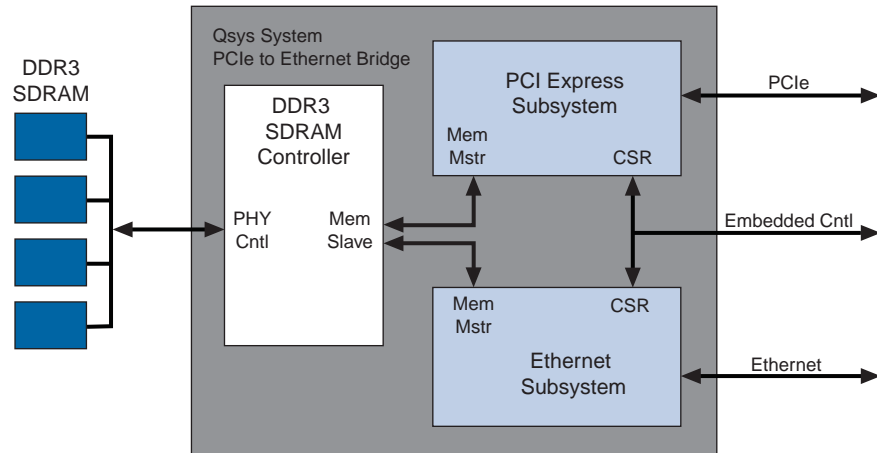


Qsys is a powerful system integration tool included as part of the Quartus® II software. Qsys captures system-level hardware designs at a relatively high level of abstraction and also automates the task of defining and integrating customized HDL components, which may include IP cores, verification IP, and other design modules. Qsys facilitates design reuse by packaging and making available your custom components and systems. Qsys integrates your custom components with Altera® and third-party developer components. In some cases, you can implement an entire design using components from the Altera component library. During system generation, Qsys automatically creates high-performance interconnect logic from the connectivity options you specify, eliminating the error-prone and time-consuming task of writing HDL to specify the system-level connections.

Qsys provides the following advantages for hardware system design:

- Automates the process of customizing and integrating components
- Supports modular system design
- Supports visualization of large systems
- Supports optimization of interconnect fabric and pipelining within the system
- Provide full integration with the Quartus II software

Qsys supports hierarchical system design. You can include any Qsys system that exports an interface as a component in another Qsys system. [Figure 5–1](#) shows the top-level of a Qsys example design that implements a PCI Express™ to Ethernet bridge. This example combines separate PCI Express and Ethernet subsystems with Altera’s DDR3 SDRAM Controller with UniPHY IP core. Different members of the design team could develop the various subsystems simultaneously, decreasing time-to-market for the complete design. For a detailed discussion of this example design, refer to [“Example Hierarchical System” on page 5–20](#).

Figure 5-1. Top-Level Block Diagram for a PCI Express to Ethernet Bridge

Hierarchical system design in Qsys offers the following advantages:

- Enables team-based, modular design by dividing large designs into subsystems
- Enables design reuse by allowing you to use any Qsys system as a component
- Enables scalability by allowing you to instantiate multiple instances of a Qsys system

Qsys supports enhanced component parameterization, allowing you to design for maximum efficiency and utility. For example, you can specify parameters that can be fed to a program that generates the RTL at run time, allowing unlimited parameterization.

This chapter introduces Qsys in the following sections:

- [“Overview of Qsys” on page 5-2](#)
- [“Qsys Design Flow” on page 5-10](#)
- [“Simulating a Qsys System” on page 5-14](#)
- [“System Examples” on page 5-20](#)
- [“Document Revision History” on page 5-26](#)

Overview of Qsys

You can create a new Qsys system in the Quartus II software by clicking **Qsys System File** in the **New** dialog box, and open a Qsys system from the File menu. Alternatively, you can launch Qsys from the Tools menu by clicking **Qsys**.

- ❓ For more information about the Qsys GUI, refer to [About Qsys](#) in Quartus II Help.

For more information about the benefits of using Qsys, refer to the video [Five Reasons to Switch from SOPC Builder to Qsys](#) on the **Webcasts and Videos** page of the Altera website.

Qsys Component Library

The Qsys component library contains all the components found on the component search path that you specify, whether or not they are included in a Quartus II project. These components include Altera-provided IP cores, third-party IP cores, and custom IP cores that you provide. Previously created Qsys systems are also listed in the component library and can be used in other designs if they have exported interfaces. The component library also includes all the components that are used in the Qsys interconnect.

Altera recommends that you use standard Avalon interfaces in your component designs. By using these standard interfaces, you can create components that interoperate with the components in the Qsys component library. In addition, you can take advantage of bus functional models (BFMs), monitors, and other verification IP when verifying your design. However, Qsys allows you to design with any interface that your design requires. If a set of signals cannot adhere to the *Avalon Interface Specifications*, you can encapsulate the signals as a conduit interface. You can connect conduit interfaces inside of Qsys, or export them for connection outside of the immediate module.



For more information about all interface types, refer to the *Avalon Interface Specifications*.



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

Altera and third-party developers provide ready-to-use Qsys components. The component library has many different types of components including all of the following:

- Microprocessors, such as the Nios® II processor
- DSP IP cores, such as the Reed Solomon II
- Interface protocols, such as the IP Compiler for PCI Express
- Memory controllers, such as the RLDRAM II Controller with UniPHY
- Avalon® Streaming (Avalon-ST) components, such as the Avalon-ST Multiplexer IP core
- Qsys interconnect components, such as the Qsys master router, which decodes addresses


These components are installed automatically with the Quartus II software, and are available in the Qsys component library.

Integrating Custom Components

Use the following steps to integrate your custom components into a Qsys system:


1. Determine the interfaces that interact with your custom component.
2. Create the component logic using either Verilog HDL or VHDL.
3. Use the Qsys component editor to define the Hardware Component Description File (**_hw.tcl**).
4. Instantiate the component in the system.

Once you have created a Qsys component, you can use the component in other Qsys systems and share the component with other design teams.

-  For instructions on developing a custom Qsys component, details about the file structure of a component, or the Qsys component editor, refer to the *Creating Qsys Components* chapter in volume 1 of the *Quartus II Handbook*.

Integrating Third-Party Components

You can also use Qsys components created by third-party IP developers. Altera awards the Qsys Compliant label to IP cores that are fully supported in Qsys. These cores support Avalon interfaces and may include timing and placement constraints, software drivers, simulation models, and reference designs.


-  You can find Qsys components on the [Intellectual Property & Reference Designs](#) web page. When browsing designs by category, look for a checkmark in the Qsys Compliant column, or you can type `Qsys Compliant` in the **Search** and **IP Cores & Reference Designs**. You can also conduct advanced searches by clicking **Launch the Altera Product Selector Guide** from the [Altera Product Selector](#) web page, and then clicking the **IP Selector** tab.

Adding System Contents

The **System Contents** tab displays the components that you add to your system.

Adding Components

To add a component to your system, click the component in the **Component Library**, and then click the **Add**. A parameter editor appears allowing you to customize the component.

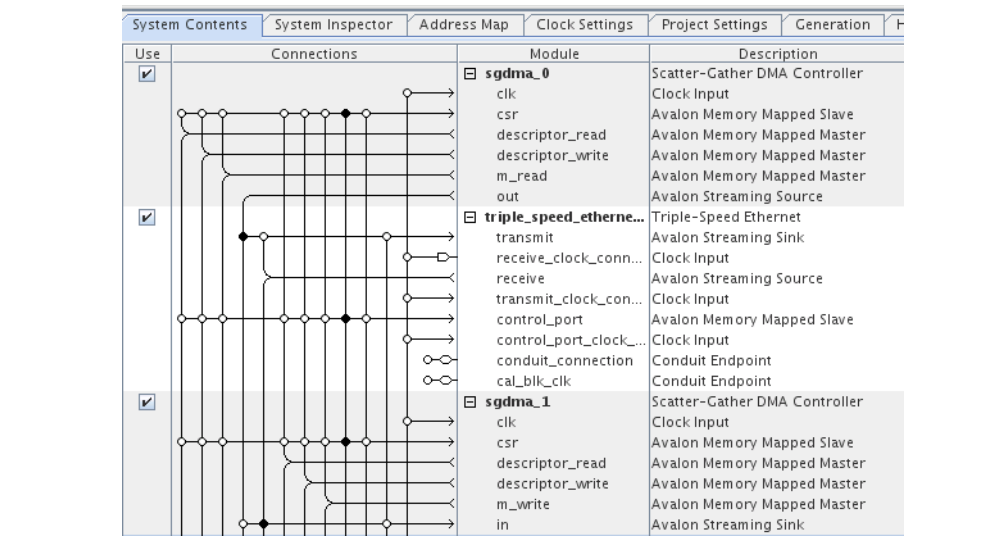
-  You can type some or all of the component's name in the **Component Library** search box to display all components including that string.

Connecting Components

You specify connections at the interface level and Qsys automatically connects the individual signals within connected interfaces. You connect interfaces of compatible types and opposing directions. For example, you can connect an Avalon Memory-Mapped (Avalon-MM) master interface to an Avalon-MM slave interface, and an Avalon Interrupt sender interface to an Avalon Interrupt receiver interface.

To view the possible connections for an interface, click the **System Contents** tab and then click the interface name. Hover your cursor in the **Connections** column. To view the connectivity matrix where open circles represent possible connections and black circles indicate connections that you have made. To make a connection, click the open circle at the intersection of the two interface names. Clicking a second time removes the connection. [Figure 5-2](#) illustrates the connectivity matrix.

Figure 5-2. Connections Column



For more information, refer to [Connecting Qsys Components](#) in Quartus II Help.

Filtering Components

You can use the **Filters** dialog box to filter the display of your system in the **System Contents** tab. You can filter the display of your system by interface type, instance name, or using custom tags. For example, you can use filtering to view only instances that include an Avalon-MM interface, instances that are connected to a particular Nios II processor, or to temporarily remove clock and reset interfaces to simplify the display.

For more information, refer to the [Filters Dialog Box](#) in Quartus II Help.

Using the System Inspector

The **System Inspector** tab displays the underlying model of your complete system. The System Inspector provides comprehensive details about your system such as the following information:

- The connections between all signals.
- The names of all signals included in exported interfaces.
- The internal connections of Qsys subsystems that are included as components.

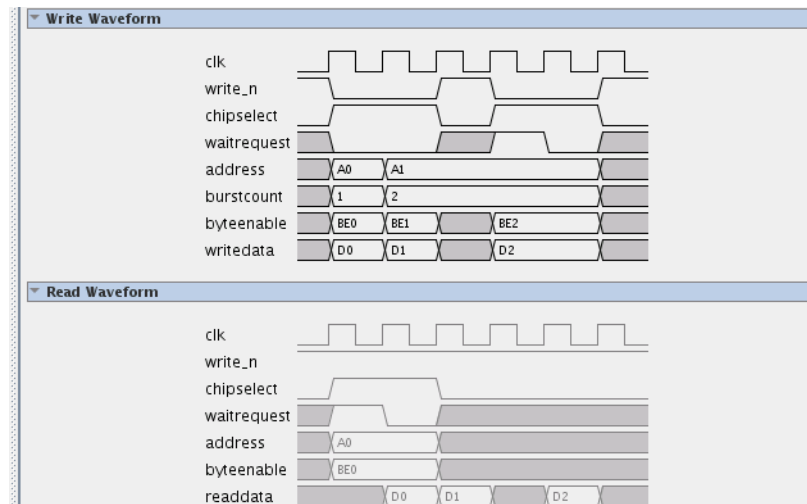



In contrast, the **System Contents** tab displays only the exported interfaces of Qsys subsystems included as components.

- The global parameter settings that you specified on the **Project Settings** tab.

You can use the **System Inspector** tab to review and change component parameters and to review interface timing. For example, [Figure 5-3](#) shows the timing for the Avalon-MM DMA write master for the PCI Express-to-Ethernet system illustrated in [Figure 5-8 on page 5-21](#).

Figure 5-3. Avalon-MM Write Master Timing Waveforms Available on the Project Settings Tab




 To display the timing for an interface, expand the component to display its interfaces and click the interface name.

Defining the Address Map

The **Address Map** tab provides a table including all the Avalon-MM slaves in your design and the address range that each connected Avalon-MM master uses to address that slave. The table shows the slaves on the left and masters across the top, with the address span of the connection shown in each cell. A blank cell implies that there is no connection between that master and slave.

Follow these steps to change or create a connection between master and slave components:

1. In Qsys, click the **Address Map** tab.
2. Locate the table cell that represents the connection between the Avalon-MM master and slave component pair.
3. Either type in a base address or update the current base address in the cell.

 You can design a system where two Avalon-MM masters access an Avalon-MM slave at different addresses. If you use this feature, the **Base** and **End** address columns of the **System Contents** tab are labeled *mixed* rather than providing the address range.

Specifying Clock Settings

You can use the **Clock Settings** tab to define the clocks in your system. The **Clock Settings** tab defines the **Name**, **Source**, and frequency (**MHz**) of each clock. Clicking the **Add** button adds a new clock. To change the default values, click in the appropriate column, backspace, and type the new value.

- ❓ For more information, refer to the [Adding Components to a Qsys System \(To define clock domains in a system\)](#) in Quartus II Help.

Specifying Project Settings

You can use the **Project Settings** tab to view and change the properties of your Qsys system. [Table 5-1](#) describes system-level parameters available on the **Project Settings** tab.

Table 5-1. Project Settings Parameters

Parameter Name	Description
Device Family	Specifies the Altera device family. If your final design targets a HardCopy® series device, specify that device here.
Clock Crossing Adapter Type	<p>Specifies the default implementation for automatically inserted clock crossing adapters. The following choices are available:</p> <ul style="list-style-type: none"> ■ Handshake—This adapter uses a simple hand-shaking protocol to propagate transfer control signals and responses across the clock boundary. This methodology uses fewer hardware resources because each transfer is safely propagated to the target domain before the next transfer can begin. The Handshake adapter is appropriate for systems with low throughput requirements. ■ FIFO—This adapter uses dual-clock FIFOs for synchronization. The latency of the FIFO-based adapter is a couple of clock cycles more than the handshaking clock crossing component, but the FIFO-based adapter can sustain higher throughput because it can support multiple transactions in flight at any given time. The FIFO-based clock crossers require more resources. The FIFO adapter is appropriate for memory-mapped transfers requiring high throughput across clock domains. ■ Auto—if you select Auto, Qsys specifies the FIFO adapter for bursting links and the Handshake adapter for all other links.
Limit interconnect pipeline stages to	Specifies the maximum number of pipeline stages in each command and the response path that Qsys may insert to increase the f_{MAX} at the expense of additional latency. You can specify between 0–4 pipeline stages, where 0 means that the interconnect has a combinational datapath. Choosing 3 or 4 pipeline stages may significantly increase the logic utilization of the system. This setting is per Qsys system or subsystem, meaning that each subsystem can have a different setting. Note that this additional latency is for both the command and response directions for the two Qsys systems, even if you combine them into a single Quartus II project.
Generation ID	A unique integer value that is set to a timestamp just before Qsys system generation. The Generation ID is used to check for software compatibility.

Defining Qsys Instance Parameters

The **Instance Parameters** tab allows you to define parameters for a Qsys system. You can use instance parameters to modify a Qsys system when you use the system as a subcomponent in another Qsys system. The higher-level Qsys system can assign values to these instance parameters.

The **Instance Script** on the **Instance Parameters** tab defines how the specified values for the instance parameters should affect your Qsys design subcomponents. The instance script allows you to make queries about the instance parameters you define and set the values of the parameters for the subcomponents in your design.

When you click the **Preview Instance** button, Qsys creates a preview of the current Qsys system with the specified parameters and instance script, and shows the parameter editor for the instance. This allows you to see how an instance of this system appears when you use it in another system. The preview instance does not affect your saved system.

- ❓ For more information, refer to the *Working with Instance Parameters in Qsys* in Quartus II Help.

To use the Instance Parameters, the components or subsystems in your Qsys system must have parameters that can be set when they are instantiated in a higher-level system. Many components in the Component Library have parameters that you can set when adding the component to your system. If you create your own IP components, you use the `_hw.tcl` file to specify which parameters can be set when the component is added to a system. If you create hierarchical Qsys systems, each Qsys system in the hierarchy can include instance parameters to pass parameter values through multiple levels of hierarchy.

- 🔧 For more information on creating your own components and specifying parameters, refer to the *Component Interface Tcl Reference* chapter in the *Quartus II Handbook*.

Creating an Instance Script

The first command in an Instance Script must specify the version of the Tcl commands to be used in the script. This command ensures the Tcl commands behave identically in future versions of the tool. Use the following Tcl command, where `<version>` is a Quartus II software version number, such as 11.1:

```
package require -exact qsys <version>
```

To use the Tcl commands that work with instance parameters in the **Instance Script**, you must specify the commands within a Tcl procedure called a composition callback. In the **Instance Script**, you specify the name for the composition callback with the following command:

```
set_module_property COMPOSITION_CALLBACK <name of callback procedure>
```

Specify the appropriate Tcl commands inside the Tcl procedure with the following syntax:


```
proc <name of procedure defined in previous command> {} {
    #Tcl commands to query and set parameters go here
}
```

Use Tcl commands in the procedure to query the parameters of a Qsys system or to set the values of the parameters of the subcomponents instantiated in the system.

Table 5-2 describes the supported Tcl commands.

Table 5-2. Hardware Tcl Commands Used in Instance Scripts

Command Name	Value	Description
get_parameters	—	Get the names of all defined parameters (as a space-separated list).
get_parameter_value	<parameter name >	Get the value of a parameter.
get_instance_parameters	<instance name>	Get the names of parameters on a child instance that can be manipulated by the parent (as a space-separated list).
get_instance_parameter_value	<instance name>	Get the value of a parameter for a child instance.
send_message	<message level> <message text>	Send a message to the user of the component, using one of the message levels Error, Warning, Info, or Debug. Text with multiple words should be enclosed in quotation marks.
set_instance_parameter_value	<instance name> <parameter name> <parameter value>	Set a parameter value for a child instance.

 For more information about `_hw.tcl` syntax and manipulating parameters, refer to the *Component Interface Tcl Reference* chapter in the *Quartus II Handbook*.

You can use standard Tcl commands to manipulate the parameters in the script, such as the `set` command to create variables, or the `expr` command for mathematical manipulation of the parameter values.

The following example shows the Instance Script of a simple system that uses a parameter called `pio_width` to set the `width` parameter of a parallel I/O (PIO) component in the system. Note that the script combines the `get_parameter_value` and `set_instance_parameter_value` commands into one command using the square brackets `[]`.

Example 5-1. Simple Instance Script

```
# Request a specific version of the scripting API
package require -exact qsys 11.1

# Set the name of the procedure to manipulate parameters:
# set_module_property COMPOSITION_CALLBACK compose

proc compose {} {

    #Get the pio_width parameter value from this Qsys system and pass
    the value to the width parameter of the pio_0 instance

    set_instance_parameter_value pio_0 width [get_parameter_value \
pio_width]
}
```

For another example, refer to “[Example Hierarchical System Using Parameters](#)” on page 5-24.

System Generation

Qsys system generation creates the interconnect between components and generates files that you use to synthesize or simulate the design. You specify the files you want to generate on the **Generation** tab. You can generate simulation models or testbench HDL files for Quartus II synthesis, or a Block Symbol File (**.bsf**) for schematic design. By default, Qsys places these output files in a subdirectory of your project directory, along with an HTML report file. To change the default behavior, click the **Output Path** directory and specify a new directory.

You must add the Quartus II IP File (**.qip**) to your Quartus II project. The **.qip** file is stored in the synthesis directory after generation. It lists the files necessary for Quartus II compilation. The **.qip** file includes references to the following information:

- HDL files used in the Qsys system
- TimeQuest Timing Analyzer Synopsys Design Constraint (**.sdc**) files
- Component definition files for archiving purposes

❓ For more information about adding files to your Quartus II project, refer to *Managing Files in a Project* in Quartus II Help.

Viewing the HDL Example

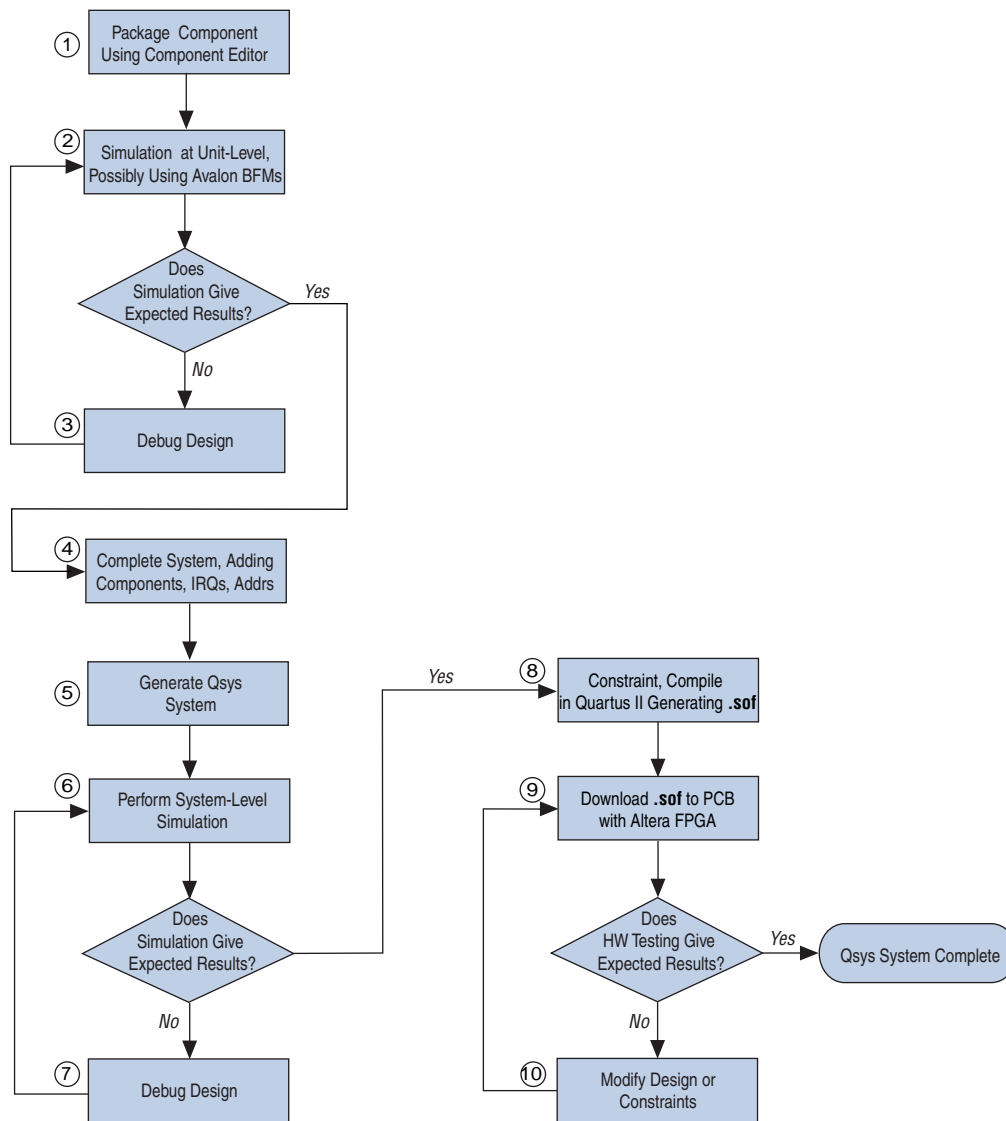
The **HDL Example** tab provides the top-level HDL definition of your Qsys system in either Verilog HDL or VHDL. This tab also displays VHDL component declarations. You can copy and paste the example into a top-level HDL file that instantiates the Qsys system, if the system is not the top-level module in your Quartus II project.


Qsys Design Flow

Figure 5-4 illustrates an example bottom-up design flow in Qsys which starts with component design. As this flow diagram illustrates, the typical design flow includes the following high-level steps:

1. Package your component for Qsys using the Qsys Component Editor.
2. Simulate at the unit-level, possibly incorporating Avalon BFM to verify the system.
3. Complete the Qsys design by adding other components, specifying interrupts, clocks, resets, and addresses.
4. Generate the Qsys system.
5. Perform system-level simulation.
6. Constrain and compile the design.
7. Download the design to an Altera device.
8. Test in hardware.

Figure 5-4. Complete Qsys Design Flow



 In the alternative top-down valid design flow, you begin by designing the Qsys system and then define and instantiate custom Qsys components. This approach clarifies the system requirements earlier in the design process.

Designs targeting HardCopy series devices require specific design constraints. Consequently, if you are targeting a HardCopy series device, you must verify your design for the HardCopy companion device.

Follow these guidelines to verify your design for both devices:

1. In the **Device** dialog box in the Quartus II software, select both the FPGA and the appropriate HardCopy companion device.
2. In Step 8 of the design flow shown in [Figure 5-4](#), compile for both the FPGA and HardCopy device.
3. After Step 10 of the design flow shown in [Figure 5-4](#), if the FPGA passes all functional simulation and hardware verification tests, generate the HardCopy handoff archive file and send this archive file to the HardCopy Design Center for the backend flow implementation.

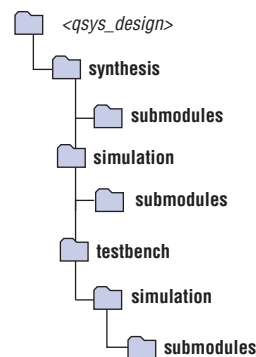
❓ For more information about designing for HardCopy devices, refer to [About Designing HardCopy Devices](#) in Quartus II Help.

Qsys Output Files

Qsys system generation creates output files for synthesis and simulation, as described in “[System Generation](#)” on page 5-10.

[Figure 5-5](#) illustrates the directory structure for the output files.

Figure 5-5. Qsys Generated Files Directory Structure



[Table 5-3](#) describes the files that Qsys generates. Each time you generate your system, Qsys overwrites these files, therefore, you should not edit Qsys-generated output files. If you have additional constraints, such as board-level timing constraints, Altera recommends that you create a separate Synopsys Design Constraints File (.sdc) and include that file in your Quartus II project.

Table 5-3. Qsys Generated Files (Part 1 of 3)

File Name or Directory Name	Description
<code><qsys_design></code>	This is the top-level project directory.
<code><qsys_design>.qsys</code>	Qsys System File (.qsys). The .qsys file contains a list of system components, connections and their parameterizations.
<code><qsys_design>.bsf</code>	A Block Symbol File (.bsf) representation of the top-level Qsys system for use in Quartus II Block Diagram Files (.bdf).

Table 5-3. Qsys Generated Files (Part 2 of 3)

File Name or Directory Name	Description
<qsys_design>.html	This is a report for the system which provides a system overview including the following information: <ul style="list-style-type: none"> ■ All external connections for the system ■ A memory map showing the address of each Avalon-MM slave with respect to each Avalon-MM master to which it is connected ■ All parameter assignments for each component
<qsys_design>.sopcinfo	Qsys information file (.sopcinfo) that describes all of the components and connections in your system. This file is a complete system description, and is used by downstream tools such as the Nios II tool chain. It also describes the parameterization of each component in the system; consequently, you can parse its contents to get requirements when developing software drivers for Qsys components. This file and the system.h file generated for the Nios II tool chain include address map information for each slave relative to each master that accesses the slave. Different masters may have a different address map to access a particular slave component.
/synthesis	This directory includes the Qsys-generated output files that the Quartus II software uses to synthesize your design.
<qsys_design>.v	An HDL file for the top-level Qsys system that instantiates each component in the system.
<qsys_design>.qip	This file lists the Quartus II software needed to compile your design. You must add the .qip file to your Quartus II project.
/submodules	Contains Verilog HDL or VHDL submodule files for synthesis.
/simulation	This directory includes the Qsys-generated output files to simulate your design.
<qsys_design>.v or <qsys_design>.vhd	An HDL file for the top-level Qsys system that instantiates each submodule in the system.
/submodules	Contains Verilog HDL or VHDL submodule files for simulation.
/mentor/	Contains a ModelSim® script msim_setup.tcl to set up and run a simulation.
/synopsys/vcs/	Contains a shell script vcs_setup.sh to set up and run a VCS® simulation.
/synopsys/vcsmx	Contains a shell script vcsmx_setup.sh and synopsys_sim.setup to set up and run a VCS MX simulation.
/cadence	Contains a shell script nccsim_setup.sh and other setup files to set up and run an NCSIM simulation.
/testbench	Contains a Qsys testbench system as described in the “ Simulating a Qsys System ” section.
<qsys_design>_tb.qsys	A Qsys testbench system.
/simulation	This directory includes the Qsys-generated output files to simulate your Qsys testbench system.
<qsys_design>_tb.v <qsys_design>_tb.vhd	The top-level testbench file, which connects BFM's to the top-level interfaces of <qsys_design>.qsys.
/submodules	Contains Verilog HDL or VHDL submodule files for the testbench.
/mentor/	Contains a ModelSim script msim_setup.tcl to set up and run a simulation.
/synopsys/vcs/	Contains a shell script vcs_setup.sh to set up and run a VCS simulation.

Table 5-3. Qsys Generated Files (Part 3 of 3)

File Name or Directory Name	Description
/synopsys/vcsmx	Contains a shell script vcsmx_setup.sh and synopsys_sim.setup to set up and run a VCS MX simulation.
/cadence	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSIM simulation.

Simulating a Qsys System

In most cases, you should select only one of the simulation model options:

- Generate a simulation model for the original system.
- Generate a simulation model for the testbench system.

The Qsys **Generation** tab provides different options for simulating your system. You have the following three flows for simulating a Qsys system:

- You can generate just the Verilog or VHDL simulation model for your system to use in your own simulation environment.
- You can generate a standard or simple testbench system with Avalon BFM components that drive the external interfaces of your system, and generate a Verilog or VHDL simulation model for the testbench system to use in your simulation tool.
- You can first generate the testbench system and then modify the system in Qsys before generating its simulation model.

Table 5-4 summarizes the different options on the **Generation** tab that correspond to the simulation flows described above.

Table 5-4. Summary of Simulation Settings on Qsys Generation Tab


Simulation Setting	Value	Description
Create simulation model	None Verilog VHDL	Creates simulation model files and simulation scripts. Use this option to include the simulation model in your own custom testbench or simulation environment. You can also use this option to generate models for a testbench system that you modify.
Create testbench Qsys system	Standard, BFM for standard Avalon interfaces	Creates testbench Qsys system with Avalon BFMs attached to all exported interfaces. Includes any simulation partner modules specified by IP cores in the system. This testbench is not supported with VHDL simulation models.
	Simple, BFM for clocks and resets	Creates testbench Qsys system with Avalon BFMs driving only clocks and reset interfaces. Includes any simulation partner modules specified by IP cores in the system.
Create testbench simulation model	None Verilog VHDL	Creates simulation model files and simulation scripts for the testbench Qsys system specified in the setting above. Use this option if you do not need to modify the Qsys-generated testbench before running the simulation.

Generating a System Simulation Model

Complete the following steps to generate a simulation model for your Qsys system:

1. On the **Generation** tab, set **Create simulation model** to **Verilog** or **VHDL**.
2. Click **Generate**.


This option creates simulation model files in the specified **Output Directory for Simulation**, along with simulation scripts.

 Qsys can generate a simulation model for the top-level design and Qsys interconnect in either Verilog HDL or VHDL. Your IP components must also provide a simulation model in the language of your choice to ensure a complete single-language simulation model file.

Generating a Standard Qsys Testbench

Follow these steps to generate a standard testbench system that instantiates and connects BFM s that you can use to provide stimulus to the Qsys system:

1. On the **System Contents** tab, export interfaces of your Qsys system that you want to access during simulation.
2. On the **Generation** tab, set **Create testbench Qsys system** to either **Standard, BFM s for standard Avalon interfaces** to create a testbench with BFM s attached to all exported interfaces, or **Simple, BFM s for clocks and resets** to create a testbench with only clocks and reset BFM s.
3. Optionally, to generate a simulation model for the newly-generated testbench Qsys system, set **Create testbench simulation model** to **Verilog** or **VHDL**. Set this option to **None** to create a testbench system but not generate an HDL simulation model. You can choose the **None** option if you want to view or modify the generated testbench system before generating its simulation model.
4. Click **Generate**.

 If a Qsys system contains Avalon BFM s, VHDL simulation models are not supported because the BFM components rely on Verilog function calls. Only the simple testbench with clock and reset is supported for VHDL simulation.

5. Optionally, to view or modify the testbench system, open the `<qsys_system>_tb.qsys` file in the `<qsys_system>/testbench` subdirectory. You can then make changes such as the following:
 - Rename the BFM s to match your simulation environment.
 - Change the BFM settings to ensure adequate test coverage.
 - Replace the default BFM s with custom test components or models.
6. If you made any changes to the testbench system in step 5, generate a simulation model for your updated testbench system with the steps in [“Generating a System Simulation Model”](#).

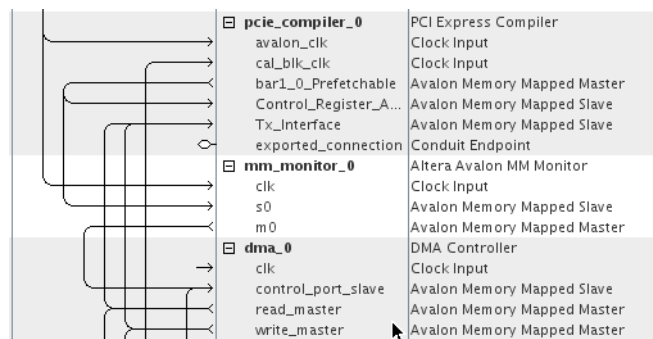
The Qsys-generated standard testbench matches inserted BFM s with the exported interfaces from the Qsys design. To write a test program that provides transaction stimulus to the BFM s, you must take the matching interfaces into account. For example, an exported Avalon-MM slave interface that expects word-aligned addresses is connected to an Avalon master BFM that requires and transacts word-aligned addresses instead of the byte or symbol addresses that are default for Avalon masters.

- For more information about using BFM s and writing transaction-level test code, including tutorials demonstrating sample systems, refer to the *Avalon Verification IP Suite User Guide*.

Adding System Monitors

You can add monitors to Avalon-MM and Avalon-ST connects in your system to verify protocol correctness and test coverage with a simulator that supports SystemVerilog assertions. Figure 5-6 demonstrates the use of monitors. It places an Avalon-MM monitor between the previously connected `pcie_compiler_bar1_0_Prefetchable` Avalon-MM master interface and the `dma_0_control_port_slave` Avalon-MM slave interface.

Figure 5-6. Inserting an Avalon-MM Monitor between Avalon-MM Master and Slave Interfaces



In a similar manner, you can insert an Avalon-ST monitor between Avalon-ST source and sink interfaces.

- For more information about using BFM s and system monitors, including tutorials demonstrating sample systems, refer to the *Avalon Verification IP Suite User Guide*.

Simulation Scripts

Qsys generates simulation scripts to set up the simulation environment for Mentor Graphics ModelSim, Synopsys VCS and VCS MX, and Cadence® Incisive® Enterprise Simulator (NCSIM). You can use the scripts to compile the required device libraries and system design files in the correct order and elaborate or load the top-level design for simulation.

The simulation scripts provide the following variables that allow flexibility in your simulation environment:

- `TOP_LEVEL_NAME`: If the Qsys testbench system is not the top-level instance in your simulation environment because you instantiate the Qsys testbench within your own top-level simulation file, set the `TOP_LEVEL_NAME` variable to the top-level hierarchy name.
- `QSYS_SIMDIR`: If the simulation files generated by Qsys are not in the simulation working directory, use the `QSYS_SIMDIR` variable to specify the directory location of the Qsys simulation files.
- Other variables to control the compilation, elaboration, and simulation process.

Example 5-2 shows a simple top-level simulation HDL file for a testbench system `pattern_generator_tb`, which was generated for a Qsys system called `pattern_generator`. The `top.sv` file defines the `top` module that instantiates the `pattern_generator_tb` simulation test model as well as a custom SystemVerilog test program with BFM transactions, called `test_program`.

Example 5-2. Top-level Simulation HDL File

```
module top();  
  pattern_generator_tb tb();  
  test_program pgm();  
endmodule
```

Mentor Graphics ModelSim Simulation Tcl Script

The simulation script for Mentor Graphics simulators ModelSim, ModelSim-Altera, and QuestaSim is called `msim_setup.tcl`. The script creates alias commands to compile the required device libraries and system design files in the correct order and elaborate or load the top-level design for simulation. To run this script, type `source msim_setup.tcl` in the ModelSim Transcript window. The commands you can use in ModelSim to setup and load the simulation are displayed in the console when you source the `.tcl` file.

Example 5-3 shows a custom top-level simulation script that sets the hierarchy variable `TOP_LEVEL_NAME` to `top` for the design in **Example 5-2**, and sets the variable `QSYS_SIMDIR` to the location of the Qsys-generated **simulation files**. In this example, the top-level simulation files are stored in the same directory as the original Qsys system under test. The `QSYS_SIMDIR` variable provides the relative hierarchy path that Qsys generates for the testbench simulation models.

The script calls the Qsys-generated ModelSim script and uses the alias commands from `msim_setup.tcl` to compile and elaborate the Qsys files required for simulation along with the custom test program and top-level simulation file. You can specify additional ModelSim elaboration command options when you run the `elab` command, for example, `elab +nowarnTFMPC`.

Example 5-3. Top-level ModelSim Simulation Script Including Custom Test Program

```
# Set hierarchy variables used in the Qsys-generated files
set TOP_LEVEL_NAME "top"
set QSYS_SIMDIR "./pattern_generator/testbench"

# Source Qsys-generated script and set up alias commands used below
source $QSYS_SIMDIR/mentor/msim_setup.tcl

# Compile device library files
dev_com

# Compile design files in correct order
com

# Compile the additional test files
vlog -sv ./test_program.sv
vlog -sv ./top.sv

# Elaborate the top-level design
elab
```



For more details about this simulation example, refer to the "Simulating Custom Components" chapter of the *Qsys Tutorial Design Example*.

Using a top-level simulation script and test program takes advantage of the Qsys-generated ModelSim script to ensure you compile all the required simulation files and allows you to modify the top-level simulation environment independently of the Qsys-generated simulation files that are over-written when you change your Qsys system and regenerate.

Synopsys VCS and VCS MX Simulation Shell Script

The simulation scripts for Synopsys VCS and VCS MX are called `vcs_setup.sh` and `vcsmx_setup.sh`. The scripts contain shell commands that compile the required device libraries and system design files in the correct order, and elaborate the top-level design for simulation and run the simulation for 100 time units by default. You can run these scripts from a Linux command shell.



The `vcs_setup.sh` script is only generated for a single-language HDL system.

To set up the simulation for a design such as [Example 5-2](#), edit the variable values in the generated shell script to set the hierarchy variable `TOP_LEVEL_NAME` to `top` and set the variable `QSYS_SIMDIR` to the location of the Qsys-generated simulation files. Edit the compilation commands to add the top-level simulation HDL file and the test program file.

You can specify additional elaboration and simulation options with the variables `USER_DEFINED_ELAB_OPTIONS` and `USER_DEFINED_SIM_OPTIONS`.

Cadence Incisive Enterprise Simulation Shell Script

The simulation script for the Cadence Incisive Enterprise Simulation Shell Script (NCSIM) is called `ncsim_setup.sh`. The script contains shell commands that compile the required device libraries and system design files in the correct order, and elaborate or load the top-level design for simulation and run the simulation for 100 time units by default. You can run this script from a Linux command shell.

To set up the simulation for a design as shown in [Example 5-2](#), edit the variable values in the generated shell script to set the hierarchy variable `TOP_LEVEL_NAME` to `top` and set the variable `QSYS_SIMDIR` to the location of the Qsys-generated **simulation** files. Edit the compilation commands to add the top-level simulation HDL file and the test program file.

You can specify additional elaboration and simulation options with the variables `USER_DEFINED_ELAB_OPTIONS` and `USER_DEFINED_SIM_OPTIONS`.

Simulating Software Running on a Nios II Processor

To simulate the software code in a system driven by a Nios II embedded processor, generate the simulation model for a simple Qsys testbench system by completing the following steps:

1. On the **Generation** tab, set **Create testbench Qsys system** to **Simple, BFM's for clocks and resets**.
2. Set **Create testbench simulation model** to **Verilog** or **VHDL**.
3. Click **Generate**.

Follow these steps to use the software build tools for simulation:

1. Open the **Nios II Software Build Tools for Eclipse**.
2. Set up an application project and board support package (BSP) for the `<qsys_system>.sopcinfo` file.
3. To optimize the BSP for simulation and disable hardware programming, right-click the BSP project and click **Properties**, then click **Nios II BSP Properties**, and turn on **ModelSim only, no hardware support**.
4. To simulate, right-click the application project in Eclipse, point to **Run as** and click **4 Nios II ModelSim**. The **Run As Nios II ModelSim** command sets up the ModelSim simulation environment, compiles and loads the Nios II software simulation.
5. To run the simulation in ModelSim, type `run -all` in the ModelSim transcript window.
6. If prompted, set ModelSim configuration settings and select the correct Qsys Testbench Simulation Package Descriptor (`.spd`) file, `<qsys_system>_tb.spd`. The SPD file is generated with the testbench simulation model for Nios II designs and specifies all the files required for the Nios II software simulation.



For more information about the Nios II SBT for Eclipse, refer to [Getting Started with the Graphical User Interface](#) in the *Nios II Software Developer's Handbook*. For more information about the Nios II SBT command-line options, refer to [Getting Started from the Command Line in the Nios II Software Developer's Handbook](#).

Figure 5-8 shows the Qsys representation of the PCI Express subsystem.

Figure 5-8. Qsys Representation of the PCI Express Subsystem

System Contents	System Inspector	Address Map	Clock Settings	Project Settings	Generation	HDL Examp
Use	Connections	Module	Description		Export As	
<input checked="" type="checkbox"/>		pci_compiler_0	PCI Express Compiler			
		bar1_0_Prefetchable	Avalon Memory Mapped Master		Click to export	
		Control_Register_A...	Avalon Memory Mapped Slave		Click to export	
		Cralrq	Interrupt Sender		Click to export	
		RxmIrq	Interrupt Receiver		Click to export	
		Tx_Interface	Avalon Memory Mapped Slave		Click to export	
		exported_connection	Conduit Endpoint			pci_link
<input checked="" type="checkbox"/>		dma_0	DMA Controller			
		control_port_slave	Avalon Memory Mapped Slave		Click to export	
		irq	Interrupt Sender		Click to export	
		read_master	Avalon Memory Mapped Master		Click to export	
		write_master	Avalon Memory Mapped Master		Click to export	
<input checked="" type="checkbox"/>		mm_bridge_0	Avalon-MM Pipeline Bridge (Qsys)			
		s0	Avalon Memory Mapped Slave		Click to export	
		m0	Avalon Memory Mapped Master			ddr3_s dram_master

Figure 5-9 shows the details of the Ethernet example subsystem from Figure 5-1. In this subsystem, the transmit (TX) DMA receives data from the DDR3 memory and writes it to the Altera Triple-Speed Ethernet IP core using an Avalon-ST source interface. The receive (RX) DMA accepts data from the Triple-Speed Ethernet IP core on its Avalon-ST sink interface and writes it to DDR3 memory.

The read and write masters of both Scatter-Gather DMA controllers and the Triple-Speed Ethernet IP core connect to the DDR3 memory through an Avalon-MM pipeline bridge. This Ethernet example subsystem exports all three control and status interfaces through an Avalon-MM pipeline bridge which connects to a controller outside of the Qsys system.

Figure 5-9. Scatter-Gather DMA-to-Ethernet Example Subsystem

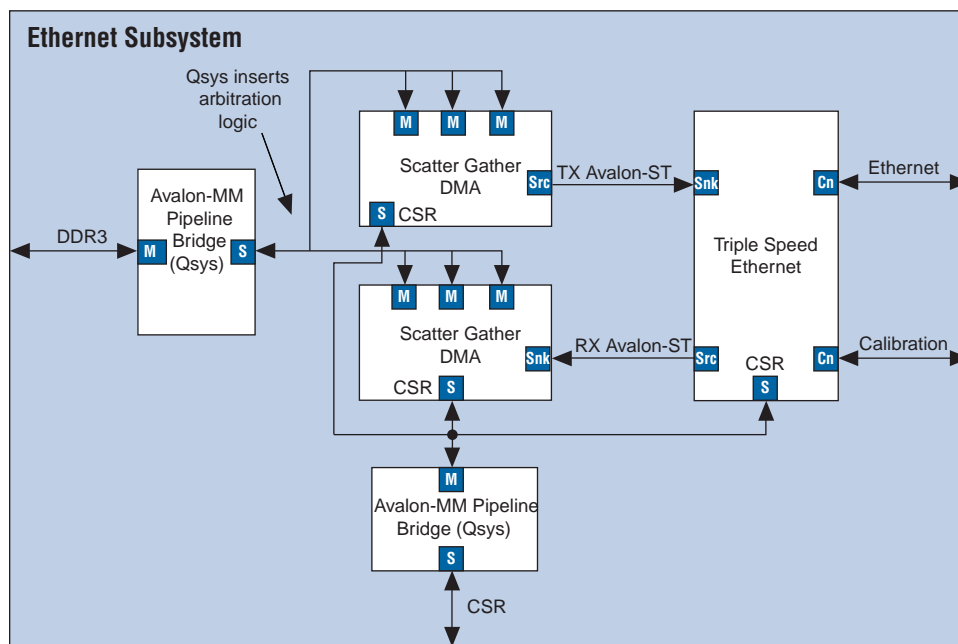
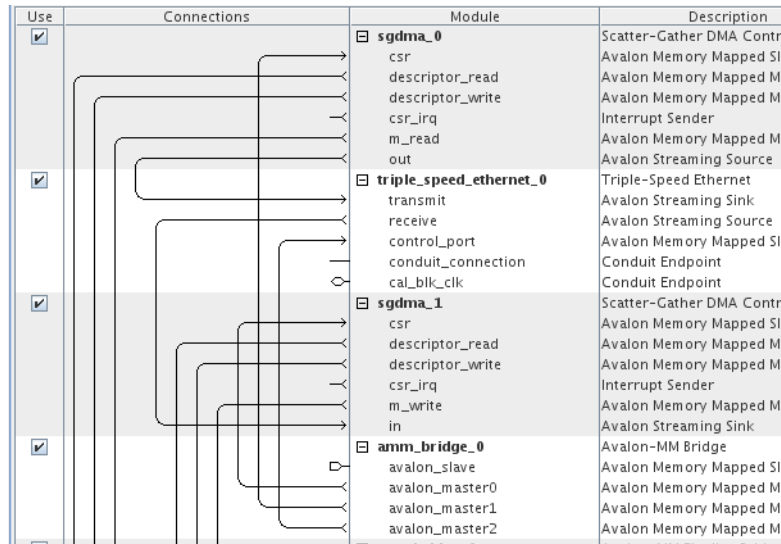


Figure 5-10 shows the Qsys representation of the Ethernet subsystem.

Figure 5-10. Qsys Representation of the Ethernet Subsystem



This example system includes two clock domains. The PCI Express and Ethernet subsystems run at 125 MHz. The DDR3 SDRAM controller runs at 200 MHz. Qsys automatically inserts clock crossing logic to synchronize the DDR3 SDRAM Controller with the with the PCI Express and Ethernet subsystems. Figure 5-11 shows the top level of the example system.

Figure 5-11. PCI Express-to-Ethernet Bridge Example System

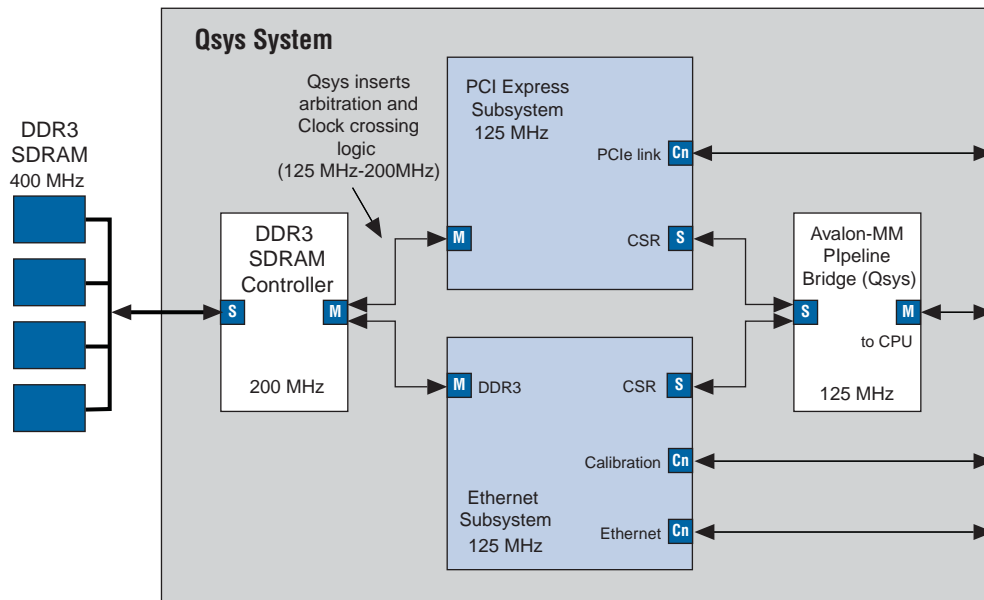


Figure 5-12 shows the Qsys representation of the complete design.


Figure 5-12. Qsys Representation of the Complete PCI Express to Ethernet Bridge

Use	Connections	Module	Description	Export As
<input checked="" type="checkbox"/>		uniphy_dds3_0	DDR3 SDRAM Controller with UniPHY (New)	
		avalon_slave_0	Avalon Memory Mapped Slave	Click to export
		memory_phy	Conduit	ddr3_sdr3m
		Other	Conduit	Click to export
		PLL_sharing	Conduit	Click to export
<input checked="" type="checkbox"/>		ethernet_dma_subsystem	ethernet_dma_subsystem	
		cal_blk_if	Conduit Endpoint	dma_calibration
		csr_if	Avalon Memory Mapped Slave	Click to export
		dma_a_if	Avalon Memory Mapped Master	Click to export
		ethernet_if	Conduit Endpoint	ethernet
<input checked="" type="checkbox"/>		pcie_subsystem	pcie_subsystem	
		pcie_link	Conduit Endpoint	pcie_link
		dds3_sdr3m_master	Avalon Memory Mapped Master	Click to export
		dma_control_slave	Avalon Memory Mapped Slave	Click to export
<input checked="" type="checkbox"/>		mm_bridge_0	Avalon-MM Pipeline Bridge (Qsys)	
		s0	Avalon Memory Mapped Slave	embedded_control
		m0	Avalon Memory Mapped Master	Click to export

Pipeline Bridges

The PCI Express to Ethernet bridge example system uses several pipeline bridges. These bridges must be configured to accommodate the address range of all of connected components, including the components in the originating subsystem and the components in the next higher level of the system hierarchy. As the name suggests, the pipeline bridge inserts a pipeline stage between the connected components. Altera recommends registering signals at the subsystem interface level for the following reasons:

- Registering interface signals decreases the amount of combinational logic that must be completed in one cycle, making it easier to close timing.
- Registering interface signals raises the potential frequency, or f_{MAX} , of your design at the expense of an additional cycle of latency, which might adversely affect system throughput.
- The Quartus II incremental compilation feature can achieve better f_{MAX} results if the subsystem boundary is registered.

 For more information about optimizing a Qsys design for performance using bridges and other techniques, refer to *Optimizing System Performance for Qsys* in volume 1 of the *Quartus II Handbook*.

Hierarchical Components

Any Qsys system that exports an interface is available for use in other Qsys systems. Figure 5-13 shows the component library, including the PCI Express and Ethernet subsystems as components in the component library for the PCI Express to Ethernet bridge example system. To include this Qsys component in other designs, you can add it to the component library or include the directory for this component in component search path for Qsys.


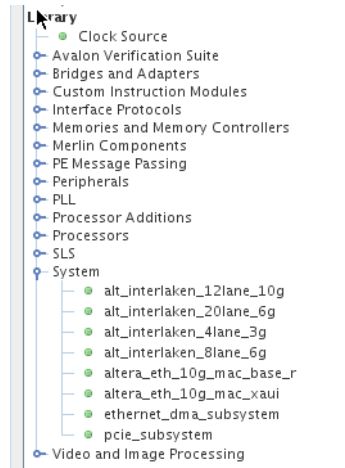
 Because Qsys systems become components in the component library, be careful not to give your system a name that is already in use.

Figure 5-13. Qsys Component Library



For more information about your IP search path and using an IP Index File (.ipx) to provide path information for a Qsys system, refer to the “Component Search Path” section in the *Creating Qsys Components* chapter in volume 1 of the *Quartus II Handbook*. For more information about adding components to your library, refer to the “Installing Additional Components” section in the *Creating Qsys Components* chapter in volume 1 of the *Quartus II Handbook*.

Example Hierarchical System Using Parameters

This section shows an example of how you can use an instance parameter to control the implementation of system components from a higher-level Qsys system. In this example, a Qsys design called **my_system.qsys** has two instances of the same IP component, **My_IP**. **My_IP** is a **_hw.tcl** component with a system identification parameter called **MY_SYSTEM_ID**. When this **my_system.qsys** Qsys design is instantiated within another higher-level Qsys system, the two **My_IP** subcomponents require different values for their **MY_SYSTEM_ID** parameters based on a value determined by that higher-level system. In this example, the value specified by the top-level system is designated **top_id** and in **my_system.qsys**, the component instance **comp0** requires **MY_SYSTEM_ID** set to **top_id + 1**, and instance **comp1** requires **MY_SYSTEM_ID** set to **top_id + 2**.

The following **_hw.tcl** code defines the **MY_SYSTEM_ID** System ID parameter in the IP component **My_IP**:

```
add_parameter MY_SYSTEM_ID int 8
set_parameter_property MY_SYSTEM_ID DISPLAY_NAME \
MY_SYSTEM_ID_PARAM
set_parameter_property MY_SYSTEM_ID UNITS None
```

To satisfy the design requirements for this example, you define an instance parameter in the **my_system.qsys** system that is set by the higher-level system. You define an instance script to specify how the values of the parameters of the **My_IP** components instantiated in **my_system.qsys** are affected by the value set on the instance parameter.

In Qsys, open the **my_system.qsys** Qsys system that instantiates the two instances of the **MY_IP** components. On the **Instance Parameters** tab, create a parameter called **system_id**. For this example, you can set this parameter to be of type Integer and choose **0** as the default value.

- ② For more information on creating a parameter on the **Instance Parameters** tab, refer to *Working with Instance Parameters in Qsys* in Quartus II Help.

Next, you provide a **Tcl Instance Script** that defines how the value of the **system_id** parameter should affect the parameters of **comp0** and **comp1** subcomponents in **my_system.qsys**.

The following example script gets the value of the parameter **system_id** from the top-level system and saves it as **top_id**, then increments the value by 1 and 2. The script then uses the new calculated values to set the **MY_SYSTEM_ID** parameter in the **My_IP** component for the instances **comp0** and **comp1**. The script uses informational messages to print the status of the parameter settings when the **my_system.qsys** system is added to the higher-level system.

Example 5-4. Example Instance Script To Set Parameters On Subcomponents

```
Package Require Qsys 11.1
Set_module_property Composition_callback My_callback
Proc My_callback { } {
    # Get The Value Of System_id Parameter From The Higher-level System
    Set Top_id [Get_parameter_value System_id]

    # Print Info Message
    Send_message Info "System_id Value Specified: $top_id"

    # Use Above Value To Set Parameter Values For The Subcomponents
    Set Child_id_0 [Expr {$top_id + 1} ]
    Set Child_id_1 [Expr {$top_id + 2} ]

    # Set The Parameter Values On The Subcomponent Instances
    Set_instance_parameter_value Comp0 My_system_id $child_id_0
    Set_instance_parameter_value Comp1 My_system_id $child_id_1

    # Print Info Messages
    Send_message Info "System_id Value Used In Comp0: $child_id_0"
    Send_message Info "System_id Value Used In Comp1: $child_id_1"
}
```


You can click **Preview Instance** to see a parameters panel that allows you to modify the parameter value interactively and see the effect of the scripts in the message panel which can be useful for debugging the script. In this example, if you change the parameter value in the **Preview** screen, the component generates messages to report the top-level ID parameter value and the parameter values used for the two instances of the component.


Document Revision History

Table 5-5 shows the revision history for this document.

Table 5-5. Document Revision History

Date	Version	Changes
November 2011	11.1.0	<ul style="list-style-type: none"> ■ Added Synopsys VCS and VCS MX Simulation Shell Script. ■ Added Cadence Incisive Enterprise (NCSIM) Simulation Shell Script. ■ Added Using Instance Parameters and Example Hierarchical System Using Parameters.
May 2011	11.0.0	<ul style="list-style-type: none"> ■ Added simulation support in Verilog HDL and VHDL. ■ Added testbench generation support. ■ Updated simulation and file generation sections.
December 2010	10.1.0	Initial release.

 For previous versions of the Quartus II Handbook, refer to the [Quartus II Handbook Archive](#).

 Take an [online survey](#) to provide feedback about this handbook chapter.