AN 812: Platform Designer System Design Tutorial

Online Version
Send Feedback

ID: 683855
AN-812
Version: 2018.04.02

Contents

683855 | 2018.04.02

Send Feedback

Platform Designer System Design Tutorial

The Platform Designer system integration tool for Intel FPGA and SoC devices automatically generates interconnect logic to connect intellectual property (IP) components and subsystems. Using Platform Designer saves time and effort in the design process. Platform Designer inherits the ease of use of Platform Designer (Standard). In addition, Platform Designer introduces hierarchical isolation between system interconnect and IP components. This tutorial is for users who have basic knowledge of Intel® Quartus® Prime Pro Edition software and Platform Designer (Standard), and want to experience the new features of Platform Designer.

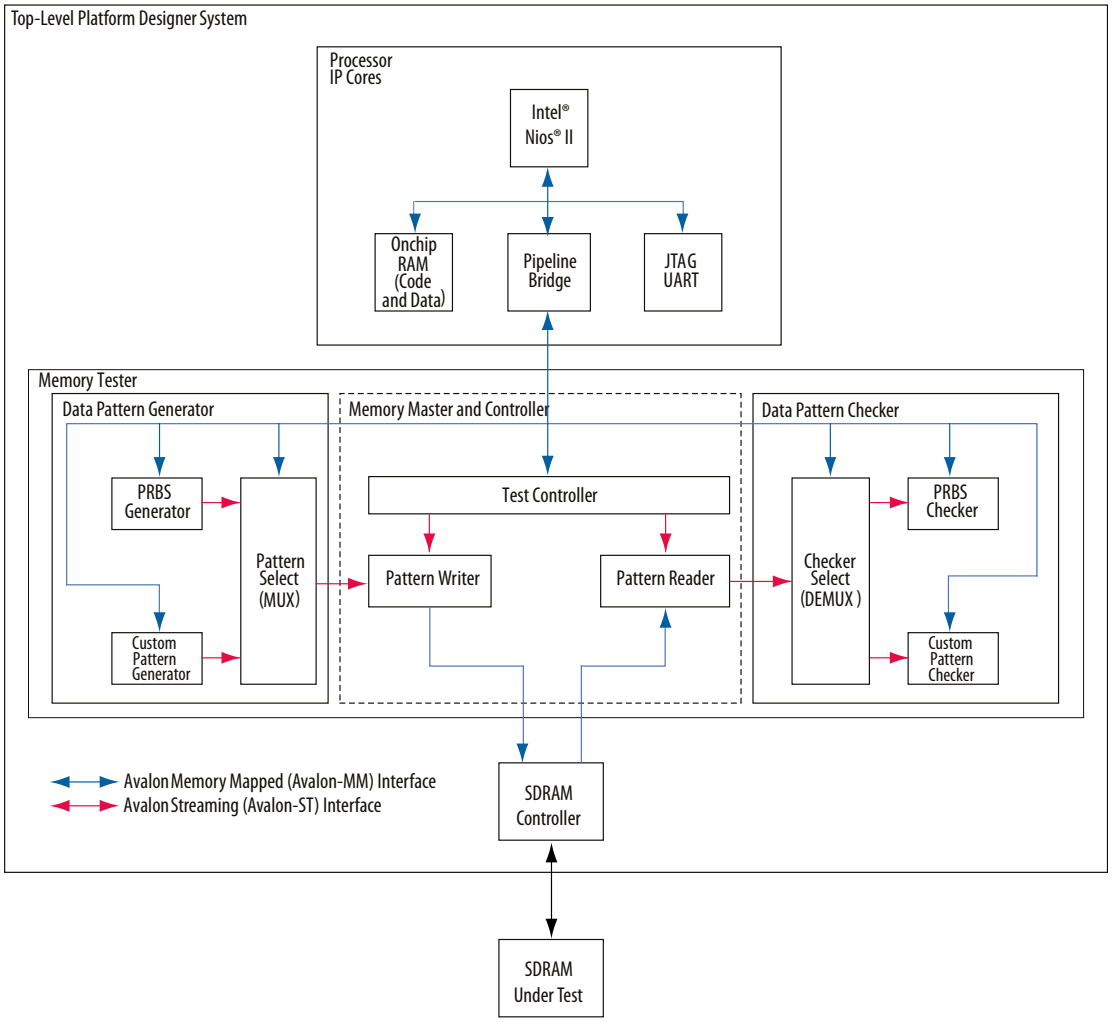
This tutorial guides you through the following processes:

- Building systems in Platform Designer, and integrating those systems into an Intel Quartus Prime Pro Edition project.
- Explains the different user flows between Platform Designer (Standard) and Platform Designer.
- Demonstrates some of the new features of Platform Designer and how it increases efficiency and flexibility for team-based design.

The procedures in this tutorial provide you with a template to design a system that uses various test patterns to test an external memory device. The final system contains the following components:

- A processor subsystem which contains an Intel Nios® II/e core. The subsystem also includes an on-chip RAM to store the software code and a JTAG UART to communicate and display the memory test results in the host PC's console.
- A memory tester subsystem to interact with an SDRAM controller.
- The memory tester subsystem consists of a pattern generator subsystem, a pattern checker subsystem, a memory tester, a pattern writer, and a pattern reader.
- The pattern generator subsystem consists of a custom pattern generator, a pseudo random binary sequence (PRBS) pattern generator, along with a multiplexer (MUX) to select between these two.
- A data pattern checker subsystem consisting of a custom pattern checker, a pseudo random binary sequence (PRBS) pattern checker, along with a demultiplexer (DEMUX).
- Pattern writer and pattern reader subsystems that interacts with the SDRAM controller.
- A SDRAM controller to control the off-chip DDR SDRAM device under test.

Figure 1. Platform Designer System



There are four broad steps in this tutorial:

1. Build a processor subsystem from scratch in Platform Designer.
2. Build a top-level Platform Designer system with memory tester subsystem instantiated as a generic component.
3. Implement a generic component.
4. Create a Nios II software application and run the design on a FPGA.

Hardware and Software Requirements

This design targets the Intel Arria® 10 GX FPGA Development Kit (with DDR4 daughter card installed). To complete this tutorial, you need the following software and tools:

- Intel Quartus Prime Pro Edition 17.0 or later
- Nios II EDS (installs with the Intel Quartus Prime Pro Edition software)
- Board Test System (installs with the Intel Arria 10 GX FPGA Development Kit package)

Related Information

- Intel Quartus Prime Pro Edition Download Page
- Intel Arria 10 GX FPGA Development Kit
- Intel Arria 10 Board Test System

Download and Install the Tutorial Design Files

1. On the **Platform Designer Tutorial Design Example** page, under **Using this Design Example**, click **Platform Designer Tutorial Design Example (.zip)** to download and install the tutorial design files for the Platform Designer tutorial.
2. Extract the contents of the archive file to a directory on your computer. Do not use spaces in the directory path name.

The `qsys_pro_tutorial_design_Arria_10_17p0.zip` contains the following project files and is referred to as <*`project folder`*> in the rest of the document.

Table 1. Qsys Pro Design Tutorial Project Files

Folder Structure	Description
`/complete_design`	The final design. You can use this design as a reference and guidance while you follow the tutorial. You may also use the prebuilt systems in it if you want to skip certain steps of this tutorial.
`/ip`	The folder that stores IP component source files. The `pattern_checker_system` and `pattern_generator_system` are pre-generated for you.
`/memory_tester_ip`	The folder that contains source files for all custom components.
`/software`	This folder contains source code for building Nios II software applications and two scripts that automate this process for you.
`A10.qpf`	An Intel Quartus Prime Project file (`.qpf`).

continued...

Folder Structure	Description
A10.qsf	An Intel Quartus Prime Settings file (.qsf), containing pre-defined pin assignments.
memory_tester_search_path.ipx	IP Index file (.ipx) that specifies the path to the source files of the custom components.
memory_tester_subsystem_bb.ipxact	The .ipxact file that defines the interfaces for your generic component.
my_constraints.sdc	A Synopsys Design Constraints, or SDC, file (.sdc) containing timing constraints.
pattern_checker_system.qsys	Pre-built Platform Designer System file (.qsys).
pattern_generator_system.qsys	Pre-built Platform Designer System file (.qsys).
top_level.v	Top-level Verilog Design file (.v) .

Related Information

- Platform Designer System Design Example
- Platform Designer System Design Example (.zip)

Build the Hardware Design

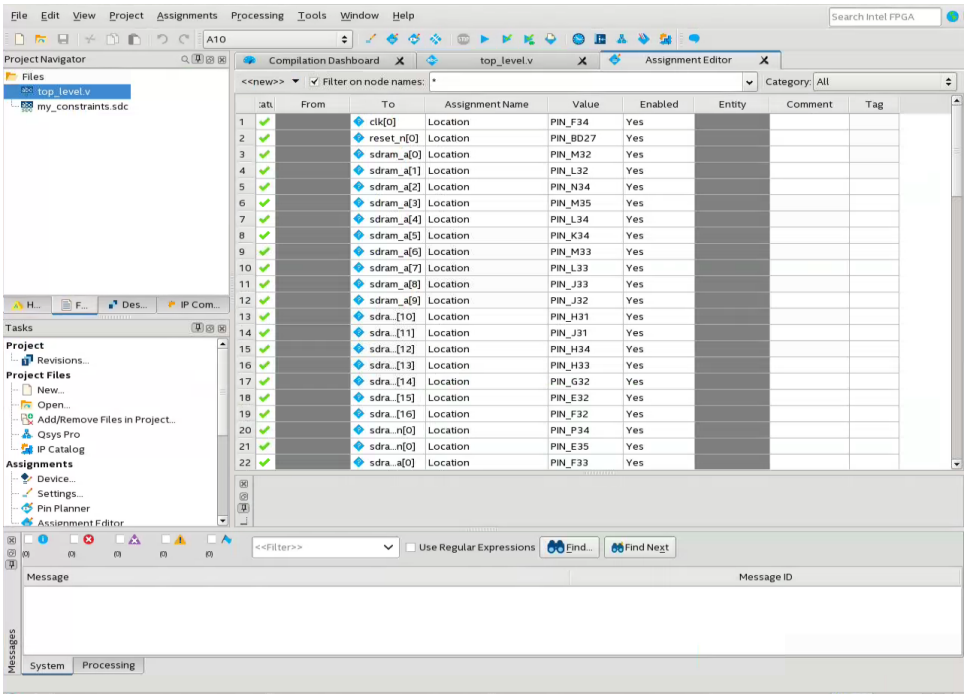
Open the Intel Quartus Prime Pro Edition Project

You must specify or create an Intel Quartus Prime Pro Edition project when you create or open a new Platform Designer system. Platform Designer inherits the device family or number from the Intel Quartus Prime Pro Edition software, which guarantees the or Platform Designer coherency. To open the Intel Quartus Prime Pro Edition project:

1. Launch Intel Quartus Prime Pro Edition software.
2. Click **File ➤ Open Project**.
3. Browse to the project directory.
4. Select A10.qpf and click **Open**.

The top-level RTL, pin assignments, and timing constraints have been created for you. The file references and pin assignments are saved in A10.qsf.

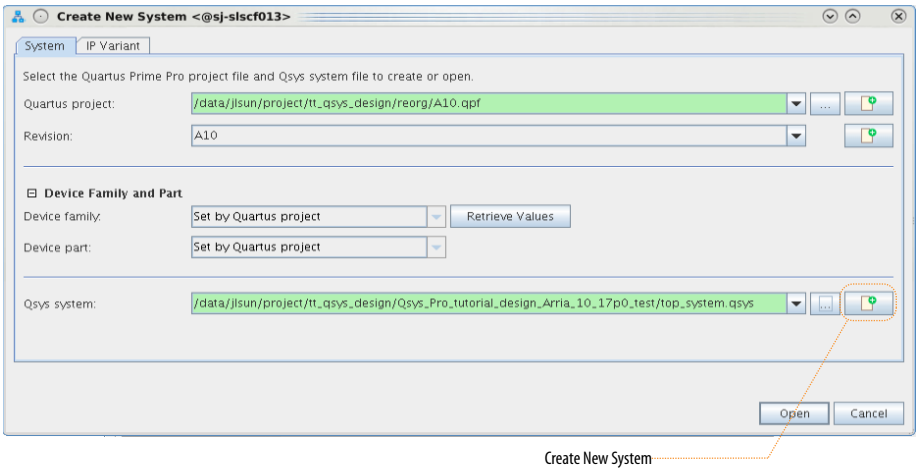
Figure 2. Intel Quartus Prime Pro Edition Project



Build a Platform Designer System with a Top-Down Approach

1. To launch Platform Designer, click **Tools ➤ Platform Designer**.
2. Click the **Create new Qsys system** button and name the new Platform Designer system **top_system.qsys**.

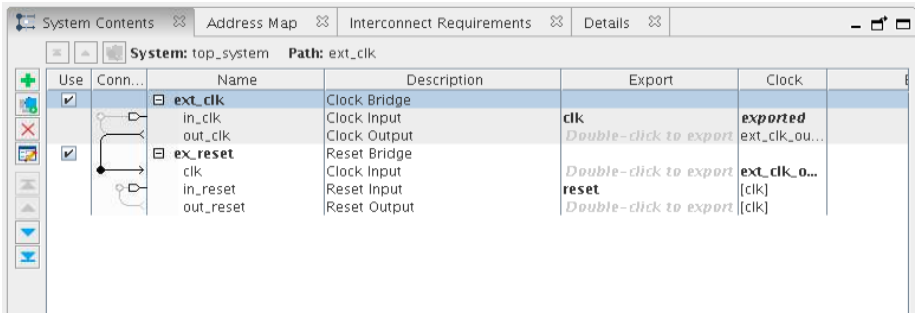
Figure 3. Create New System Dialog Box



3. Click **Create**. The resulting system comes pre-populated with a clock bridge and a reset bridge.

4. Right-click the name of the **clock_in** component and click **Rename**. Type `ext_clk`.
5. In the parameter editor, change the **Explicit clock rate** to `100MHz` (100,000).
6. Right-click the name of the **reset_in** component and click **Rename**. Type `ext_reset`.

Figure 4. Rename the Clock Bridge and Reset Bridge

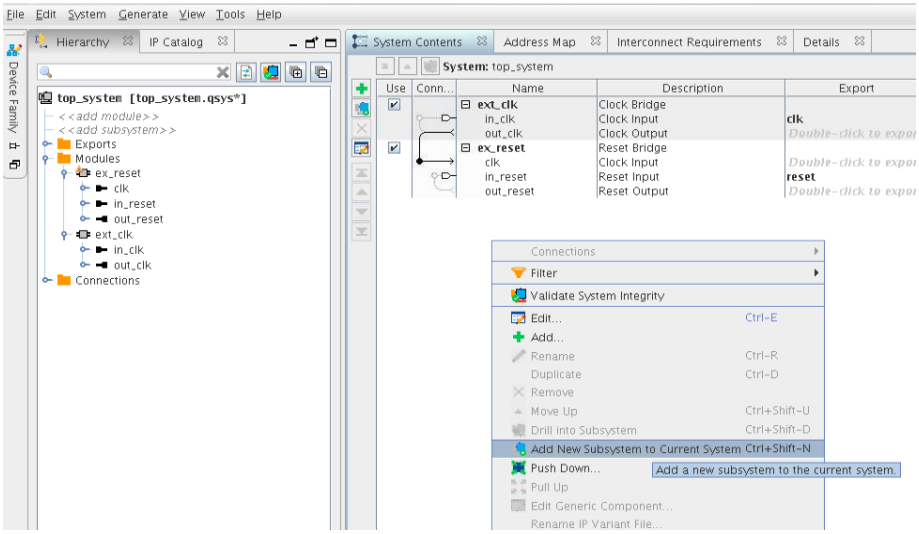


Add a Processor Subsystem to the Top-Level

Using subsystems helps maintain design hierarchy. You can add a subsystem in Platform Designer and easily implement it.

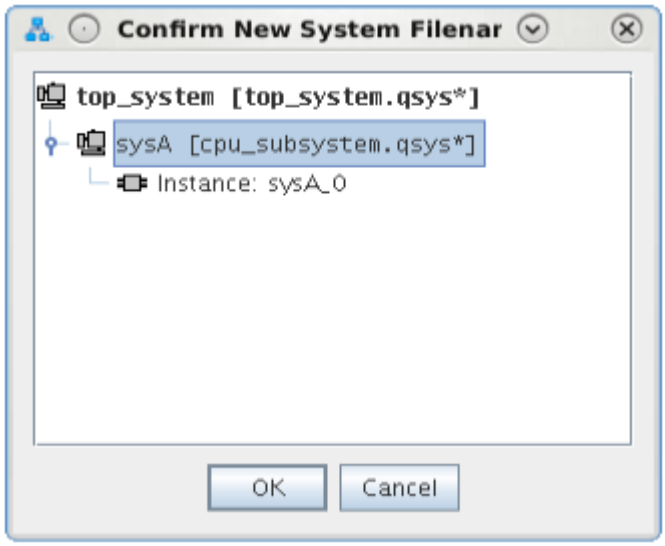
1. Right-click in the **System Contents** tab and click **Add New Subsystem to Current System**.

Figure 5. Add a New Subsystem to the Current System



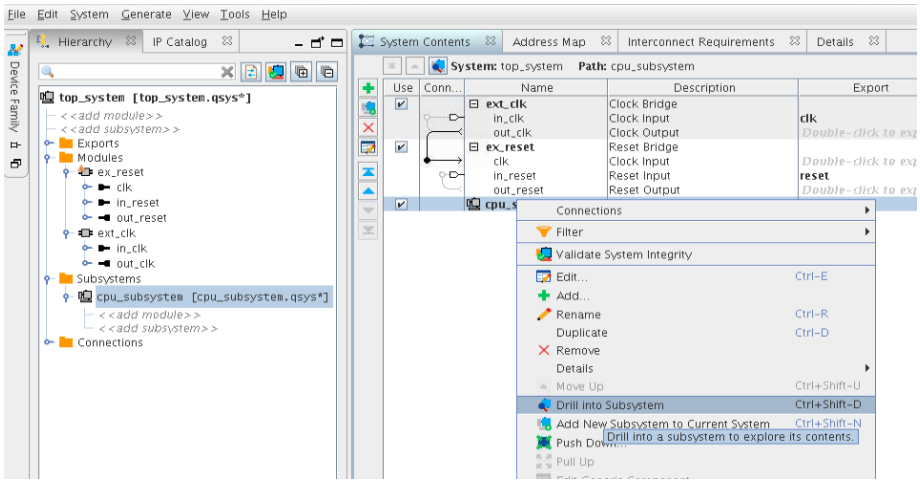
2. In the **Confirm New System Filename** dialog box, click the sysA subsystem and rename it by typing `cpu_subsystem.qsys`.
3. Click **OK**.

Figure 6. Confirm New System Filename Dialog Box



4. To rename the instance from `sysA_0` to `cpu_subsystem`, right-click the name of the new subsystem in **System Contents** and click **Rename**. Type `cpu_subsystem`.
5. To implement the `cpu_subsystem` component, right-click the name and click **Drill into Subsystem**. Alternatively, you can double-click `cpu_subsystem` in the **Subsystems** folder in the Hierarchy list.

Figure 7. Drill into Subsystem Command to Modify a New Subsystem



This opens `cpu_subsystem.qsys` as a new Platform Designer project where you can add components.

Build the Processor Subsystem

To build the `cpu_subsystem` subsystem, you add IP components from the IP Catalog:

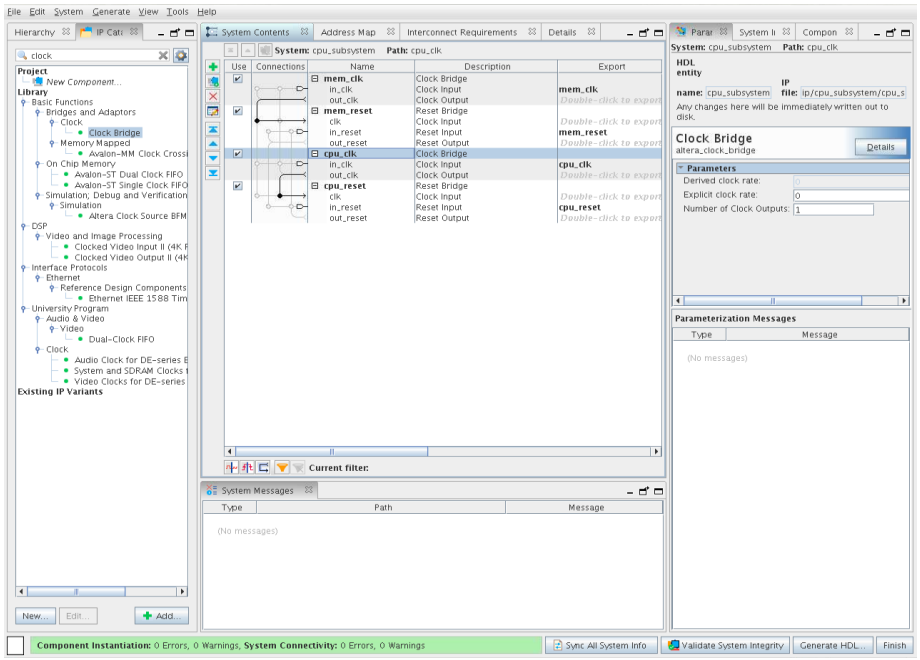
1. Type `clock` in the search box of the IP Catalog and double-click **Clock Bridge** to add that component.
2. Type `reset` in the search box of the IP Catalog and double-click **Reset Bridge** to add that component.
3. Right-click the name of the clock bridge and click **Rename**. Type `mem_clk` to rename the clock bridge.
4. Right-click the name of the reset bridge and click **Rename**. Type `mem_reset` to rename the reset bridge.
5. To add a second clock bridge, type `clock` in the search box of the IP Catalog and double-click **Clock Bridge** to add that component.
6. To add a second reset bridge type `reset` in the search box of the and double-click **Reset Bridge** to add that component.
7. Right-click and rename the new clock bridge and reset bridge to `cpu_clk` and `cpu_reset`, respectively.
8. Connect the `out_clk` signal of `mem_clk` to the `clk` signal of `mem_reset`.
9. Connect the `out_clk` signal of `cpu_clk` to the `clk` signal of `cpu_reset`.
10. Edit the exported interface by double-clicking the name in the **Export** column, from the following table:

Table 2. Export Rename Values

Component Name	Description	Export Value
`mem_clk`	**Clock Input**	`mem_clk`
`mem_reset`	**Reset Input**	`mem_reset`
`cpu_clk`	**Clock Input**	`cpu_clk`
`cpu_reset`	**Reset Input**	`cpu_reset`

Your results should match those in the following figure:

Figure 8. Clock and Reset Components

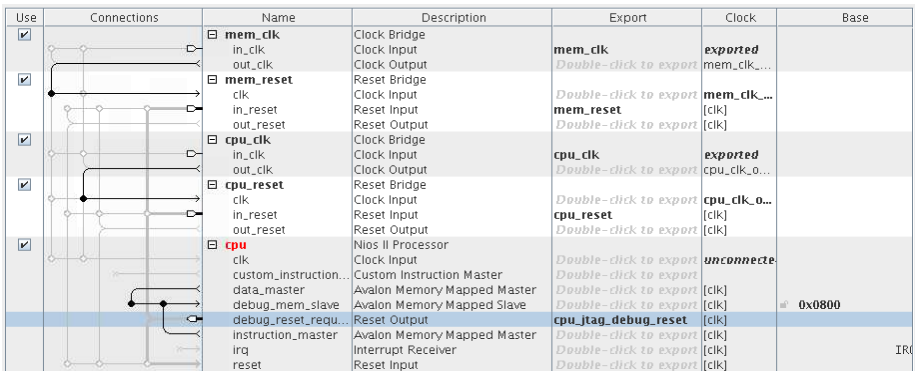


Add a Nios II Processor

1. Type `nios` in the search box of the IP Catalog and double-click **Nios II Processor**.
2. In the **Select an Implementation** parameter editor, select the **Nios II/e** processor.
3. To add the **Nios II/e** processor to the design, click **Finish**.
4. Right-click the name of the Nios II processor component and click **Rename**. Type `cpu` to change the name.
5. In the **Export** column, double-click the entry corresponding to the **Reset Output** for the **cpu** component and rename it `cpu_jtag_debug_reset`.

   Errors regarding reset and exception slaves can be resolved after you add connections.

Figure 9. cpu_subsystem Export Naming



Add RAM, JTAG UART, and Avalon-MM Pipeline Bridge

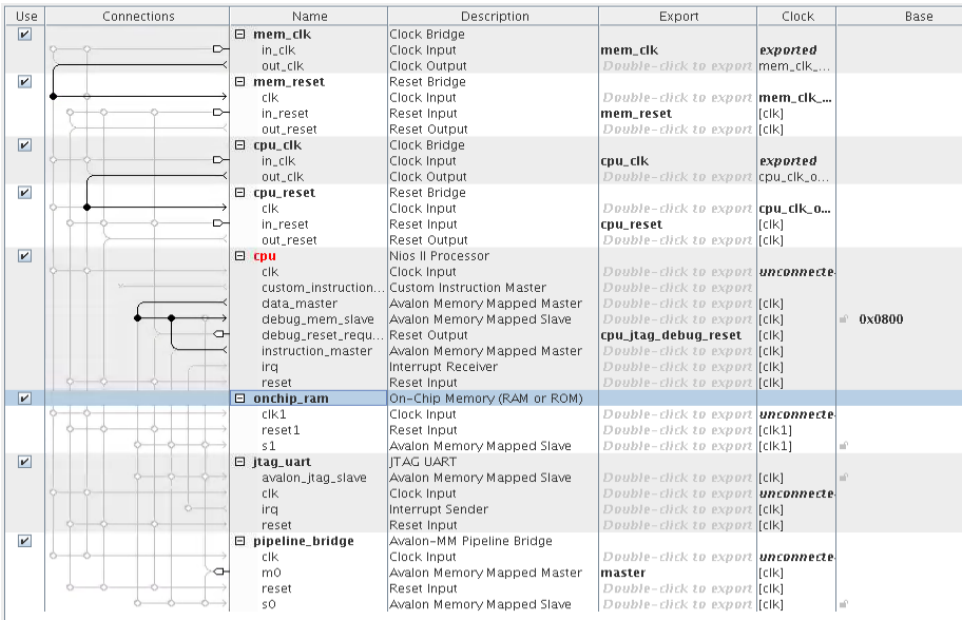
The final components you'll need to add and configure are an On-Chip RAM, a JTAG UART, and an Avalon-MM Pipeline Bridge.

1. Type `ram` in the IP Catalog search box and double-click **On-Chip Memory (RAM or ROM)**.
2. In the **On-Chip Memory (RAM or ROM)** parameter editor, in the **Size** box, set the **Total memory size** to 8192 bytes.
3. To add the **On-Chip Memory (RAM or ROM)** component to your design, click **Finish**.
4. Right-click the name of the **On-Chip Memory (RAM or ROM)** component and click **Rename**. Type `onchip_ram` to change the name.
5. Type `jtag uart` in the IP Catalog search box and double-click **JTAG UART**.
6. To add the **JTAG UART** component to your design with default settings, click **Finish**.
7. Right-click the name of the **JTAG UART** component and click **Rename**. Type `jtag_uart` to change the name.
8. Type `pipeline bridge` in the **IP Catalog** search box and double-click **Avalon-MM Pipeline Bridge**.
9. In the **Avalon-MM Pipeline Bridge** parameter editor, change the following settings:
   - Set the **Address width** to 16.
   - Set the **Maximum pending read transactions** to 1.
10. To add the **Avalon-MM Pipeline Bridge** to your design, click **Finish**.
11. Right-click the name of the **Avalon-MM Pipeline Bridge** component and click **Rename**. Type `pipeline_bridge` to change the name.
12. In the **Export** column, double-click the entry that corresponds to the `m0` signal for the **pipeline_bridge** component and type `master`.

The Avalon-MM Pipeline Bridge allows the processor subsystem `cpu_subsystem` to export a single Avalon-MM master interface. Your design can then access the slave interfaces in a higher-level system, and handle address offsets automatically. The bridge also improves timing performance.

All the required components are now included in this subsystem. Compare the settings in your design with the following figure and make sure your components and exported interfaces are named correctly.

Figure 10. Export Names for cpu_subsystem Components



Connect cpu_subsystem Components

Connect the component signals below by clicking the dots across from the appropriate signals, or by right-clicking the signal and choosing from the drop-down menu.

Follow these steps to connect the components:

Figure 11. Illustrated Clock and Reset Component Connections for cpu_subsystem

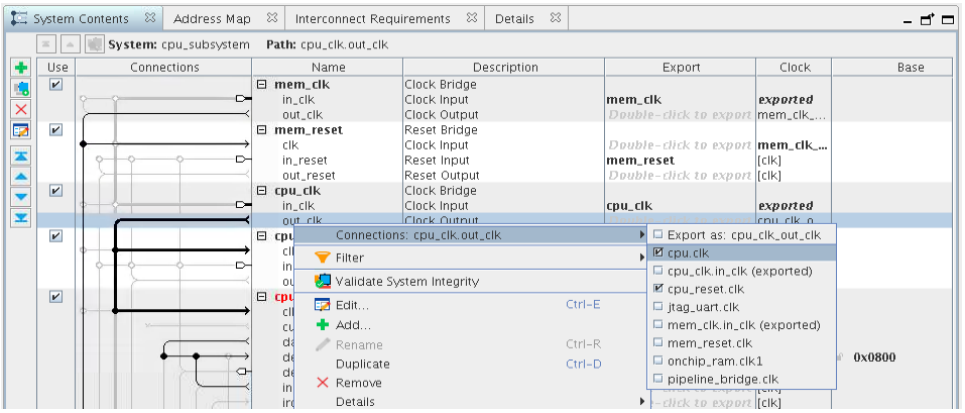
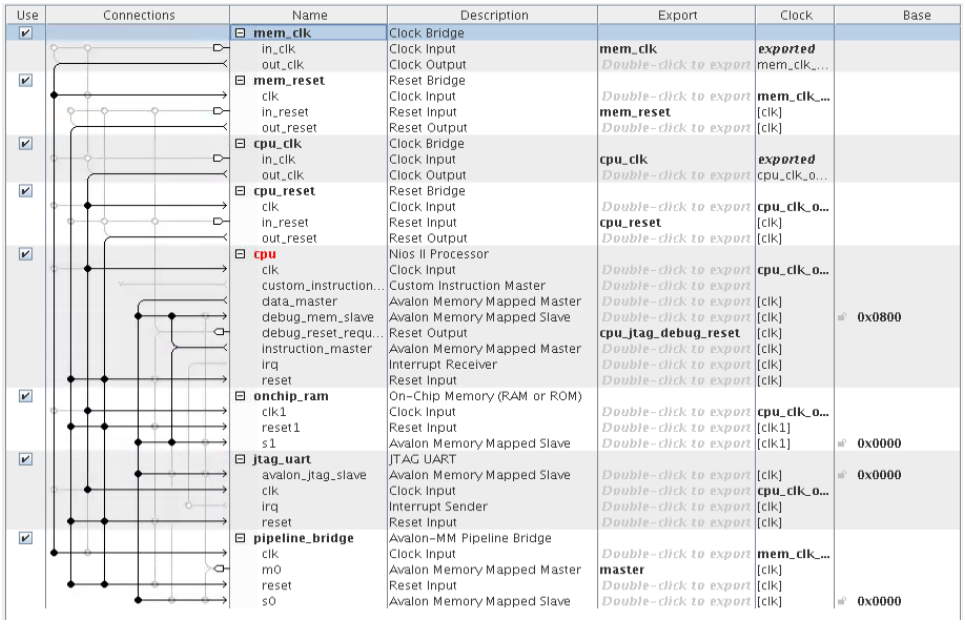


Table 3. Component Connections for cpu_subsystem

Source Component/Signal	Target Component/Signal
mem_clk/out_clk	**pipeline_bridge**/clk
cpu_clk/out_clk	• **cpu_reset**/clk • **cpu**/clk • **onchip_ram**/clk1 • **jtag_uart**/clk
mem_reset/out_reset	• **cpu**/reset • **onchip_ram**/reset1 • **jtag_uart**/reset • **pipleline_bridge**/reset
cpu_reset/out_reset	• **cpu**/reset • **onchip_ram**/reset1 • **jtag_uart**/reset • **pipleline_bridge**/reset
cpu/data_master	• **onchip_ram**/s1 • **jtag_uart**/avalon_jtag_slave • **pipleline_bridge**/s0
cpu/instruction_master	**onchip_ram**/s1

Compare the finished connections to the following figure:

Figure 12. Component Connections for cpu_subsystem

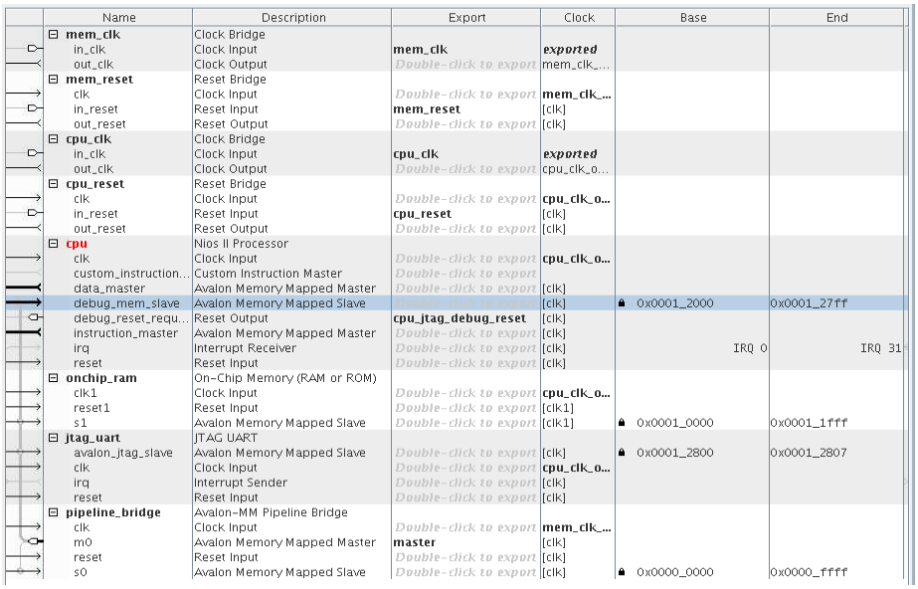


System Connectivity Error appears in the **System Messages** tab. To access this tab, click **View ➤ System Messages**. The **System Connectivity Error** occurs because when the base address of the Avalon-MM slaves are not assigned, which can cause address overlap.

Follow these steps to assign the **Base** address to the value shown in the following figure. Click the "lock" icon to lock the address.

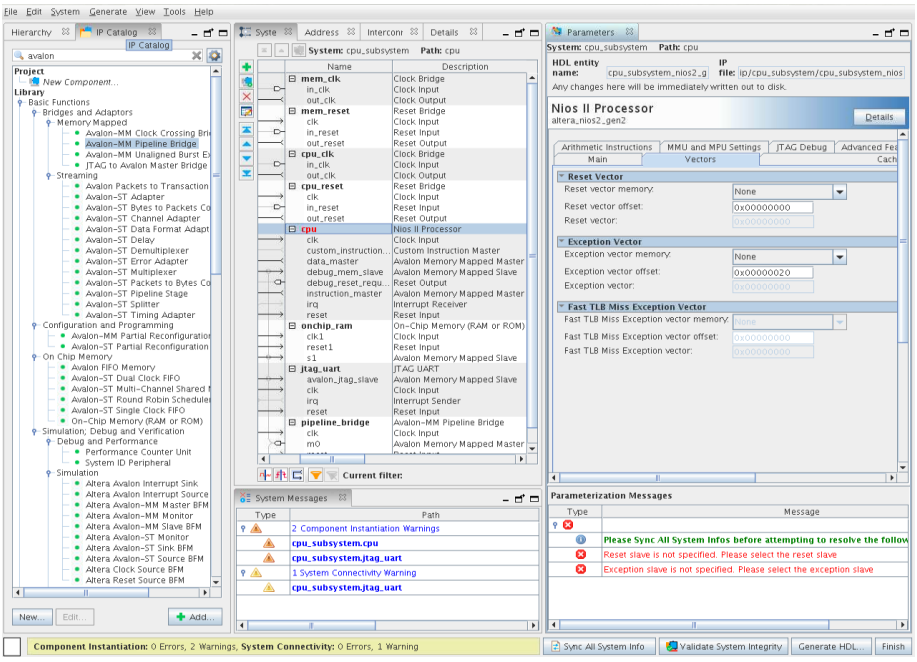
1. In the **Base** column, click the value for **Avalon Memory Mapped Slave** (**Description** column) of the **cpu** component and type `12000`.
2. Find the **Avalon Memory Mapped Slave** entry for the **onchip_ram** component and type `10000` as the value in the **Base** column.
3. Find the **Avalon Memory Mapped Slave** entry for the **jtag_uart** component and type `12800` as the value in the **Base** column.

Figure 13. Base Address Assignments for cpu_subsystem Components



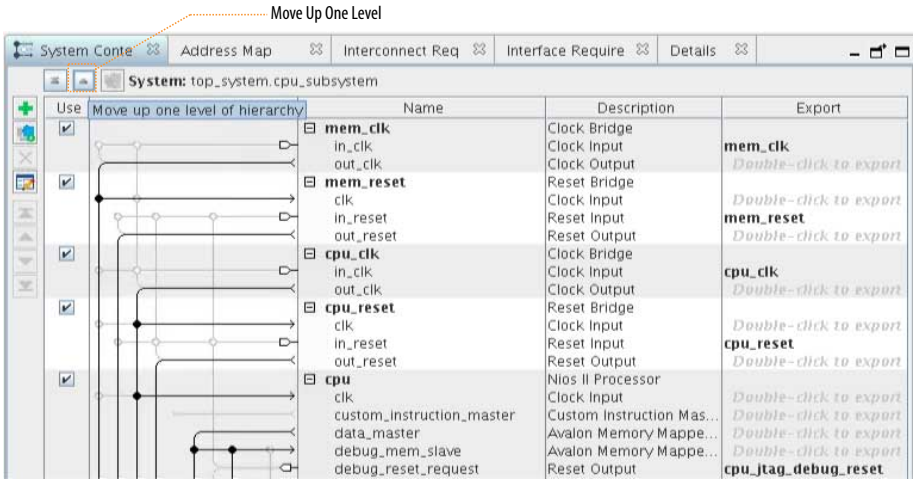
4. To resolve any remaining system connectivity errors, in the **System Messages** tab, click **Sync All System Info** in the bottom of the GUI. This synchronizes the component instantiations with their `.ip` files.
5. To resolve errors in the parameterization of the **cpu** component (the name of the component is still red), double-click **cpu** and you can see the **Parameterization Messages** in the **Parameters** tab. Platform Designer separates the messages for system connectivity and component parameterization, which simplifies the error and resolution compared to the combined messaging in Platform Designer (Standard).

Figure 14. Parameterization Messages



6. In the **Vectors** tab, set **Reset vector memory** and **Exception vector memory** both to **onchip_ram.s1** to resolve the error messages.
7. Click **File ➤ Save** to save the project. There is no need to generate the RTL for the Platform Designer system at this time. Click **Move up one level of hierarchy** to return to `top_level.qsys` system.

Figure 15. Move Up One Hierarchy Level



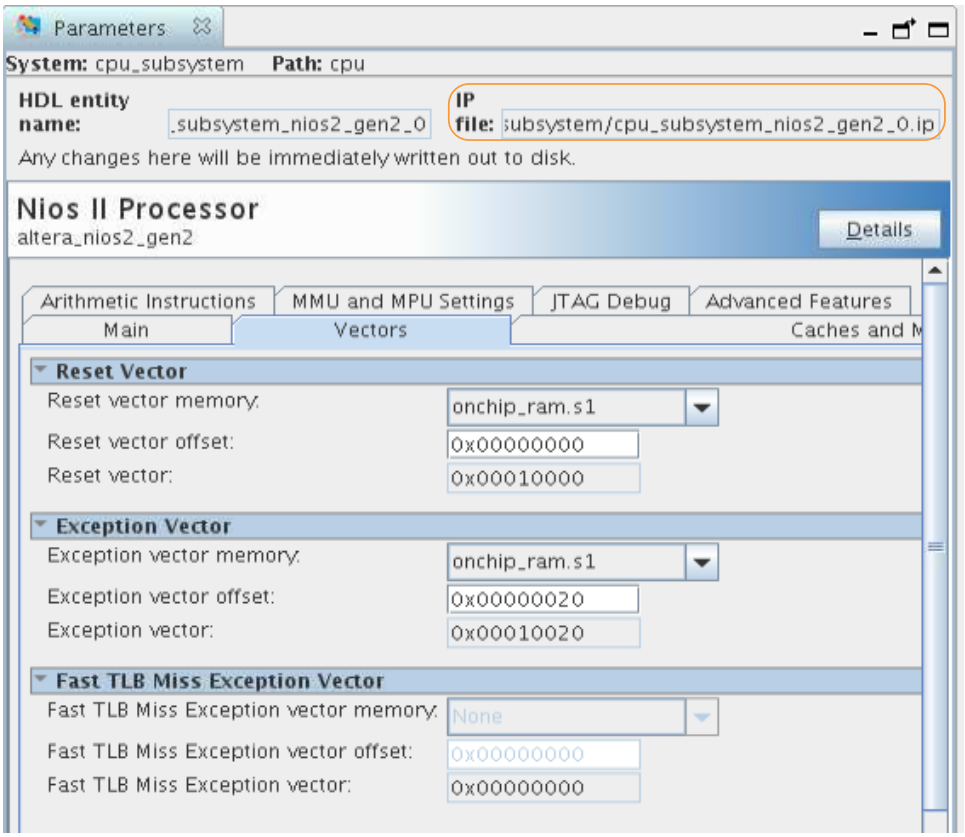
Platform Designer and Platform Designer (Standard) Differences

Platform Designer introduces a hierarchical isolation between system interconnect and IP components by saving the parameters of each IP component in a `.ip` file under `<project folder>/ip/<Platform Designer system name>` and saving of the

system interconnect in a `.qsys` file under `<project folder>`. The RTL of each `.ip` or `.qsys` file can be generated in isolation as it contains the full information required to reproduce the state of the RTL. There are no unresolved dependencies between files.

For example, Platform Designer saves the Nios II processor parameterization in `<project folder>/ip/cpu_subsystem/cpu_subsystem_ nios2_gen2_0.ip`, and the system interconnect in `<project folder>/cpu_subsystem.qsys`.

Figure 16. File Location for Nios II Processor IP File



Platform Designer and Platform Designer (Standard) differ also differ in how they handle IP files:

- Platform Designer requires that you include the `.qsys` file along with a list of `.ip` files associated with that Platform Designer project. The Intel Quartus Prime Pro Edition software manages this for you after you save your Platform Designer project.
- The older Platform Designer (Standard) tool saves both component instantiation and system interconnects in a `.qsys` file. When integrating a Platform Designer (Standard) system to a Intel Quartus Prime project, you only need to include a single Intel Quartus Prime IP file (`.qip`).

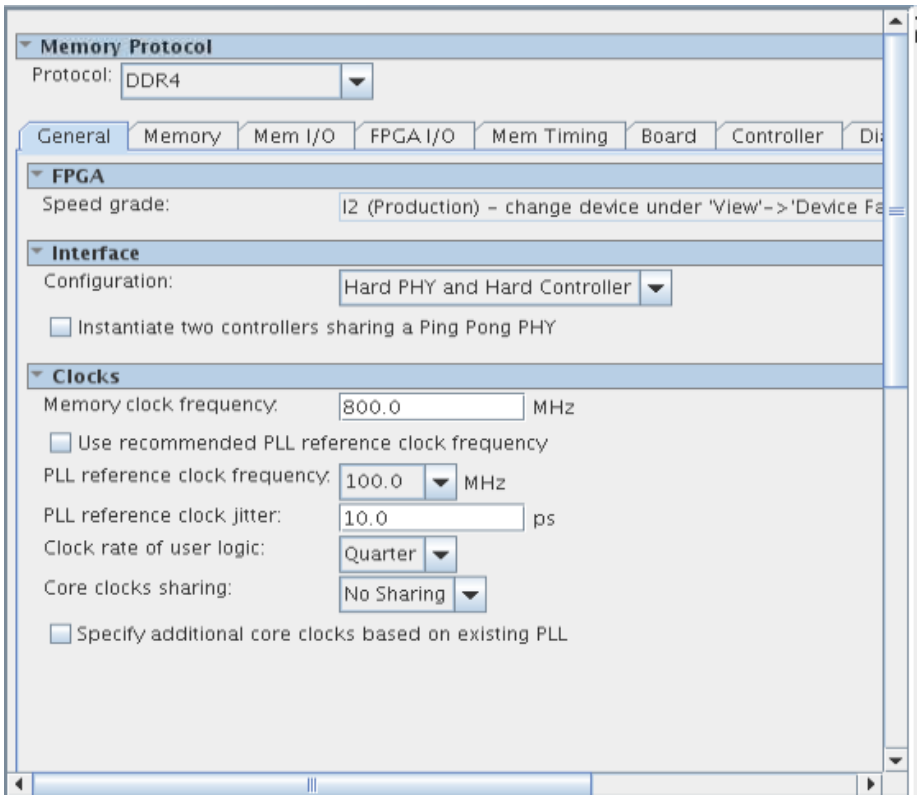
Add External Memory Interface

The next step is to add an **Arria 10 External Memory Interfaces** component and use presets to configure the parameters.

The **Presets** tab displays a list of applications consisting of different protocols and development kits. You can choose from the list and apply a pre-defined set of parameters to the selected IP components. The DDR4 component from the list of **Presets** implements a pre-configured module. Modify the following parameters to help meet timing for this design:

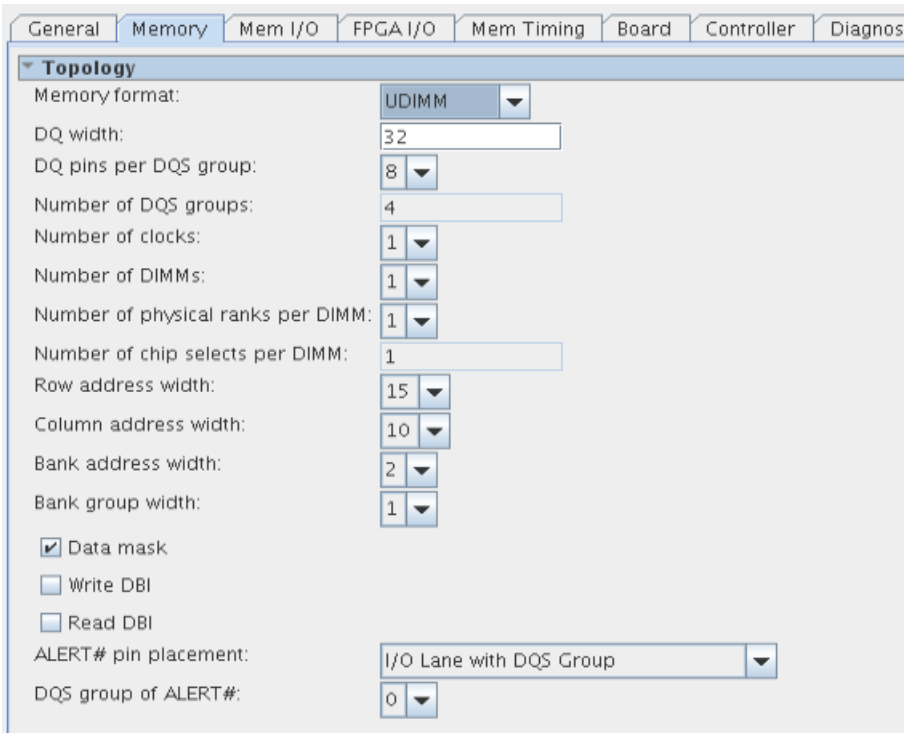
1. Type `external memory` in the **IP Catalog** search box and double-click **Arria 10 External Memory Interfaces** to add it to the system.
2. In the **Arria 10 External Memory Interfaces** parameter editor, select the **Arria 10 GX FPGA Development Kit with DDR4 HILO** from the **Preset** library and click **Apply**.

Figure 17. Arria 10 External Memory Interfaces Pane



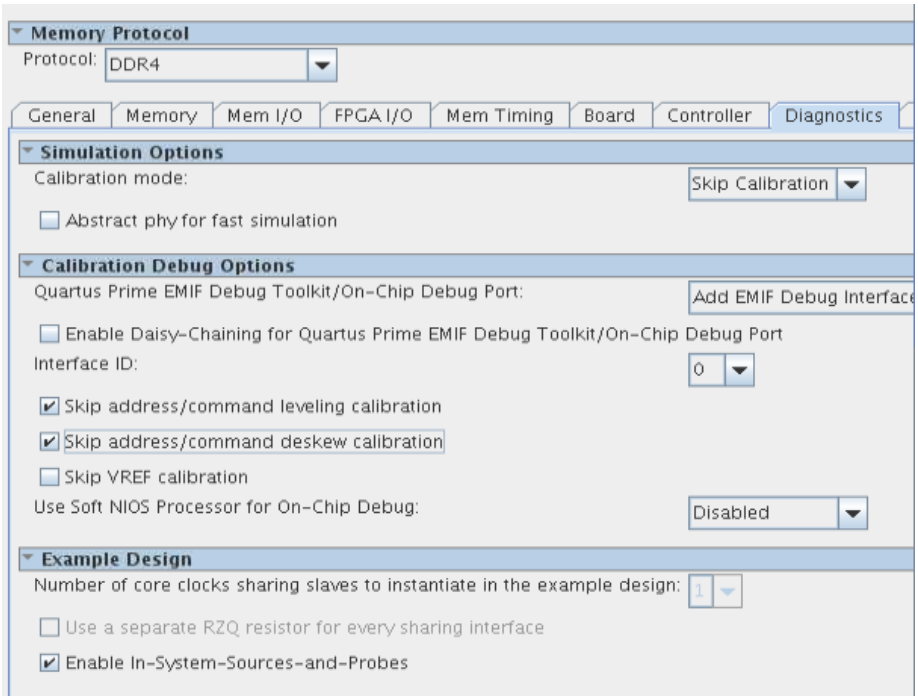
3. In the **Clocks** section of the **General** tab, change **Memory clock frequency** to 800MHz and the **PLL reference clock frequency** value to 100MHz.

Figure 18. Memory Tab



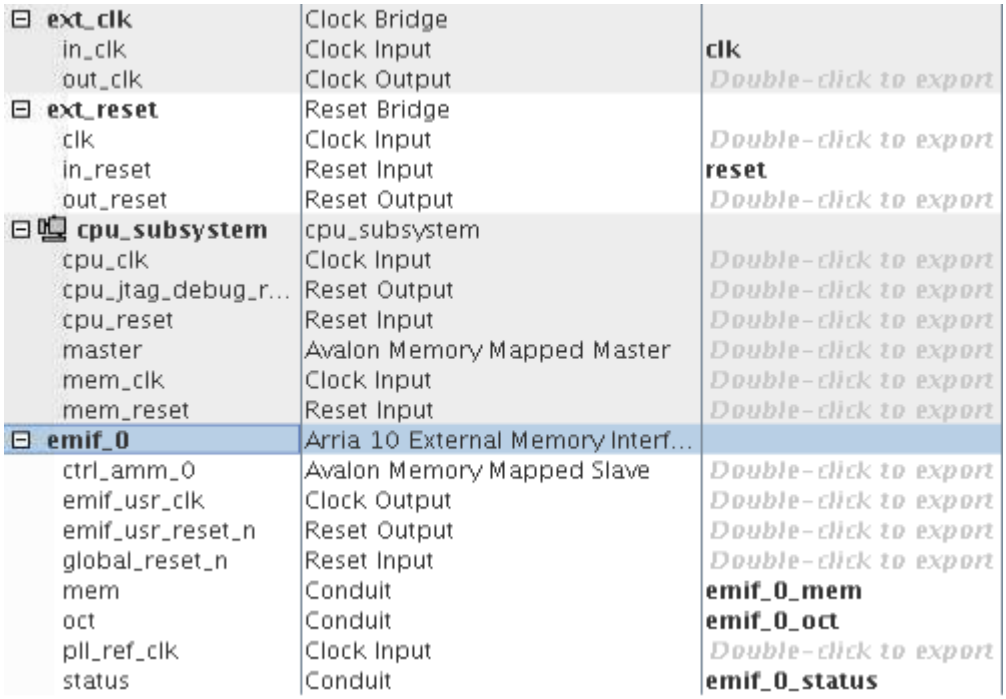
4. Click the **Memory** tab and specify the following:
   - Change the **DQ width** to 32.
   - Turn off **Read DBI**.
   - Select '0' from the **DQS group of ALERT#** list.

Figure 19. Diagnostics Tab



5. Click the **Diagnostics** tab and specify the following:
   - Turn on **Skip address/command leveling calibration**.
   - Turn on **Skip address/command deskew calibration**.
6. Click **Finish**.
7. Right-click the name of the **top_system_emif_0** component and click **Rename**. Type `emif_0`.
8. In the **Export** column, double-click the `mem`, `oct`, and `status` conduit interfaces and rename them `emif_0_mem`, `emif_0_oct`, and `emif_0_status`, respectively.

Figure 20. Export Names for emif_0 Signals



Instantiate a Memory Tester Subsystem as a Generic Component

At this stage of memory tester subsystem design, you can use Platform Designer to assemble the whole design to verify and debug components such as the Nios II logic resource usage and DDR4 Calibration.

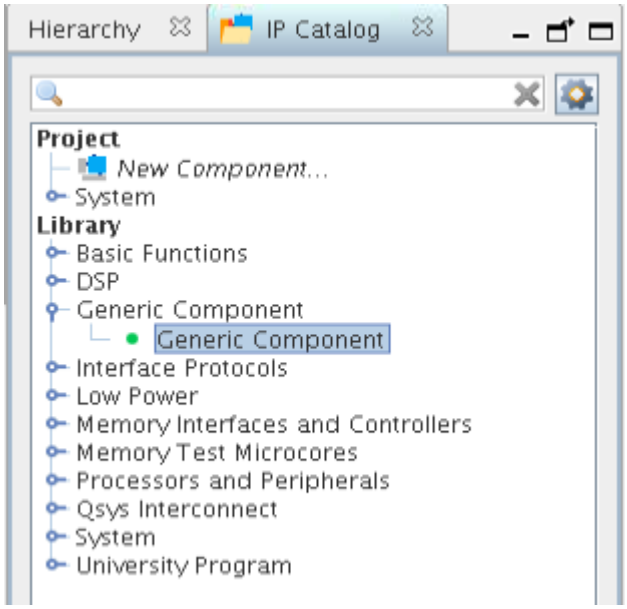
You can instantiate the memory tester subsystem as a generic component (an empty entity with only interfaces defined). When integrating a memory tester subsystem with a processor subsystem and an EMIF controller only the interfaces of the memory tester subsystem are significant.

Instantiation of a generic component does not prevent the completion of other parts of the design. This feature provides a lot of flexibility in the design, and is especially beneficial for large and team-based designs. You need only verify that, when adding the entity implementation, the entity interfaces match the interfaces defined for the generic component.

To instantiate a generic component:

1. In the IP Catalog, double-click **Generic Component**.

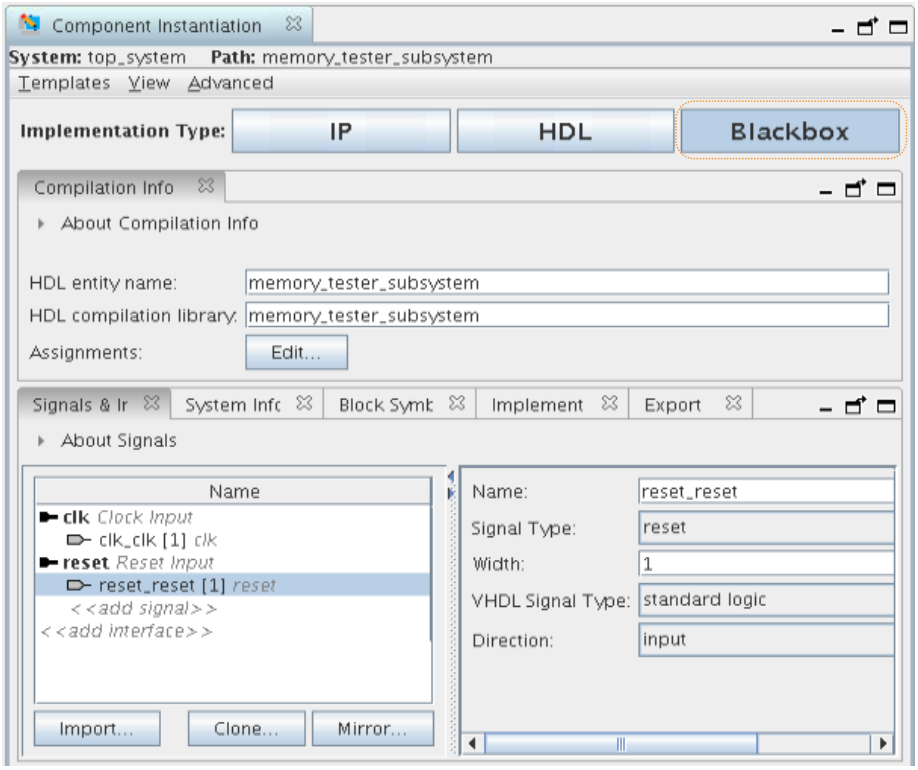
Figure 21. IP Catalog Generic Component



The **Component Instantiation** tab contains three implementation types: **IP**, **HDL**, and **Blackbox**. When you add a generic component, **Blackbox** is the default.

2. Change the **HDL entity name** and **HDL compilation library** to `memory_tester_subsystem`.

Figure 22. Component Instantiation Tab for memory_tester_subsystem Generic Component



3. To add the component, click **Finish**.
4. Right-click the name of the **top_system_generic_component_0** component and click **Rename**. Type `memory_tester_subsystem`.

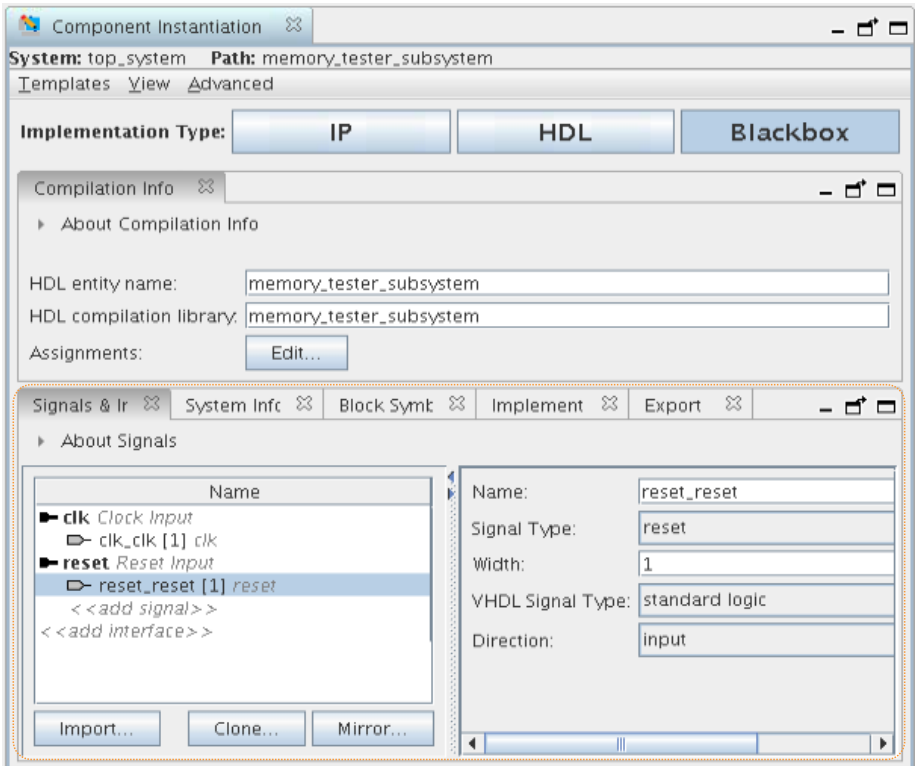
Note: When implementing a generic component with the **Blackbox** option, you don't have to provide the HDL implementation during component instantiation. Simply customize the interfaces and signals and generate an empty HDL file. Then, connect the generic component to other components in Platform Designer, generate interconnects, and finally, compile the project with this empty entity. When you finish the implementation of the generic component, simply replace the generic component with the actual implementation to complete the design. In other words, the generic component functions as a placeholder for the actual component you plan to use.

Platform Designer provides many features to help you add interfaces and signals for a generic component. The following steps, 1-11, showcase how to add signals manually, by using **Mirror** or **Clone**, and how to change parameters. In the final steps, you are going to import a complete interface definition from an `.ipxact` file.

Platform Designer provides many ways to help you add interfaces easily and efficiently.

1. Click **View ➤ Component Instantiation**.
2. Select the **memory_tester_subsystem** component. The instantiation information appears in the **Component Instantiation** tab.
3. Click the **Signals & Interfaces** tab. You can add interfaces manually, **Import** from an IP-XACT file, **Mirror**, or **Clone** from existing interfaces in the system.
4. Click **<< add interface>>** and select **Clock Input** from the drop down list.
5. To change the name of the interface, in the **Name** field, type `clk`.
6. Click **<<add signal>>** and choose `clk`.
7. Repeat steps 4-6 to add a **Reset Input** interface and signal, and rename it `reset`.
8. Click **Apply**.

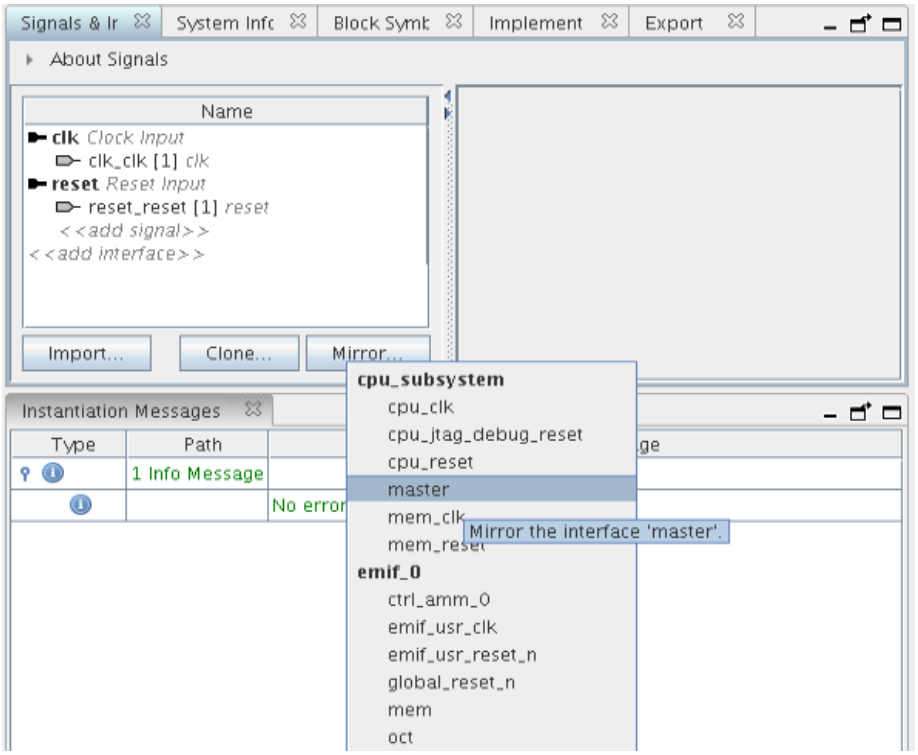
Figure 23. Signal and Interface Options for memory_tester_subsystem



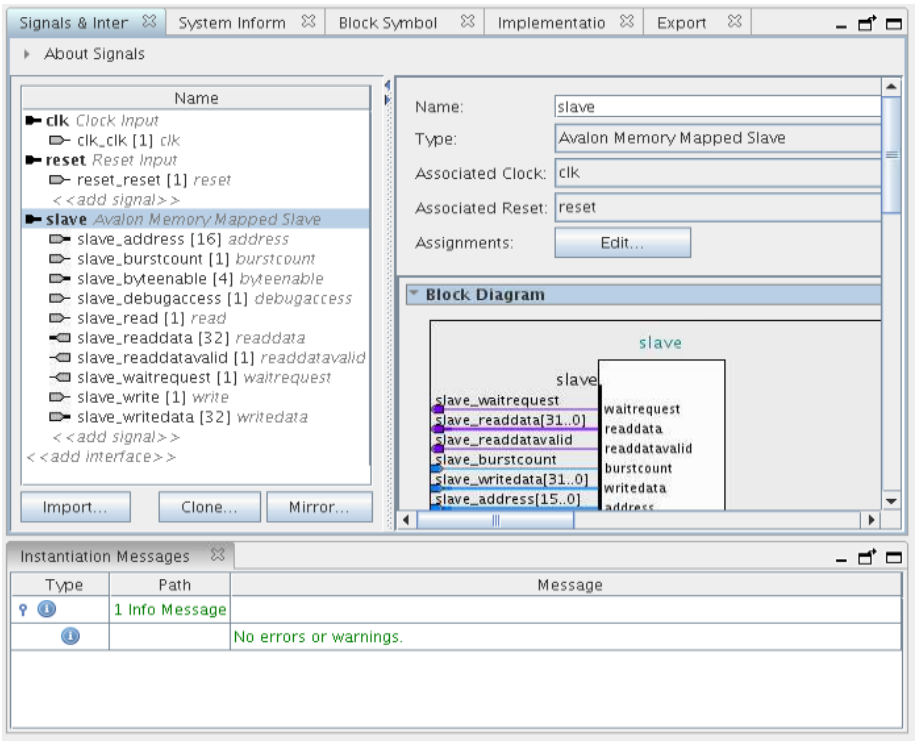
Apart from `clock` and `reset`, the design also requires an Avalon-MM slave interface to communicate with the processor subsystem. It could be tedious to add Avalon-MM slave interface manually since there are address bus, data bus, and many other parameter settings to configure. An easier way is to use the **Mirror** feature.

9. Click **Mirror** and choose the `master` interface of **cpu_subsystem** to add a slave interface.

Figure 24. Create Mirror of Interface 'master'

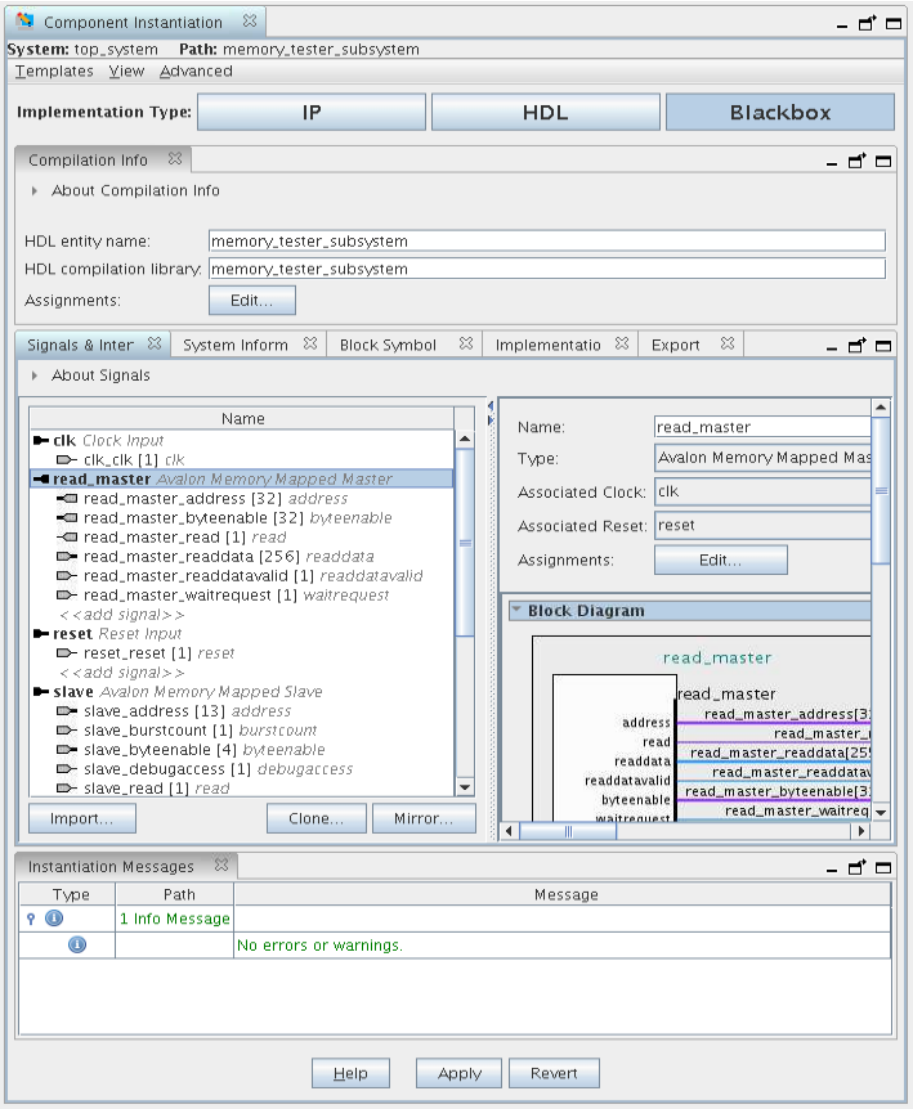


10. You can resolve the errors that appear in the **Instantiation Messages** box by assigning **Associated Clock** and **Associated Reset** to the `clk` and `reset` interfaces in the parameter editor.



11. Locate the **Maximum pending read transactions** box under **Pipelined Transfers** and change that value to 4.
12. Click **Import** and choose `memory_tester_subsystem_bb.ipxact` to add the interfaces.
13. To complete the import step, click **Apply**.

Figure 25. Results of Importing memory_tester_subsystem_ bb.ipxact

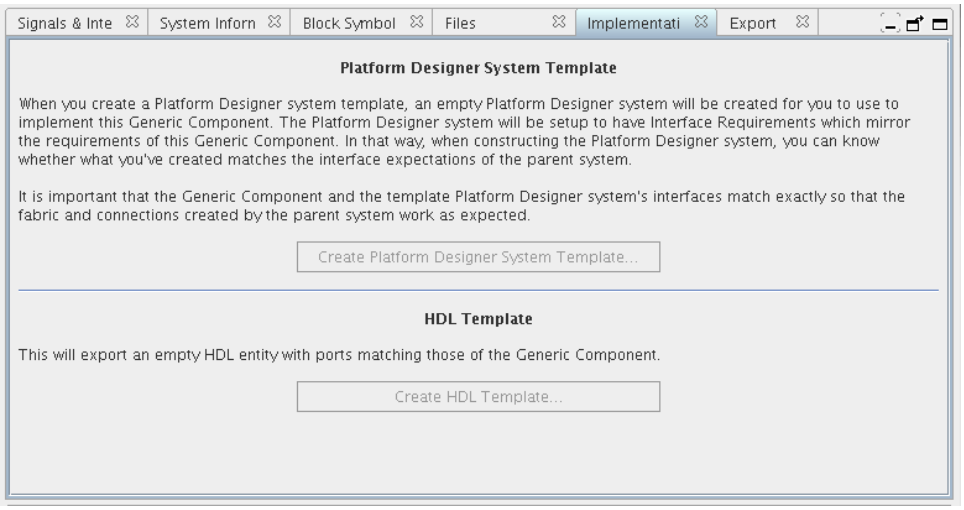


Click the **Implementation** tab to create a Platform Designer template with interface requirements setup to implement the memory tester subsystem. You can also create an HDL template with ports defined.

14. Click **Create Platform Designer System Template ➤ Save** to create the `memory_tester_ subsystem.qsys` file in the `<project folder>`.
15. Click **Create HDL Template ➤ Save** to create the `memory_tester_ subsystem.v` file in the `<project folder>`.

The **Export** tab allows you to export the interfaces and requirements to an `.ipxact` or a `_hw.tcl` file, however, this feature is not used in this project.

Figure 26. Create Platform Designer System Template and HDL Template Options



Related Information

Implement the Memory Tester Subsystem on page 31

Connect and Generate IP Files

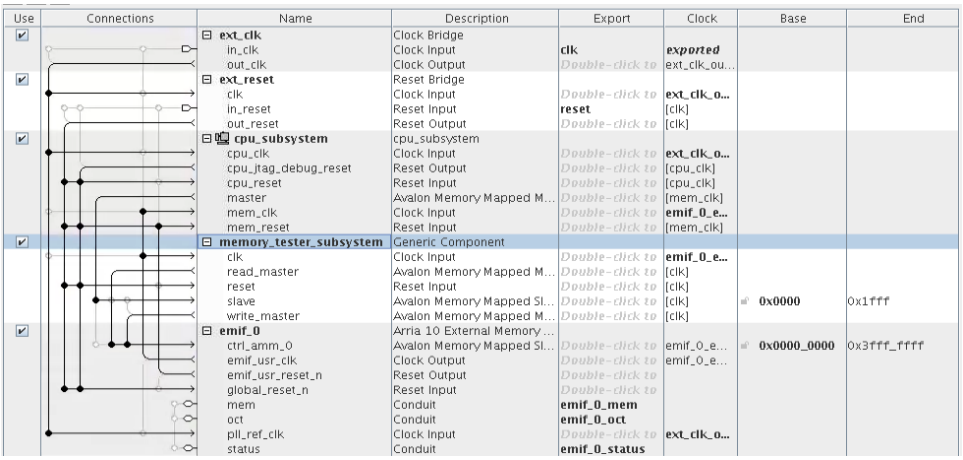
You can make connections once the component instantiation is complete. Connect the source and target components with the entries in the following table:

Table 4. Top Level Platform Designer Connections

Source Component/Signal	Target Component/Signal
ext_clk/out_clk	• **ext_reset**/clk • **cpu_subsystem**/cpu_clk • **emif_0**/pll_ref_clk
ext_reset/out_reset	• **cpu_subsystem**/cpu_reset • **cpu_subsystem**/mem_reset • **memory_test_subsystem**/reset • **emif_0**/global_reset_n
cpu_subsystem/master	• **memory_test_subsystem**/slave
cpu_subsystem/cpu_jtag_reset	• **cpu_subsystem**/cpu_reset • **cpu_subsystem**/mem_reset • **memory_test_subsystem**/reset • **emif_0**/global_reset_n
memory_test_subsystem/read_master	• **emif_0**/ctrl_amm_0
memory_test_subsystem/write_master	• **emif_0**/ctrl_amm_0
emif_0/emif_usr_clk	• **cpu_subsystem**/mem_clk • **memory_test_subsystem**/clk
emif_0/emif_usr_reset_n	• **cpu_subsystem**/mem_reset

Compare your completed system to the following figure:

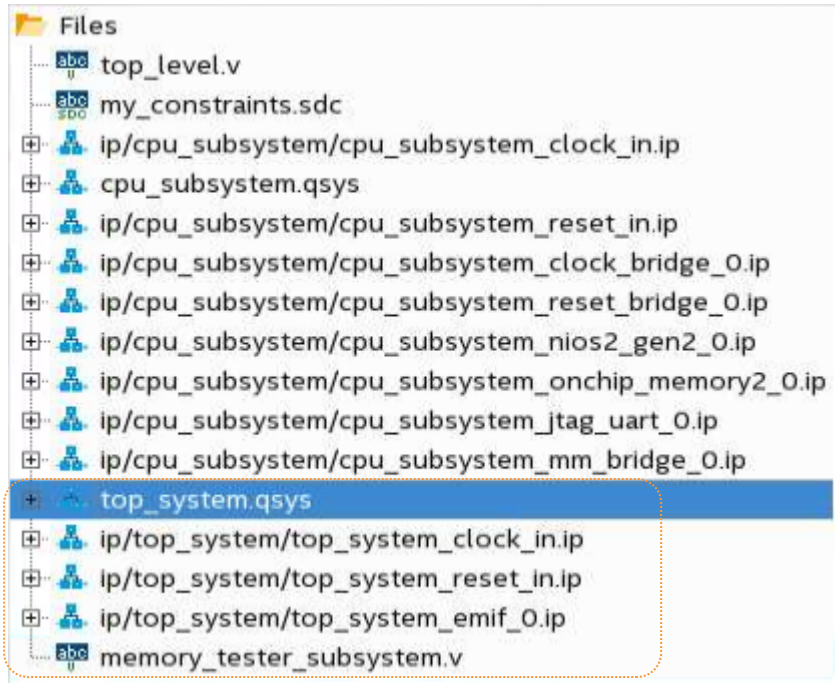
Figure 27. Top Level Platform Designer System Connections



If there are any errors, read the error message and fix the error.

1. Click **File ➤ Save** to save the top-level system.
2. Click **Generate ➤ Generate HDL** and click **Generate** to generate RTL for each component, including components in the **cpu_subsystem**.
3. Close Platform Designer. New files appear in the in the **Project Navigator ➤ Files** tab in the Intel Quartus Prime project. You must add another file `memory_tester_subsystem.v`. Adding this provides an empty entity for `memory_tester_subsystem` so Intel Quartus Prime Pro Edition can elaborate the hierarchy.
4. In the **Tasks** window, double-click **Add/Remove Files in Project** to open the **Settings** dialog box.
5. To add an empty `memory_tester_subsystem.v` file, type `memory_tester_subsystem.v` in the **File name** box.
6. Click **Add**.

Figure 28. top_system.qsys IP Files



7. Compile the project by clicking **Processing ➤ Start Compilation**. If there are any errors, verify that all required files are present, and that you correctly name the exported ports in the Platform Designer system.

After compilation completes successfully, check the Compilation Reports (**Processing ➤ Compilation Report**) for **Logic Resource Usage**, **I/O Bank Usage**, **Clock tree**. You can also upload the `A10.sof` file generated during compilation to a board to check the calibration status of the DDR4 RAM. In the `top_level.v` file, `sdram_cal_success`, and `sdram_cal_fail` are connected to LED3 on the board. A green light indicates that calibration was successful. A red light indicates that calibration failed.

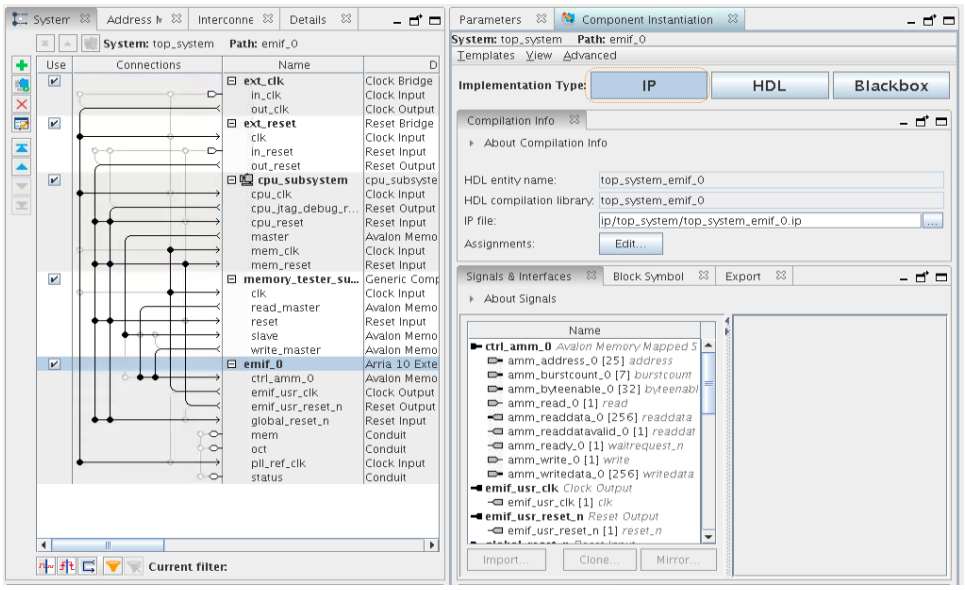
This design flow allows you to verify and debug DDR4 RAM calibration, while maintaining the system structure, before finishing the implementation of the memory tester subsystem.

Examples of Platform Designer Generic Components

You can instantiate components in Platform Designer using generic components.

Generic components fall into one of the three implementation types: **IP**, **HDL**, or **Blackbox**. Each type is selectable by the corresponding button in the **Component Instantiation** tab. All the `_hw.tcl` based IP components found in the IP Catalog, such as On-chip Memory and External Memory Interfaces (EMIF), belong to the IP type. If you want to add a custom component written in RTL, you can use the HDL type and link the source files in the **Component Instantiation** tab.

Figure 29. Example of an IP Component Instantiation



Implement the Memory Tester Subsystem

Next, you implement the memory tester subsystem (previously instantiated as a generic component) using the Platform Designer template.

You typically perform this process as a member of a remote team with a need to implement the memory tester subsystem. The remote team member receives a `.qsys` file which serves as the requirement hand off for an implementation. This `.qsys` file contains the details needed for designing a block for the larger design, without access to the top level.

To implement the memory tester subsystem you must add components from the IP Catalog to this Platform Designer project, make connections, and export interfaces to match what is defined for the generic component. Once those processes are complete, replace the generic component in the top level system with this subsystem implementation.

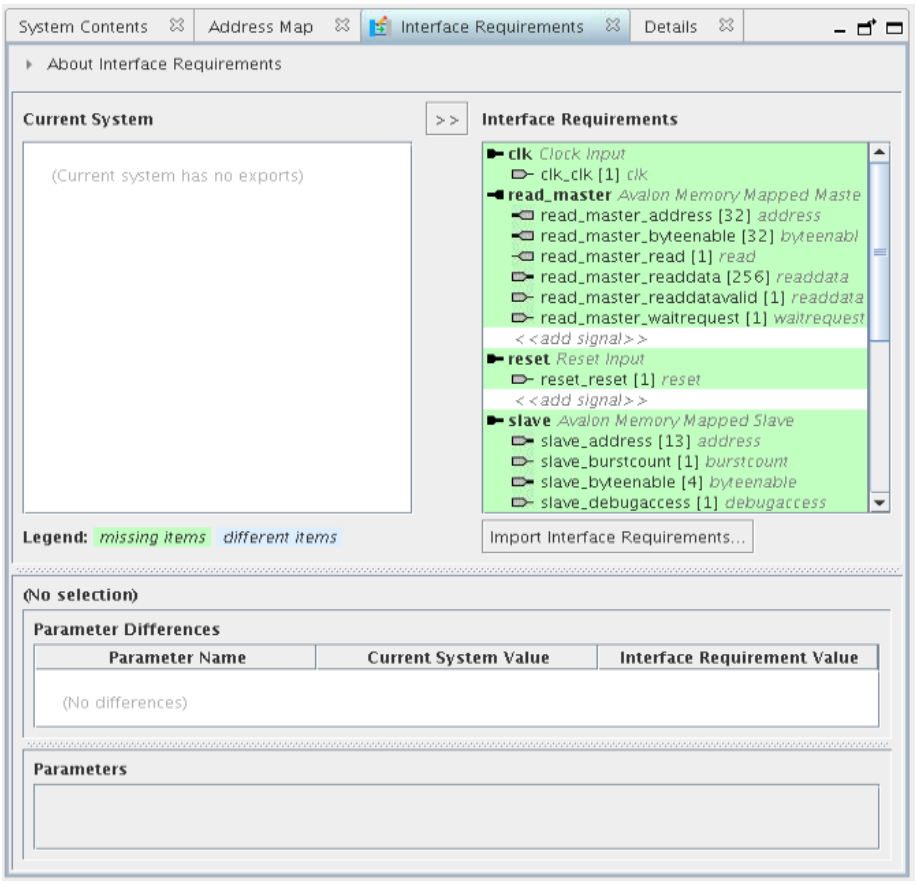
To implement the subsystem, complete the following steps:

1. To launch Platform Designer, click **Tools ➤ Platform Designer**.
2. Browse to the `memory_tester_subsystem.qsys` file and click **Open**.

   Platform Designer opens and displays an empty project. However, it embeds the interface requirements you defined in the generic component representation within the top level system used as a guide to implement this subsystem.
3. To view these interfaces, click **View ➤ Interface Requirements**.

   The left column shows the interfaces instantiated in the current Platform Designer (Standard) Pro system. The right column shows the requirements you define in previous steps. Since there are no components or exported interfaces, all the interface names are highlighted in green, denoting missing items.

Figure 30. Interface Requirements Dialog Box



Add Clock, Reset, and Avalon-MM components

In order to resolve missing items in the **Interface Requirements** list, add a clock bridge, reset bridge, and Avalon-MM pipeline bridge first:

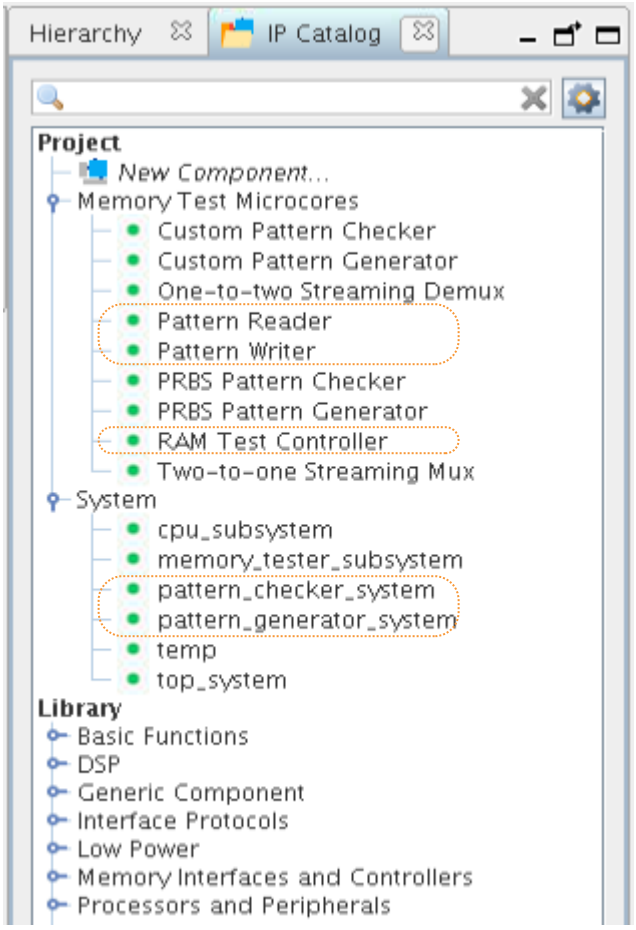
1. In the IP Catalog, type `clock` in the search box and double-click **Clock Bridge**.
2. Click **Finish** to add the clock bridge.
3. Type `reset` in the search box and double-click **Reset Bridge**.
4. Click **Finish** to add the reset bridge.
5. In the System Components tab, right-click the name of the clock bridge and click **Rename**. Type `clk`.
6. In the System Components tab, right-click the name of the reset bridge and click **Rename**. Type `reset`.
7. In the **Export** column, double-click the entry corresponding to the `clock input` for the `clk` component and rename it `clk`.
8. In the **Export** column, double-click the entry corresponding to the `reset input` for the `reset` component and rename it `reset`.

9. Type `pipeline` in the IP Catalog search box and double-click **Avalon-MM Pipeline Bridge**.
10. When the **Avalon-MM Pipeline Bridge** parameter editor opens, in **Address** , set the **Address width** to 13.
11. To add the component, click **Finish**.
12. Right-click the name of the **Avalon-MM Pipeline Bridge** and click **Rename**. Type `mm_bridge`.

Add Pre-Built Systems and Memory Test Microcore Components

The files listed in the *Design Files* topic contain two pre-built systems: a pattern checker system, and a pattern generator system. These pre-built Platform Designer systems appear in the IP Catalog in the System folder. The IP Catalog also contains a list of available Memory Test Microcores. The source files of these custom IP cores are located in `<project folder>/memory_tester_ip`. The `memory_tester_search_path.ipx` file included in the project provides this path to Platform Designer.

Figure 31. IP Catalog Memory Test Microcore and System Components



To add these pre-built Platform Designer systems, complete the following steps:

1. In the IP Catalog, expand the **System** folder and double-click **pattern_checker_system** to add the pattern checker component.
2. To add the component, click **Finish**.
3. To rename the pattern checker, right-click the system in the **Name** column and type `pattern_checker_subsystem`.
4. In the IP Catalog, double-click **pattern_generator_system** to add the prebuilt pattern generator component.
5. To add the component, click **Finish**.
6. To rename the pre-built pattern generator, right-click the system in the **Name** column and type `pattern_generator_subsystem`.
7. In the IP Catalog, expand the **Memory Test Microcores** folder, and double-click **Pattern Writer**.
8. In the **Pattern Writer** parameter editor, turn on **Burst Enable**.
9. To add the component, click **Finish**.
10. To rename the pattern writer component, right-click the system in the **Name** column and type `pattern_writer`.
11. In the IP Catalog, double-click **Pattern Reader**.
12. In the **Pattern Reader** parameter editor, turn on **Burst Enable**.
13. Click **Finish** to add the component.
14. In the IP Catalog, double-click to begin adding a **RAM Test Controller**.
15. Click **Finish** to add the component.

Related Information

Download and Install the Tutorial Design Files on page 5

Export Signals, Set Base Address Assignments, and Connect Memory Tester Interface Components

To export signals, set base address assignments, and connect components, perform the following steps:

1. To export the **Avalon Memory Mapped Master** interface for **Pattern Writer**, in the **Export** column double-click the row adjacent to the **Avalon Memory Mapped Master** and type `write_master`.
2. To export the **Avalon Memory Mapped Master** interface for **Pattern Reader**, in the **Export** column, double-click the row adjacent to the **Avalon Memory Mapped Master** and type `read_master`.
3. Make connections for the system based on the following table:

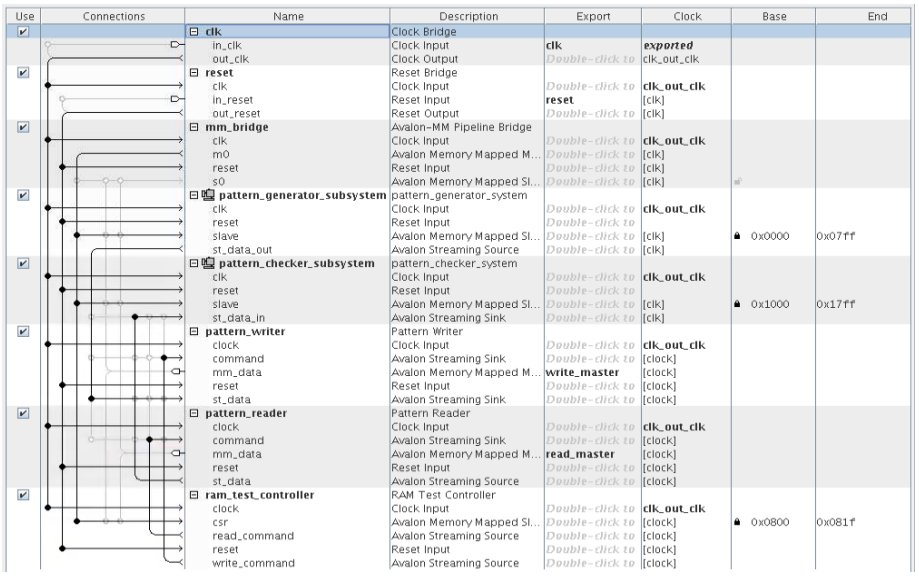
Table 5. Memory Tester Interface Component Connections

Source Compont/Signal	Target Component/Signal
clk/out_clk	• **reset**/clk • **mm_bridge**/clk • **pattern_generator_subsystem**/clk • **pattern_checker_subsystem**/clk • **pattern_writer**/clock • **pattern_reader**/clock • **ram_test_controller**/clock
reset/out_reset	• **mm_bridge**/reset • **pattern_generator_subsystem**/reset • **pattern_checker_subsystem**/reset • **pattern_writer**/reset • **pattern_reader**/reset • **ram_test_controller**/reset
mm_bridge/m0	• **pattern_generator_subsystem**/slave • **pattern_checker_subsystem**/slave • **ram_test_controller**/csr
pattern_generator_subsystem/st_data_out	• **pattern_writer**/st_data
pattern_reader/st_data	• **pattern_checker_subsystem**/st_data_in
ram_test_controller/read_command	• **pattern_reader**/command
ram_test_controller//write command	• **pattern_writer**/command

4. Assign base addresses for **Avalon Memory Mapped Slave** interfaces:
   a. In the **Base** column, click the value for slave signal of the **pattern_generator_subsystem** component and type 0000.
   b. In the **Base** column, click the value for slave signal of the **pattern_checker_subsystem** component and type 1000.
   c. In the **Base** column, click the value for csr signal of the **ram_test_controller** component and type 800.

   The following figure shows the completed system:

Figure 32. Connections and Base Address Values for the memory_tester_sybsystem



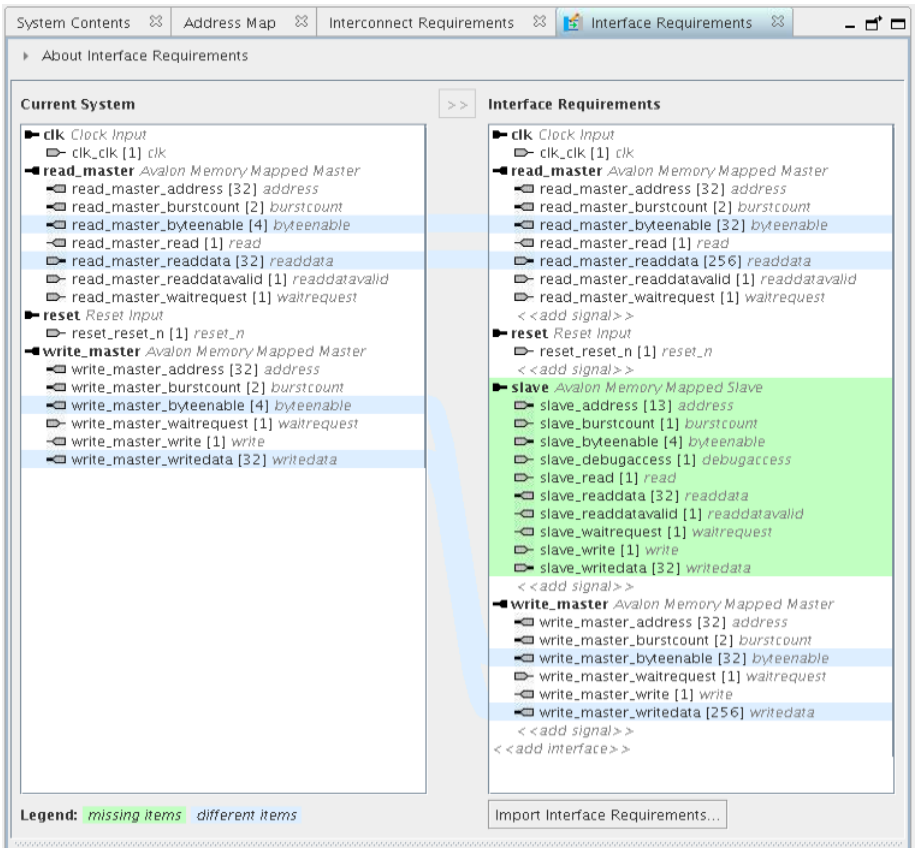
Resolve Interface Requirements and Value Mismatches

In the **Interface Requirements** tab you can verify that the exported interfaces meet the interface requirements.

1. Click the **Interface Requirements** tab in Platform Designer.

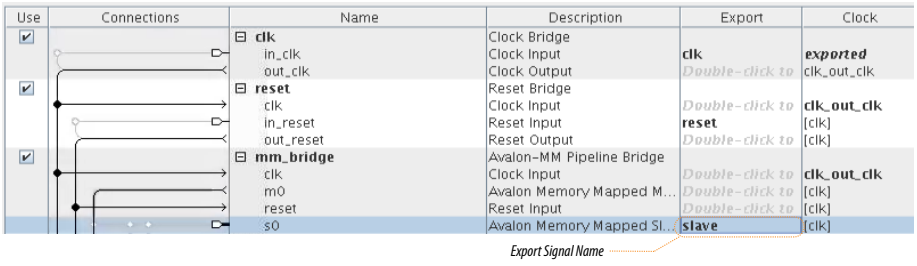
   The exported interfaces in the tutorial system appear in the **Current System** list. The **Interface Requirements** list shows the definition of the generic component. A green highlight indicates a missing item. A blue highlight indicates an item with parameter mismatches.

Figure 33. Missing Components and Value Mismatches



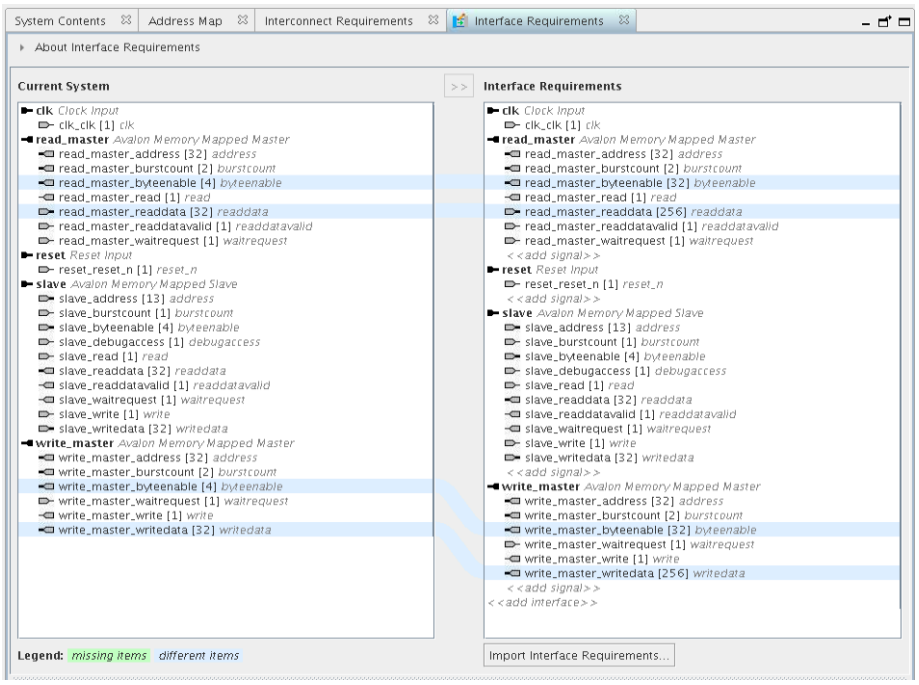
2. View the **Interface Requirements** list for missing items. What appears in the figure indicates a missing `slave` interface of the pipeline bridge. Fix the missing items by exporting the appropriate signal.
3. In the **System Contents** tab, double-click the entry in the **Export** column corresponding to the `s0` for the **mm_bridge** component and rename it to `slave`.

Figure 34. Export and Rename Avalon-MM Slave



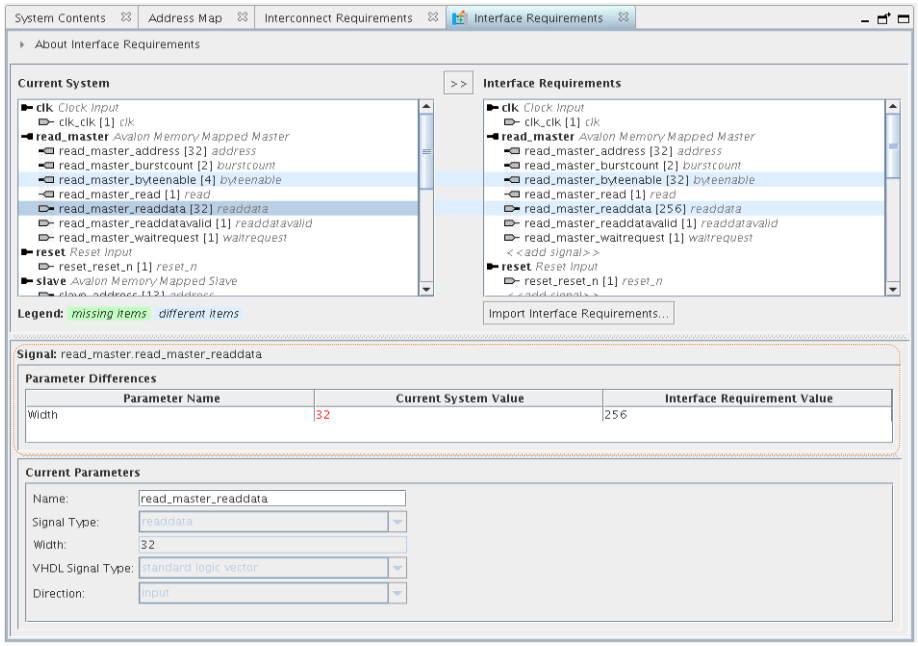
4. Re-examine the **Interface Requirements** tab. The **Current System** list contains the slave interface with no green highlight. Next you resolve the different item highlighted in blue.

Figure 35. Current System / Interface Requirement Value Mismatch



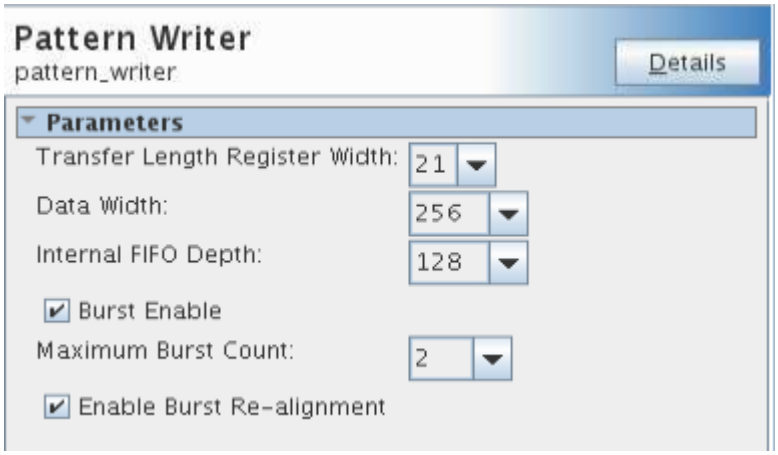
5. Click the signal name highlighted in blue to display more information in the **Parameter Differences** pane. Typically, you change the **Current System Value** to match the **Interface Requirement Value** by editing the parameters of that component.

Figure 36. Changing Current System Value



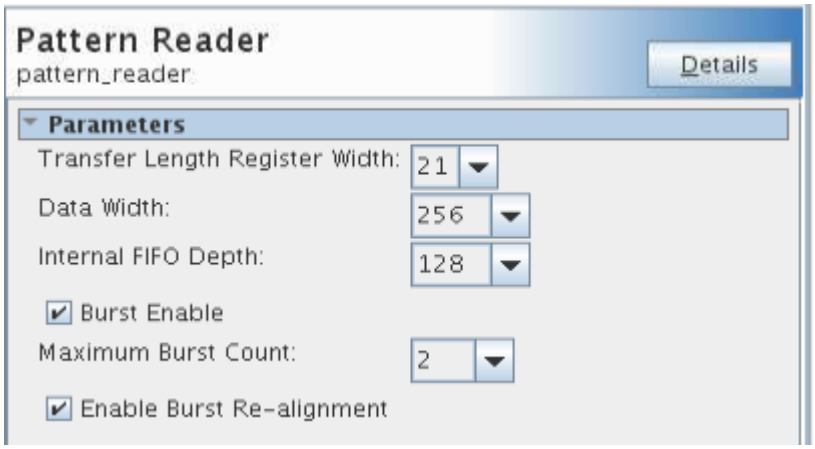
6. Click **read_master_readdata[32]** and examine the **Parameter Differences** pane. In the `top_system`, the data width of the **Avalon Memory Mapped Master** of the EMIF controller is 256. The data width of the **memory_tester_subsystem** must match with a value of 256. Adapters inserted to handle data width mismatch may become the bottle-neck of a design.
7. This exported interface comes from the Pattern Writer. To alter its width, alter the parameters of that IP core. To change the data width of **Pattern Writer**, double-click the **pattern_writer** component. Change the **Data Width** in the parameter editor to 256.

Figure 37. Pattern Writer Settings Dialog Box



Repeat the Step 7 for the **pattern_reader** component.

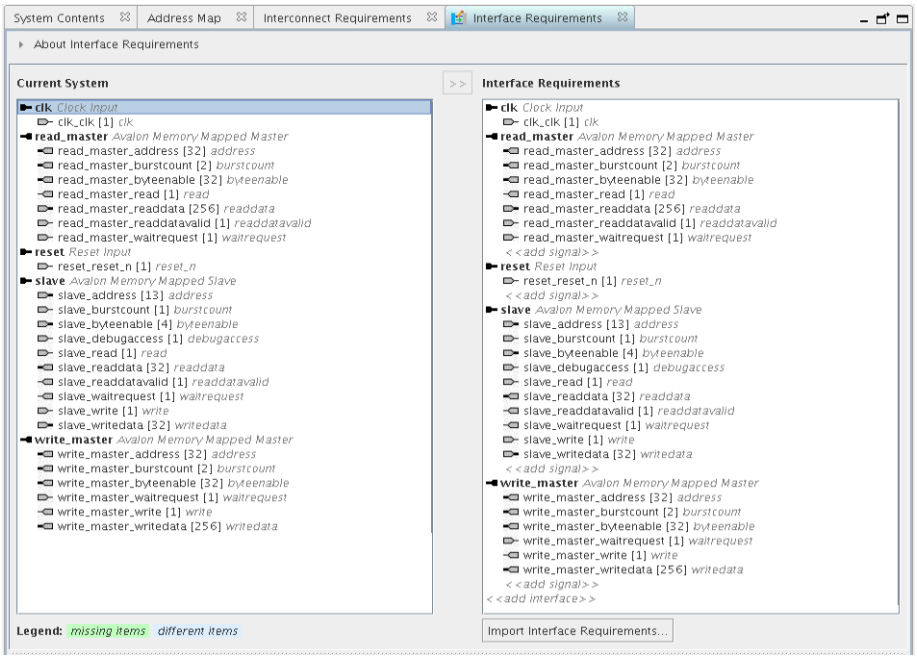
Figure 38. Pattern Reader Settings Dialog Box



These parameter changes alter the width of `*_byteenable` signals accordingly.

8. Verify that your Interface Requirements tab contains no missing items or mismatched items. In cases where you want to keep the current system value, you can click the **Copy** button to copy items from the left table to the right.

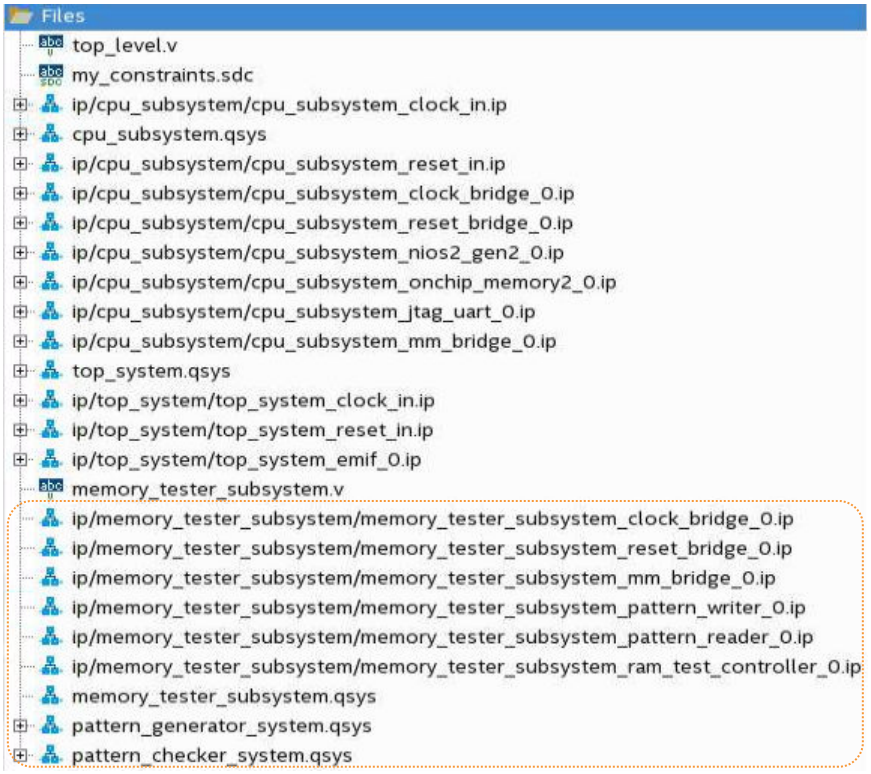
Figure 39. Completed Interface Requirements



This completes the editing of component parameters to validate interface requirements.

9. Save and close the project. There is no requirement to generate HDL because we are replacing the generic component in `top_system.qsys` with the implemented subsystem.
10. Close Platform Designer and inspect the **Files** tab in the Project Navigator. Files for the `memory_tester_subsystem` are present in the Intel Quartus Prime Pro Edition project.

Figure 40. Files List for memory_tester_subsystem.v



Replace the memory_tester_subsystem Generic Component

Next, replace the generic component with the `memory_tester_subsystem` implementation:

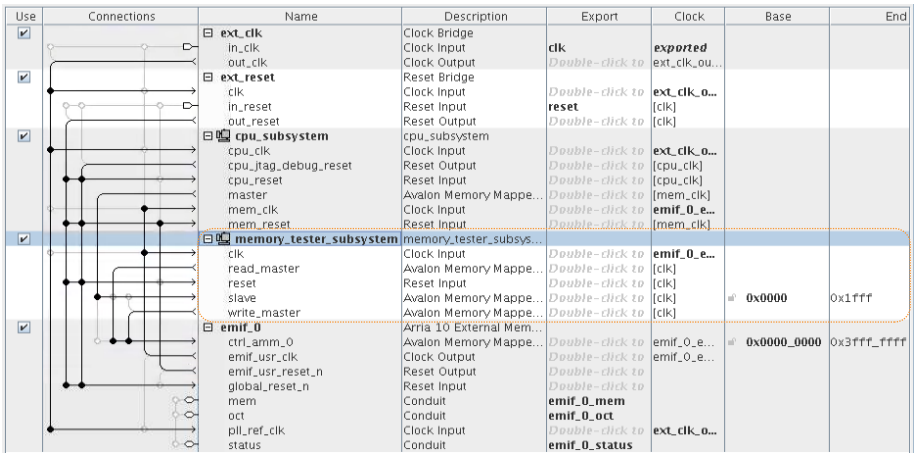
1. Click **Tools ➤ Platform Designer** to launch Platform Designer. Browse to the `top_system.qsys` file and click **Open**.
2. Right-click the `memory_tester_subsystem` component and click **Remove**.
3. In the IP Catalog, browse to the **System** folder and double-click to `memory_tester_subsystem`. Keep the same name and update the connections.
4. Right-click the name of the `top_system_subsystem_0` and click **Rename**. Type `memory_tester_subsystem`.
5. Verify and complete connections based on the following table:

Source Component/Signal	Target Component/Signal
ext_clk/`out_clk`	• **ext_reset**/`clk` • **cpu_subsystem**/`cpu_clk` • **emif_0**/`pll_ref_clk`
ext_reset/`out_reset`	• **cpu_subsystem**/`cpu_reset` • **cpu_subsystem**/`mem_reset` • **memory_tester_subsystem**/`reset` • **emif_0**/`global_reset_n`

Source Component/Signal	Target Component/Signal
cpu_subsystem/cpu_jtag_debug_reset	• **cpu_subsystem**/cpu_reset • **cpu_subsystem**/mem_reset • **memory_tester_subsystem**/reset • **emif_0**/global_reset_n
cpu_subsystem/master	• **memory_tester_subsystem**/slave
memory_tester_subsystem/read_master	• **emif_0**/ctrl_amm_0
memory_tester_subsystem/write_master	• **emif_0**/ctrl_amm_0
emif_0/emif_usr_clk	• **cpu_subsystem**/mem_clk • **memory_tester_subsystem**/clk
emif_0/emif_usr_reset_n	• **cpu_subsystem**/mem_reset

6. Compare the connections to the following figure:

Figure 41. memory_tester_subsystem Implementation Connections



7. Click **File ➤ Save**.
8. Click **Generate ➤ Generate HDL**.
9. Click **Generate**.
10. Close the current Platform Designer project when generation is done.

The files in included with this design are Verilog (`.v`) files, but you can also use VHDL (`.vhdl`) in your design if you prefer.

Synchronize IP Results

When you synchronize IP files, Platform Designer checks IP file references.

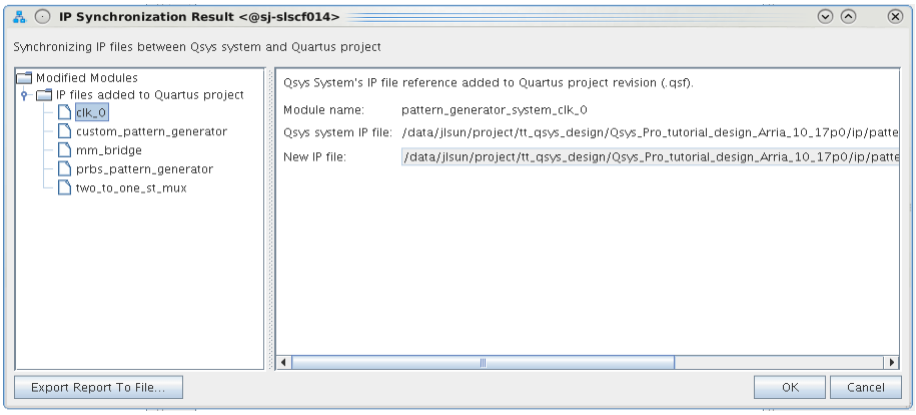
1. In the Intel Quartus Prime Pro Edition, click **Files** in the Project Navigator and browse to `memory_tester_subsystem.v`.
2. Delete the empty entity RTL `memory_tester_subsystem.v` since we now have the actual `memory_tester_subsystem` implementation.

The files included in `memory_tester_subsystem` are not complete though. We are missing IP components in the pattern generator system and pattern checker system.

3. Open the `pattern_generator_system.qsys` and `pattern_checker_system.qsys` in Platform Designer and save them without generating HDL. This designates the IP components in these systems for elaboration during compilation.

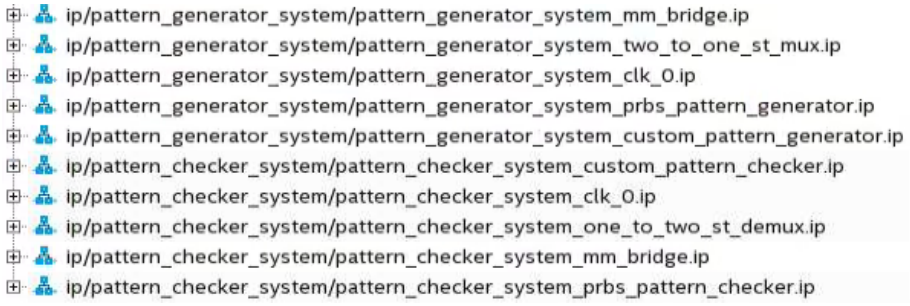
   Each time you open a Platform Designer project, Platform Designer automatically checks the IP file references and opens a dialog box if there is any mismatch. In the following figure, **IP Synchronization** detects the Platform Designer system includes these IP, but the Intel Quartus Prime Pro Edition project does not. This dialog box informs you when you must add these files to the project..

Figure 42. IP Synchronization Dialog Box



4. Click **OK** and the Intel Quartus Prime Pro Edition synchronizes the file references.
5. Examine the Project Navigator and these new files appear:

Figure 43. Files Added through IP Synchronization



6. Click **Processing ➤ Start Compilation** to compile the project. The Intel Quartus Prime Pro Edition software may return missing file errors, for example:

```
"Instance ‘ abc|def|ghi ’ instantiates undefined entity ‘ xyz ’ "
```

This type of error is caused when an expected IP file is missing. Resolve it by adding the `xyz.ip` file to the project.

Build Software Applications and Download the Design

The final steps in this tutorial show how to download your design onto a Dev Kit board.

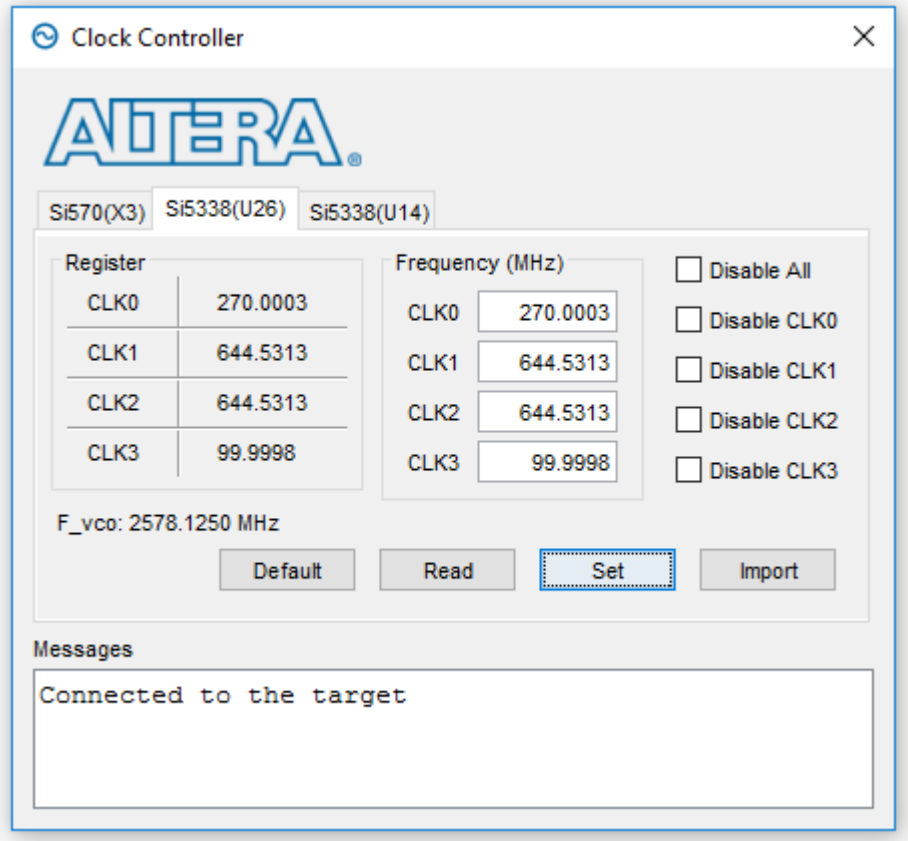
Hardware setup

First, you reprogram the clock generator chip on the board.

The default clock resource in this design runs at 133.33 MHz. Program the clock to run at 100 MHz.

1. Connect the board to the host PC with a USB cable and apply power to the board.
2. Run the `ClockController.exe` executable that installs with the Dev Kit package. This executable installs to `<package installation folder>/examples/board_test_system` by default.
3. Click the **Si5338(U26)** tab.
4. Change the frequency setting for CLK3 to 100MHz.
5. Click **Set**.

Figure 44. Clock Controller Settings



Related Information

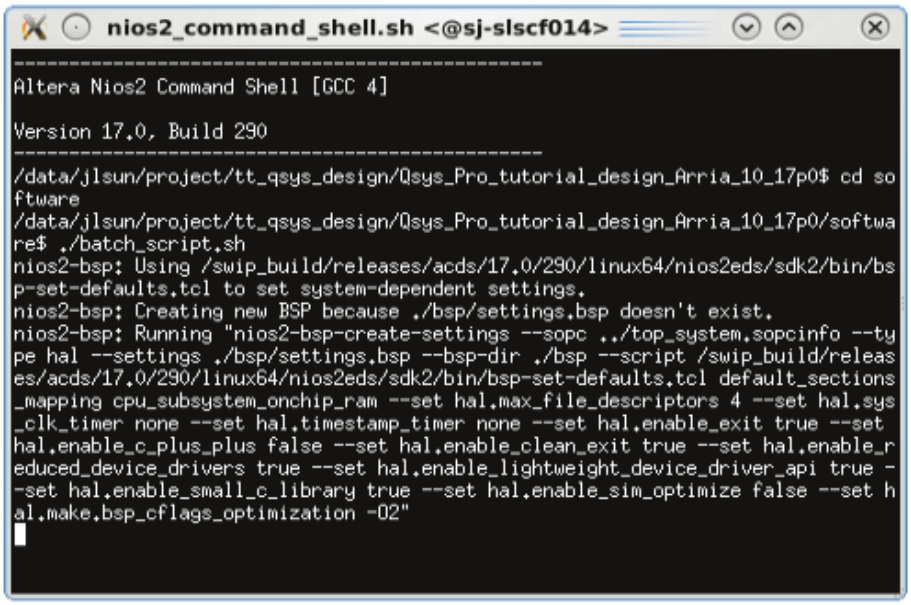
Hardware and Software Requirements on page 5

Run the Bash Script

Nios II EDS enables you to build board support packages (device drivers, HAL) and applications based on the `top_system.sopcinfo` file, an output file of `top_system.qsys` generation.

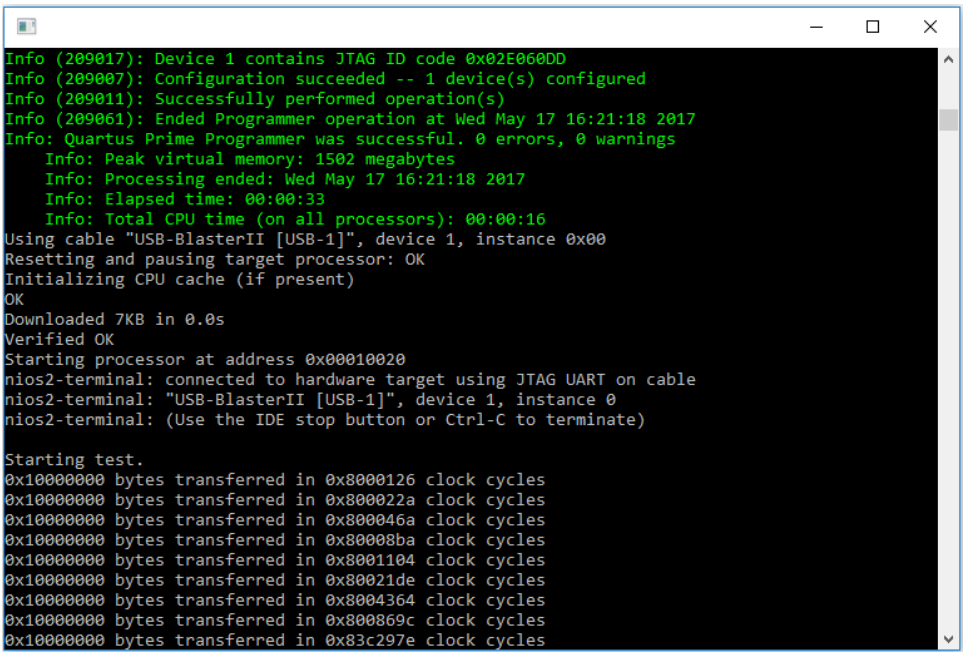
1. Copy `top_system.sopcinfo` from `/top_system` to `/<project folder>`.
2. To launch the **Nios II Command Shell** from Platform Designer, click **Tools ➤ Nios II Command Shell (gcc4)**.
3. In the Nios II Command Shell, browse to `<project folder>/software`, and run `batch_script.sh`.

Figure 45. Run the nios2_command_shell.sh



The `batch_script.sh` script calls commands in Nios II EDS to build a board support package and applications. The script then configures the FPGA with the `A10.sof` file that you generate during Intel Quartus Prime software compilation, runs the software applications, and establishes a terminal connection with the board. The test software performs test sweeps, such as Walking Ones, Walking Zeros, and PRBS, on the SDRAM and the output values appear in the command terminal.

Figure 46. Terminal Connection Console



The `<project folder>/software` folder contains a `rerun.sh` script. You can run this script when you already have the Nios II board support package and applications built, and don't need to build them again. This script downloads only the `.sof` file and runs Nios II applications.

AN 812: Platform Designer System Design Tutorial Revision History

Document Version	Changes
2018.04.02	Updated for terminology change from Qsys Pro to Platform Designer.
2018.05.04	Updated
2017.08.15	Initial release.