker_system** from the **System** group.
2. Click **Finish**.
3. Rename the instance to `pattern_checker_subsystem`.
4. Set the **pattern_checker_subsystem** clock domain to **clk_0**.
5. Connect the **pattern_checker_subsystem** `slave` interface to the **mm_bridge_0** `m0` interface.
6. Connect the **pattern_checker_subsystem** `reset` interface to the **clk_0** `clk_reset` interface.

Adding the Memory Master Components

In this section you add the memory masters and the RAM test controller. Memory masters access the SDRAM controller by writing the test pattern to the memory and reading the pattern back for validation. The RAM test controller accepts commands from the processor and controls the memory masters. Each command contains information such as a start address, test length in bytes, and memory block size in bytes. The RAM test controller segments the commands into smaller block transfers and issues them to the read and write masters independently via streaming connections.

When the pattern reader or writer components complete a block transfer, they signal to the RAM test controller that they are ready for another command. The RAM test controller issues the block-sized commands independently, which minimizes the number of idle cycles between memory transfers. The RAM test controller also ensures that the pattern reader never overtakes the pattern writer with respect to the memory locations it is testing, otherwise data corruption occurs.

The SDRAM controller in this design is parameterized to use a local maximum burst length of 2. The pattern reader and writer components are also configured to match this burst length to maximize the memory bandwidth.

Adding a Pattern Writer Component

The pattern writer component accepts memory transfer commands from the RAM test controller with the `command` streaming interface. The `st_data` streaming interface accepts data provided by the design's pattern generator. The `mm_data` memory-mapped interface writes the pattern data to the SDRAM controller. To add the pattern writer component to the system, follow these steps:

1. Double-click **Pattern Writer** core from the **Memory Test Microcores** group. The parameter editor appears.
2. Turn on **Burst Enable** support.
3. Ensure the **Maximum Burst Count** is **2**.
4. Make sure that **Enable Burst Re-alignment** is turned on.
5. To accept the other default parameters, click **Finish** in the parameter editor.
6. Rename the instance to `pattern_writer`.
7. Set the **pattern_writer** clock domain to **clk_0**.

8. Connect the **pattern_writer** st_data interface to the **pattern_generator_subsystem** st_data_out interface.
9. Export the **pattern_writer** mm_data interface with the name write_master.

Adding a Pattern Reader Component

The pattern reader component accepts memory transfer commands from the RAM test controller with the command streaming interface. The mm_data Avalon-MM interface reads the pattern data from the SDRAM controller. The st_data Avalon-ST interface sends the data read from memory to the design's pattern checker. To add the pattern reader component to the system, follow these steps:

1. Double-click **Pattern Reader** core from the **Memory Test Microcores** group. The parameter editor appears.
2. Turn on **Burst Enable** support.
3. Ensure the **Maximum Burst Count** is **2**.
4. Make sure that **Enable Burst Re-alignment** is turned on.
5. To accept the other default parameters, click **Finish** in the parameter editor.
6. Rename the instance to pattern_reader.
7. Set the **pattern_reader** clock domain to **clk_0.**
8. Connect the **pattern_reader** st_data interface to the **pattern_checker_subsystem** st_data_in interface.
9. Export the **pattern_reader** mm_data interface with the name read_master.

Adding a RAM Test Controller

The RAM test controller contains two interfaces that send commands to the pattern reader and writer components. The two streaming command interfaces are write_command and read_command. These streaming interfaces issue commands effectively because Avalon-ST interfaces offer low latency and a simple handshaking protocol. Also the processor accesses a slave port, csr, to write commands to the controller.

To add the RAM test controller to the system, follow these steps:

1. Double-click **RAM Test Controller** from the **Memory Test Microcores** group. The parameter editor appears.
2. To accept the default parameters, click **Finish** in the parameter editor.
3. On the **System Contents** tab, rename the instance to ram_test_controller**.**
4. Set the **ram_test_controller** clock domain to **clk_0**.
5. Connect the **ram_test_controller** write_command interface to the **pattern_writer** command interface.
6. Connect the **ram_test_controller** read_command interface to the **pattern_reader** command interface.
7. Connect the **ram_test_controller** csr interface to the **mm_bridge_0** m0 interface.

Do not use the **Generation** tab at this point in the tutorial to generate HDL code for these subsystems, because you must only generate files for the entire top-level system, which includes all the subsystems. The batch script provided for you to program the device requires that only one system is generated in the project directory. The top-level design includes a Nios II subsystem, and the Nios II software build tools require the **.sopcinfo** file to be generated for the top-level design. If there are multiple **.sopcinfo** files, the batch script to program the device fails with an error from the software build tools.

Connecting the Reset Signals

You must connect all the reset signals. On the System menu, select **Create Global Reset Network**.

Specifying the Memory Address Map

In this section, you use the **Address Map** tab to set the addresses in the memory map of the system to ensure it matches the memory map that other sections of the tutorial use. To set the base addresses, follow these steps:

1. Click the **Address Map** tab. Red exclamation marks indicate overlapping addresses, because all of the slave address maps currently start at address 0x0.
2. Double-click next to each interface in the **mm_bridge_0.m0** column, so that Qsys replaces the address range in the cell with a base address that you can edit. Delete the current address, refer to Table 3–1, and type the correct base address.

Table 3–1. Memory Tester Address Map

Component Name	Base Address	Resulting Address Range
mm_bridge_0.s0	N/A	N/A
pattern_generator_subsystem.slave	0x0	0x00000000 – 0x000007ff
pattern_checker_subsystem.slave	0x1000	0x0001000 – 0x000017ff
ram_test_controller.scsr	0x800	0x00000800 – 0x0000081f

3. When you click out of the cell, Qsys displays the resulting address range. Confirm the resulting address range in your table matches the values in Table 3–1.

There are now no remaining error or warning messages. Save the memory tester system.

Completing the Top-Level System

This section describes how to complete top-level system. To add the memory tester system to complete the top-level system, follow these steps:

1. In Qsys, open the **top_system.qsys** file from the **tt_qsys_design\quartus_ii_projects_for_boards\**<*development_board*> directory.
2. This system is set up for your development board, with an external clock source, a processor system, and an SDRAM controller. You can view the clocks in this top-level system on the **Clock Settings** tab. You can view the partially-completed system connections in the **System Contents** tab.

3. Double-click **memory_tester_system** from the **System** group.
4. Click **Finish**. The memory tester system is added to the top-level system.
5. On the **System Contents** tab, rename the system to `memory_tester_subsystem`.
6. On the **System Contents** tab move the **memory_test_subsystem** up between the **cpu_subsystem** and the **sdram**. The **cpu_subsystem** controls the **memory_tester_subsystem**, which controls the **sdram**, so performing this move helps you to visualize the system easier. Select the **memory_test_subsystem** and click the up arrow once.
7. Set the **memory_tester_subsystem** clock domain to:
   - **sdram_sysclk** for ALTMEMPHY-based designs
   - **sdram_afi_clk** for UniPHY-based designs

   Some boards have an FPGA and SDRAM device that use either the Altera DDR or DDR2 SDRAM Controller with ALTMEMPHY; others use the Altera DDR3 SDRAM controller with UniPHY.

8. Connect the **memory_tester_subsystem** `reset` interface to the **ext_clk** `clk_reset` interface and the **cpu_subsystem** `cpu_jtag_debug_reset` interface.

This design exports the Nios II processor JTAG debug reset output interface, `jtag_debug_module_reset`, from the cpu_subsystem with the interface name `cpu_jtag_debug_reset`. The design must connect this Nios II reset output to any component reset inputs that require resetting by the Nios II processor code or JTAG interface, and also to the Nios II processor's reset input interface. The cpu_subsystem `cpu_reset` interface connects to the Nios II processor's reset input interface. The **top_level.qsys** file connects the `cpu_jtag_debug_reset` interface to the `cpu_reset` interface.

9. Connect the **memory_tester_subsystem** `write_master` and `read_master` interfaces to the:
   - **sdram** `s1` interface for ALTMEMPHY-based designs
   - **sdram** `avl` interface for UniPHY-based designs
10. Connect the **memory_tester_subsystem** `slave` interface to the **cpu_subsystem** `master` interface.
11. Keep the base addresses to 0x0 for the **memory_tester_subsystem** `slave` interface and the:
    - **sdram** `s1` interface for ALTMEMPHY-based designs
    - **sdram** `avl` interface for UniPHY-based designs.

    The two slave interfaces can use the same address map range because different masters control them. The **cpu_subsystem** `master` interface controls the memory test subsystem, and the **memory_tester_subsystem** `write_master` and `read_master` interfaces control the `sdram` interface.

The design is complete. If you have any error messages in the **Messages** tab, review the procedures to create this system to ensure you did not miss a step.

Save the system. Optionally, you can click on the following tabs:

- The **System Inspector** tab shows the hierarchical system. The default view shows the project settings for the top-level system. Expand **Submodules** to view information about the lower-level systems and components.
- The **HDL Example** tab shows the input and output signals of the Qsys system. This tab displays the HDL for an example instantiation of this system in an HDL file, and lists all the signals from the exported interfaces in the system. The signal names are the exported interface name followed by an underscore and then the signal name specified in the component or IP core. In this case, most of the signals connect to the external SDRAM device under test.
- The **Generation** tab allows you to generate design files from Qsys. You can also compile the hardware design in the Quartus II software after the generation is complete and download the image and software to the board ("Compiling and Downloading Software to a Development Board" on page 3–9).

Compiling and Downloading Software to a Development Board

Altera recommends you download the system and software to a development board to complete the design process and test the memory interface of the board. If you do not have a development board you can follow the steps provided in the accompanying **readme.txt** file to learn more details about porting designs to other FPGA devices or boards.

The Altera-provided software tests the memory using various test parameters and patterns. The software is scripted for compilation and download to the board. To download the top-level system to a development board, follow these steps:

1. In Qsys, click the **Generation** tab.
2. Ensure that you turn on **Create HDL design files for synthesis**. This step does not require any of the other options.
3. Click **Generate**. Qsys generates HDL files for the system and the Quartus IP File (**.qip**) that provides the list of required HDL files for the Quartus II compilation.
4. When Qsys completes the generation, click **Close**. You can also close Qsys.
5. In the Quartus II software, on the Project menu, click **Add/Remove Files in Project** and verify that project includes the **.qip** file for this system. The project also includes the top-level Verilog HDL wrapper file **top_level.v** that instantiates the **top_system.qsys** system, and the SDC timing constraint file **my_constraints.sdc**.
6. On the Processing menu, click **Start Compilation**.
7. Connect the development board to a supported programming cable.
8. Open a Linux or Nios II command shell. In Qsys, on the Tools menu, click **Nios II Command Shell [gcc4]**. Alternatively, on Windows, open the Nios II Command Shell. On the Start menu, point to **Altera**, point to **Nios II EDS**, and click **Nios II EDS Command Shell**. For Linux, open a Linux terminal.
9. Navigate to the **quartus_ii_projects_for_boards\**<*development_board*>**\\software** directory.
10. When compilation completes, click **OK**.

11. Type the following command at the Nios II command Shell or Linux terminal

```
./batch_script.sh.
```

The batch script compiles the Nios II software and downloads the **.sof** programming file into the FPGA.

After the script configures the FPGA, it downloads the compiled Nios II software to the board and establishes a terminal connection with the board. The test software performs test sweeps on the SDRAM by varying the following parameters:

- Pattern type
- Memory block size
- Memory block trail distance (number of blocks by which the pattern reader trails the pattern writer)
- Memory span tested

☞ Ensure that you do not have multiple sets of generated system files in the project directory, which produce multiple **.sopcinfo** files, otherwise the batch script to program the device fails with an error from the software build tools.

The memory throughput values appear in the command terminal as the memory is tested. These values are reported in hexadecimal and represent the number of clock cycles required to test the entire SDRAM address span. The output is restricted to hexadecimal due to a very small software library that prints the characters to the terminal. Because the memory tester system writes to the memory and then reads it back, the number of bytes it accesses and reports in the transcript window is double the memory span. This number varies depending on the span of memory being tested for your memory device. Knowing the data width of the memory interface, the number of bytes transferred, and the number of clock cycles for the transfer, you can determine the memory access efficiency (Equation 3–1).

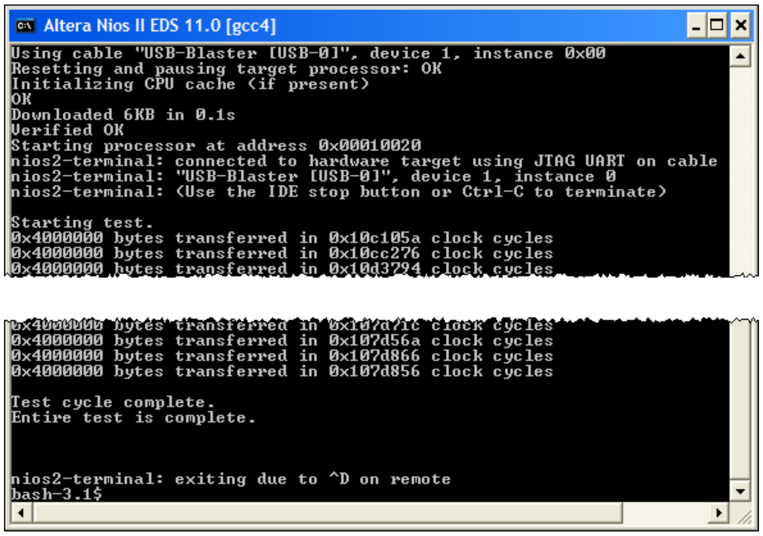
☞ The SDRAM controller in the top-level Qsys system has a 32-bit local interface width, therefore *memory data width in bytes* in Equation 3–1 is 4 bytes for the tutorial design.

Equation 3–1. Equation to Calculate Memory Efficiency

$$\text{Efficiency} = 100 \times \textit{total bytes transferred} / (\textit{memory data width in bytes} \times \textit{total clock cycles})$$

Figure 3–3 shows example output in the command window.

Figure 3–3. Sample Output from Terminal



```
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 6KB in 0.1s
Verified OK
Starting processor at address 0x00010020
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Starting test.
0x4000000 bytes transferred in 0x10c105a clock cycles
0x4000000 bytes transferred in 0x10cc276 clock cycles
0x4000000 bytes transferred in 0x10d3794 clock cycles

0x4000000 bytes transferred in 0x107d71c clock cycles
0x4000000 bytes transferred in 0x107d56a clock cycles
0x4000000 bytes transferred in 0x107d866 clock cycles
0x4000000 bytes transferred in 0x107d856 clock cycles

Test cycle complete.
Entire test is complete.


nios2-terminal: exiting due to ^D on remote
bash-3.1$
```

The test runs until the design finishes testing the full memory. To end the test early, type Ctrl+C in the command window.

To calculate the efficiency for the last throughput numbers in Figure 3–3, convert the hexadecimal numbers to decimal. Thus:

0x4000000 bytes transferred is 0d67108864 total bytes transferred.

0x107d856 clock cycles is 0d17291350 total clock cycles.

Therefore the efficiency for this example is:

$100 \times 67108864 \;/\; (4 \times 17291350) = 97.0\%$.

If the memory test starts but does not complete successfully, the terminal displays failure messages. If you see failure messages from the memory test, review the previous sections and check that you have completed all the instructions in this tutorial successfully. A missed connection or incorrect memory address assignment may cause the tester design to fail on the board. Altera provide completed systems, so that you can check your results. You can copy the completed systems into the project directory with a different name, so that you can open them side-by-side with your systems for comparison. Alternatively, you can replace your systems with the provided completed systems to run the memory tester design successfully. The completed systems are in the following directories:

- **tt_qsys_design\completed_subsystems\pattern_checker_system.qsys**
- **tt_qsys_design\completed_subsystems\pattern_generator_system.qsys**
- **tt_qsys_design\completed_subsystems\completed_memory_tester_system \memory_tester_system.qsys**
- **\tt_qsys_design\quartus_ii_projects_for_boards\**<*development_board*> **\backup_and_completed_top_system\completed_top_system\top_system.qsys**

4. Verifying Hardware in System Console

This tutorial shows you how to use System Console available in the Quartus II software to verify your system design. The design example files include scripts that exercise your system via System Console Tcl commands. The system is similar to the system in Chapter 3, Assembling Hierarchical Systems; however, this system uses a JTAG-to-Avalon Master Bridge component to drive all the slave components, instead of a Nios II processor system.

Understanding the Scripts

The **\quartus_ii_projects_for_boards**<*development_board*>**\system_console** directory contains the **run_sweep.tcl**, **base_address.tcl**, and **test_cases.tcl** scripts. You use these scripts to set up and run the memory tests on the various provided development board projects. You can view the scripts to help you understand the System Console commands that drive the slave component registers. The scripts work with any board, if you keep the same Qsys system structure.

The **run_sweep.tcl** file is the main script, which calls the other two scripts. The **base_address.tcl** file includes all the information about the base addresses of the slave components that the previous chapters use. If you change the base addresses of the slave components, you must also change them in the **base_address.tcl** file. The **test_cases.tcl** file includes settings for memory span, memory block sizes, and memory block trails.

The **run_sweep.tcl** file contains System Console Tcl commands for the following actions:

- Initialize the components
- Adjust test parameters
- Start the PRBS pattern checker, PRBS pattern generator, and RAM controller
- Continuously poll the stop and fail bits in the PRBS checker

Opening the Tutorial Project

If you did not complete Chapter 3, Assembling Hierarchical Systems, to set up your Quartus II project, follow these steps:

1. Download and install the tutorial design files (refer to “Downloading and Installing the Design Example Files” on page 1–3).

2. Copy the following completed systems to the appropriate **tt_qsys_design\quartus_ii_projects_for_boards\**<*development_board*> directory for your board:
   - The two completed systems **pattern_checker_system.qsys**, and **patter_generator_system.qsys** from the **tt_qsys_design\completed_subsystems** directory.
   - The completed system **memory_tester_system.qsys** from the **tt_qsys_design\completed_subsystems\completed_memory_tester_system** directory.
   - The completed top-level system **top-system.qsys** from the **tt_qsys_design\quartus_ii_projects_for_boards\**<*development_board*>**\backup_and_completed_top_system\completed_top_system** directory.

   You can learn how to build these systems in Chapter 2, Creating Qsys Systems and Chapter 3, Assembling Hierarchical Systems.

3. Open the Quartus II project for your development board (refer to "Opening the Tutorial Project" on page 1–4).

To open the top-level Qsys file, follow these steps:

1. On the Tools menu, click **Qsys**.
2. On the Qsys File menu, open the **top_system.qsys** file in the project directory.

Adding the JTAG-to-Avalon Master Bridge

The JTAG-to-Avalon master bridge acts as a bridge between the JTAG interface and the system's memory tester. To add this bridge to the top-level system, follow these steps:

1. On the **Component Library** tab, expand **Bridges and Adapters**, and then expand **Memory Mapped**.
2. Click **JTAG to Avalon Master Bridge** component and click **Add**. The parameter editor opens.
3. To accept the default parameters, click **Finish** in the parameter editor.
4. On the **System Contents** tab, rename the instance to `jtag_to_avalon_bridge`.
5. Connect the **jtag_to_avalon_bridge** `master` interface to the **memory_tester_subsystem** `slave` interface.
6. Set the **jtag_to_avalon_bridge** clock domain to **sdram_sysclk**.
7. Connect the **jtag_avalon_bridge** `clk_reset` interface to the **ext_clk** `clk_reset` interface and the:
   - **sdram** `reset_request_n` interface for ALTMEMPHY-based designs
   - **sdram** `afi_reset` interface for UniPHY-based designs

8. Connect the **jtag_avalon_bridge** `master_reset` interface to the **memory_tester_subsystem** `reset` interface and the:
   - **sdram** `soft_reset_n` interface for ALTMEMPHY-based designs
   - **sdram** `soft_reset` interface for UniPHY-based designs
9. Disable the **cpu_subsystem** to remove it from the system, because you are replacing its function with the bridge and System Console. In the **Use** column, turn off **Use**.
10. On the File menu, click **Save**.

Compiling and Using System Console with a Development Board

The design example scripts test the memory in loops for different block sizes, that is, the number of bytes to group together in a single instance of back-to-back reads or writes. The scripts also test the memory in loops for different memory block trails, that is, the number of blocks by which the pattern reader trails the pattern writer. To download the programming file to your development board, follow these steps:

1. In Qsys, click the **Generation** tab.
2. Ensure that you turn on **Create HDL design files for synthesis**. This step does not require any of the other options.
3. Click **Generate**. Qsys generates HDL files for the system and the Quartus IP File (**.qip**) that provides the list of required HDL files for the Quartus II compilation.
4. When Qsys completes the generation, click **Close**.
5. In the Quartus II software, on the Project menu, click **Add/Remove Files in Project**, and verify that the project contains the **top_system.qip** file for this system.
6. On the Processing menu, click **Start Compilation**.
7. Connect the development board to a supported programming cable.
8. On the Tools menu, click **Programmer**.
9. Check that the Programmer displays the correct programming hardware. Otherwise, click **Hardware Setup** and select the correct programming hardware, then click **Close**.
10. When the Quartus II software competes the compilation, click **OK**.
11. Program the device, by clicking **Start**.
12. In Qsys, on the Tools menu, click **System Console**.
13. Before you execute any scripts in System Console, you must be in the directory that has the Tcl scripts. In the Tcl Console window type the following command to change the directory:

    ```
    cd system_console
    ```
14. On the File menu, click **Execute Script**.

15. To start the memory tests, run the **run_sweep.tcl** file from the **tt_qsys_design\quartus_ii_projects_for_boards\**<*development_board*> **\system_console** directory. Table 4–1 shows the tasks that the **run_sweep.tcl** script runs via System Console.

Table 4–1. run_sweep.tcl Script Tasks

Script Task	Description
Initialize components	1. Switches the multiplexer and demultiplexer over to input/output `B` by accessing the `csr` slave interface. 2. Writes to the PRBS generator core and PRBS check core `csr` slave interface and performs the following tasks: a. Disables the infinite payload size. b. Writes a payload size equal to the memory span of the design. 3. Writes the following to the RAM test controller `csr` slave interface: a. RAM base address for the start of the test. b. Transfer length equal to the memory span of the design. c. Concurrent read/write enable, which determines if the read and write accesses to the memory blocks should be concurrent. Disabling this setting provides the highest throughput but does not test switching between reads and writes.
Adjust test parameters	■ Writes various block sizes within a loop. ■ Writes various block trails within a loop.
Start the test	1. Writes the start bit to the PRBS generator core `csr` slave interface. 2. Writes the start bit to the PRBS checker core `csr` slave interface. 3. Writes to the start bit of the RAM controller `csr` interface.
Continuously polls the stop and fail bits of PRBS checker	1. Polling the PRBS checker core `csr` slave interface to determine if the failure bit is enabled and polling the run bit to determine when the test is complete. 2. Looping back for different memory block trail and memory block size when the test is complete.

After running the **run_sweep.tcl** script, the System Console displays the progress of the tests in the Messages window. The tests perform test sweeps on the SDRAM by varying the memory block size and memory block trail distance. When the tests finish correctly, the Tcl console displays the following message:

```
.... All tests have finished without any Failures ...
```

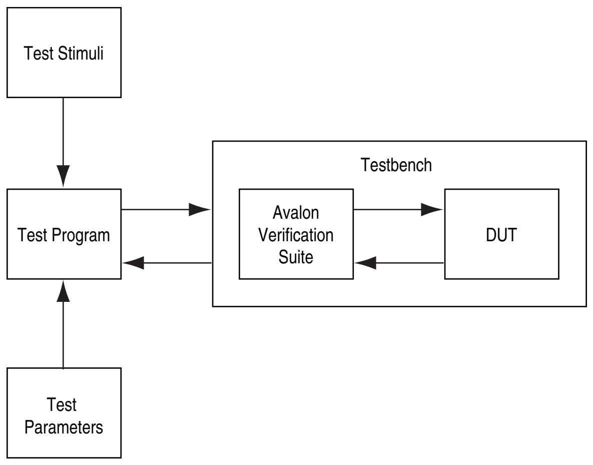
5. Simulating Custom Components

This chapter shows you how to verify a custom component with Qsys and the Avalon Verification IP Suite. You use Qsys to generate a testbench system for the design under test and perform a functional simulation with the ModelSim simulator. The Qsys-generated testbench uses the Avalon Verification IP Suite components.

For more information about the Avalon Verification IP Suite, refer to the *Avalon Verification IP Suite User Guide*.

Figure 5–1 shows the block diagram of a typical test environment.

Figure 5–1. Typical Test Environment Block Diagram



Generating a Testbench System in Qsys

In this section you generate a testbench system in Qsys for the design under test. The design under test in this chapter is the custom pattern generator from the other tutorial chapters. This component generates high-speed streaming data for testing memory devices. The soft-programmable custom pattern generator can generate multiple test patterns. The component is programmed with the pattern data and pattern length. When the end of the pattern is reached, the custom pattern generator cycles back to the first element of the pattern.

For your own designs, if you do not want to use the Qsys-generated testbench system, you can create your own Qsys testbench system by adding the Avalon Verification Suite BFMs or your own models for simulation. You can also generate a Qsys simulation model for the design or Qsys system under test, and use your own custom HDL testbench to provide the simulation stimulus.

Opening the Tutorial Project

The tutorial design example files include a Quartus II project to set up the working environment. To open the project, follow these steps:

1. Download and install the tutorial design files, as described in Chapter 1, Introduction.
2. In the Quartus II software, open the Quartus II Project File (**.qpf**), **qsys_sim_tutorial.qpf**, from the **\simulation_tutorial** directory.

Creating a New Qsys System for the Design Under Test

To create a new Qsys system for the design under test, follow these steps:

1. In the Quartus II software, on the File menu, click **New**.
2. Select **Qsys System File** and click **OK**.
3. Click **Close** on the **Initializing Complete** window.
4. The new system instantiates a clock source. However, the system does not need a clock source, so remove it. Click on the **clk_0** instance and click the **X** icon, or right-click and select **Remove**.
5. On the **Component Library** tab, expand **Memory Test Microcores**.
6. Select **Custom Pattern Generator** and click **Add**.
7. Click **Finish**.
8. Rename the instance to `pg`:
   a. In the **Module** column, right-click on **custom_pattern_generator_0**, and select **Rename**.
   b. Enter `pg` for the instance name.

Exporting All Design Under Test Interfaces

To export all design under test interfaces, follow these steps:

1. On the **System Contents** tab, in the **Export** column, for each interface click **Click to export**. Keep the default export names.
2. Save and name the system:
   a. On the File menu, click **Save As**.
   b. Type file name `pattern_generator` and click **Save**.

Generating a Qsys Testbench System

To generate a testbench system for the design under test, follow these steps:

1. Click the **Generation** tab.
2. Under **Simulation**, for **Create testbench Qsys system**, select **Standard, BFMs for standard Avalon interfaces**.
3. Under **Synthesis**, turn off **Create HDL design files for synthesis** and turn off **Create block symbol file (.bsf)**.
4. Click **Generate**.
5. After Qsys generates the testbench, click **Close** on the message window.

☞ Qsys generates this testbench system in the **\simulation_tutorial\pattern_generator\testbench** directory.

You can generate the simulation model for the Qsys testbench system at the same time by turning on **Create testbench simulation model**. However, the Qsys-generated testbench system's components names are assigned automatically and you may want to control the instance names to make it easier to run the test program for the BFMs. In this tutorial, you edit the Qsys testbench system before generating the simulation model.

Generate Qsys Testbench System's Simulation Models

In this section, you open the generated Qsys testbench system and rename the BFM component instance names to ensure the testbench names match the test program provided with the tutorial design files. You also generate the testbench's simulation model. To rename the BFMs and generate the simulation models, follow these steps:

1. In Qsys, open the testbench system, **pattern_generator_tb.qsys**, from the **simulation_tutorial\pattern_generator\testbench** directory.
2. On the **System Contents** tab, rename the instance as in Table 5–1.

Table 5–1. New Instance Names

Qsys-Generated Components' Names	New Instance Name
pattern_generator_inst	DUT
pattern_generator_inst_pg_clock_bfm	clock_source
pattern_generator_inst_pg_clock_reset_bfm	reset_source
pattern_generator_inst_pg_csr_bfm	csr_master
pattern_generator_inst_pg_pattern_access	pattern_master
pattern_generator_inst_pg_pattern_output_bfm	pattern_sink

3. Double-click one of the BFM components to open the parameter editor and view its settings. These components are BFMs available in the Avalon Verification Suite group in the Component Library. If necessary, you can change the settings for these BFMs, to ensure adequate test coverage for your design, in the parameter editor.

   ☞ The Qsys-generated testbench matches inserted BFMs with the exported interfaces from the design that they drive. The test program that provides stimulus to the BFMs must account for the matching interface. For example, an exported Avalon-MM slave interface (which expects word-aligned addresses) is connected to an Avalon master BFM, which expects and transacts word-aligned addresses instead of the byte or symbol addresses that are default for Avalon masters.

4. Click **Cancel** to close the parameter editor without making changes.
5. On the **Generation** tab, under **Simulation**, for **Create simulation model**, select **Verilog**.
6. For **Create testbench Qsys system** and **Create testbench simulation model**, select **None**.

7. Under **Synthesis** section, turn off all options.
8. Save the system.
9. Click **Generate**.
10. After Qsys generates the testbench, click **Close** on the message window.

☞ Qsys generates the testbench system's simulation models in the **\simulation_tutorial\pattern_generator\testbench\pattern_generator_tb\simulation** directory.

Qsys generates the simulation models and a ModelSim simulation script (**msim_setup.tcl**), which compiles the required files for simulation and sets up commands to load the simulation in the ModelSim simulator. You can run this ModelSim script in ModelSim-Altera to compile, elaborate, or load everything for simulation. In this tutorial, there is an external test program to provide simulation stimulus. The tutorial design files include a simulation script, **load_sim.tcl** that compiles the top-level simulation file and test program, and calls the Qsys-generated script to compile the required files.

Running Simulation In the ModelSim-Altera Software

In this section you run a simulation in the ModelSim-Altera software on the testbench that you created. To complete this simulation you use the test program provided in the design files. This test program performs the following actions:

- Reads a pattern file
- Writes the pattern to the design under test via the pattern master BFM
- Sets various design under test options via the CSR master BFM
- Starts the design under test pattern generation
- Collects data generated by the design under test
- Compares the results against the original pattern file

The test starts by writing a walking ones pattern to the design under test.

Setting Up the Simulation Environment

This tutorial includes test program files that you can use with the Qsys-generated testbench and ModelSim simulation script. To learn more about Qsys simulation support, open and review the simulation script, **\simulation_tutorial\load_sim.tcl**.

The **load_sim.tcl** script sets simulation variables to set up the correct hierarchical paths in the Qsys-generated simulation model and ModelSim script. In addition the script identifies the top-level instance name for the simulation and provides the path to the location of the Qsys-generated files. Some functions, such as memory initialization, rely on correct hierarchical paths names in the simulation model. The script then performs the following actions:

- Sources the Qsys-generated ModelSim simulation script, **msim_setup.tcl**
- Uses the command aliases defined in the **msim_setup.tcl** script to compile and elaborate the files for the Qsys testbench simulation model

- Compiles and elaborates the extra simulation files for the tutorial—the test program and top-level simulation file that instantiates the test program
- Loads the **wave.do** file that provides signals for the ModelSim waveform view

Close the **load_sim.tcl** script without making any changes.

Running the Simulation

To run the simulation, follow these steps:

1. Start the ModelSim-Altera software.
2. On the File menu, click **Change Directory**, browse to the **\simulation_tutorial** directory, and click **OK**.
3. On the Compile menu, click **Compile Options**.
4. Click the **Verilog & SystemVerilog** tab, select **Use SystemVerilog**, and click **OK**.
5. On the File menu, click **Load**.

   Ensure you activate the ModelSim-Altera Transcript window, otherwise the **Load** function is disabled.

6. Select the **load_sim.tcl** script and click **Open**.

   The warning messages relate to unused connections in an ALTSYNCRAM megafunction. Because these ports are not used, you can ignore the warning messages.

7. Run the simulation for 40 µs. To run the simulation, in the ModelSim-Altera Transcript window type the following command:

   ```
   run 40us↵
   ```

   You can run the h command to show the available options for the **msim_setup.tcl** script.

8. Observe the results. Example 5–1 shows the messages that you should see.

Example 5–1. Simulation Results

```
INFO: top.tb.reset_source.reset_deassert: Reset deasserted
INFO: top.pgm: Starting test walking_ones.hex
INFO: top.pgm.read_file: Read file walking_ones.hex success
INFO: top.pgm.read_file: Read file walking_ones_rev.hex success
INFO: top.pgm: Test walking_ones.hex passed
```

9. To run the low frequency test, modify **\simulation_tutorial\test_include.svh** (Table 5–2).

Table 5–2. Values for Low Frequency Pattern Test

Macro	New Value
PATTERN_POSITION	0
NUM_OF_PATTERN	2
NUM_OF_PAYLOAD_BYTES	256
FILENAME	low_freq.hex
FILENAME_REV	low_freq_rev.hex

10. Reload the **load_sim.tcl** script, run the simulation for 40 µs, and observe the result in the Transcript window. Example 5–2 shows the messages you should see.

Example 5–2. Transcript Message for the Low Frequency Pattern Test

```
INFO: top.pgm: Starting test low_freq.hex
INFO: top.pgm.read_file: Read file low_freq.hex success
INFO: top.pgm.read_file: Read file walking_ones_rev.hex success
INFO: top.pgm: Test low_freq.hex passed
```

11. To run the random number pattern test, modify **\simulation_tutorial\test_include.svh** (Table 5–3).

Table 5–3. Values for Random Number Pattern Test

Macro	New Value
PATTERN_POSITION	32
NUM_OF_PATTERN	64
NUM_OF_PAYLOAD_BYTES	1024
FILENAME	random_num.hex
FILENAME_REV	random_num_rev.hex

12. Reload the **load_sim.tcl** script, run the simulation for 40 µs, and you observe the results that Example 5–3 shows.

Example 5–3. Transcript Message for the Random Number Pattern Test

```
INFO: top.pgm: Starting test random_num.hex
INFO: top.pgm.read_file: Read file random_num.hex success
INFO: top.pgm.read_file: Read file random_num_rev.hex success
INFO: top.pgm: Test random_num.hex passed
```

In this chapter, you set up the simulation environment for the custom pattern generator component and used BFM test code to perform simulation. You can test your own custom Qsys components with this method, to verify their functionality before you integrate them into a complete system. You can also create a testbench system for a complete Qsys system with this method, and test your top-level system behavior with BFMs.

Additional Information

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this document.

Date	Version	Changes
April 2011	2.0	Updated for Qsys v11.0.
December 2010	1.1	Maintenance release.

How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

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

Note to Table:

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

Typographic Conventions

The following table shows the typographic conventions this document uses.

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

Visual Cue	**Meaning**
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the 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`. The suffix n denotes an active-low signal. For 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`).
↵	An angled arrow instructs you to press the Enter key.
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 question mark directs you to a software help system with related information.
👣	The feet direct you to another document or website with related information.
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 envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.