

2014.08.18

QIIS1020



Subscribe



Send Feedback

Qsys is a system integration tool included as part of the Quartus® II software. Qsys captures system-level hardware designs at a high level of abstraction and simplifies the task of defining and integrating customized IP components. These components include verification IP cores, and other design modules. Qsys facilitates design reuse by packaging and integrating your custom IP components with Altera® and third-party IP components. Qsys automatically creates interconnect logic from the high-level connectivity that you specify, thereby eliminating the error-prone and time-consuming task of writing HDL to specify system-level connections.

Qsys is more powerful if you design your custom IP components using standard interfaces. By using standard interfaces, your custom IP components inter-operate with the Altera IP components in the IP Catalog. In addition, you can take advantage of bus functional models (BFMs), monitors, and other verification IP to verify your system.

Qsys supports Avalon®, AMBA® AXI3™ (version 1.0), AMBA AXI4™ (version 2.0), AMBA AXI4-Lite™ (version 2.0), AMBA AXI4-Stream (version 1.0), and AMBA APB™ 3 (version 1.0) interface specifications.

Qsys provides the following advantages:

- Simplifies the process of customizing and integrating IP components into systems
- Generates an IP core variation for use in your Quartus II software projects
- Supports up to 64-bit addressing
- Supports modular system design
- Supports visualization of systems
- Supports optimization of interconnect and pipelining within the system
- Supports auto-adaptation of different data widths and burst characteristics
- Supports inter-operation between standard protocols, such as Avalon and AXI
- Fully integrated with the Quartus II software

Note: For information on how to define and generate single IP cores for use in your Quartus II software projects, refer to *Introduction to Altera IP Cores*.

Related Information

- [Avalon Interface Specifications](#)
- [AMBA Protocol Specifications](#)
- [Creating Qsys Components](#)

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO
9001:2008
Registered



- [Qsys Interconnect](#)
- [Introduction to Altera IP Cores](#)
- [Managing Quartus II Projects](#)

IP Component Interface Support

IP components can have any number of interfaces in any combination. Each interface represents a set of signals that you can connect within a Qsys system, or export outside of a Qsys system.

Qsys IP components can include the following interface types:

Table 5-1: IP Component Interface Types

Interface Type	Description
Memory-Mapped	Implements a partial crossbar interconnect structure (Avalon-MM, AXI, and APB) that provides concurrent paths between master and slaves. Interconnect consists of synchronous logic and routing resources inside the FPGA, and implementation is based on a network-on-chip architecture.
Streaming	Connects Avalon Streaming (Avalon-ST) sources and sinks that stream unidirectional data, as well as high-bandwidth, low-latency IP components. Streaming creates datapaths for unidirectional traffic, including multichannel streams, packets, and DSP data. The Avalon-ST interconnect is flexible and can implement on-chip interfaces for industry standard telecommunications and data communications cores, such as Ethernet, Interlaken, and video. You can define bus widths, packets, and error conditions.
Interrupts	Connects interrupt senders to interrupt receivers. Qsys supports individual, single-bit interrupt requests (IRQs). In the event that multiple senders assert their IRQs simultaneously, the receiver logic (typically under software control) determines which IRQ has highest priority, then responds appropriately
Clocks	Connects clock output interfaces with clock input interfaces. Clock outputs can fan-out without the use of a bridge. A bridge is required only when a clock from an external (exported) source connects internally to more than one source.
Resets	Connects reset sources with reset input interfaces. If your system requires a particular positive-edge or negative-edge synchronized reset, Qsys inserts a reset controller to create the appropriate reset signal. If you design a system with multiple reset inputs, the reset controller ORs all reset inputs and generates a single reset output.

Interface Type	Description
Conduits	Connects point-to-point conduit interfaces, or represent signals that are exported from the Qsys system. Qsys uses conduits for component I/O signals that are not part of any supported standard interface. You can connect two conduits directly within a Qsys system as a point-to-point connection, or conduit interfaces can be exported and brought to the top-level of the system as top-level system I/O. You can use conduits to connect to external devices, for example external DDR SDRAM memory, and to FPGA logic defined outside of the Qsys system.

Creating a Qsys System

In the Quartus II software, you can create a new Qsys design by clicking **Tools > Qsys**.

To open a previously created Qsys design, click **File > Open** in the Quartus II software, or in Qsys.

Related Information

[Creating Qsys Components](#)

[Component Interface Tcl Reference](#)

Adding and Connecting IP Components

The **System Contents** tab displays the IP components that you add to your system, and allows you to connect interfaces.

Related Information

[System Contents Tab](#)

Adding IP Components to your Qsys System

In Qsys, the IP Catalog displays IP components available for your target device. Double-click any component to launch the parameter editor, which allows you to create a custom IP variation of the selected component to satisfy the requirements of your Qsys design.

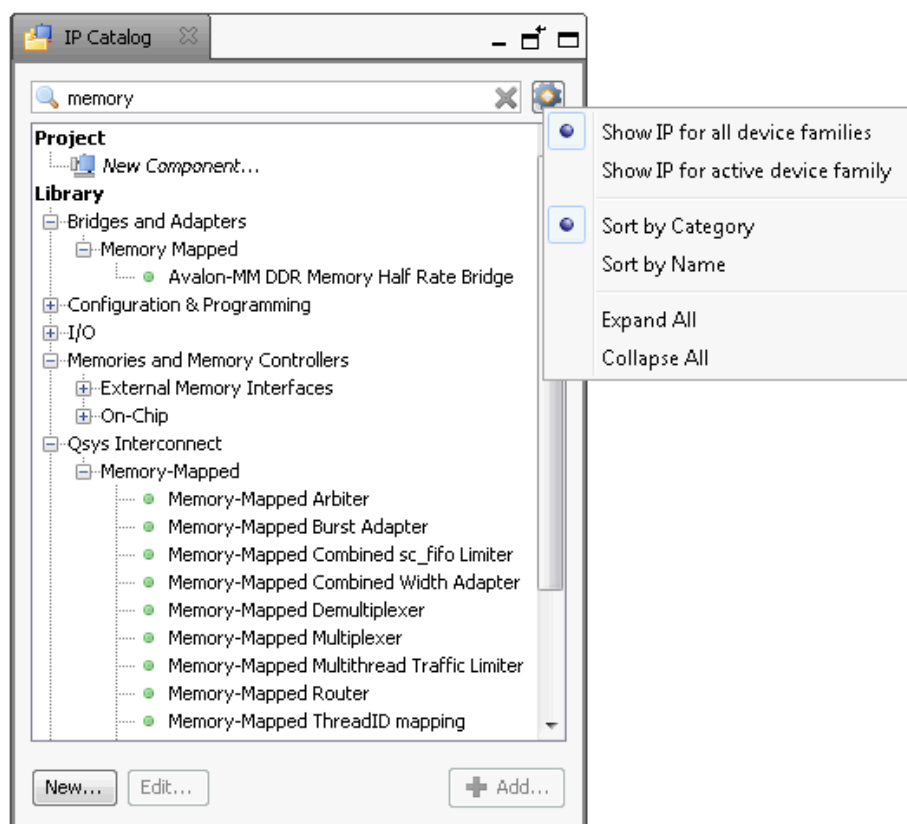
In the parameter editor, you can specify optional ports and architecture features. Some components have preset settings, which allow you to instantly apply preset parameter values appropriate for a specific application. When you complete your customization and click **Finish**, the component displays in the **System Contents** tab.

A Qsys system can contain a single or multiple IP component variations. A **.qsys** file represents your Qsys system in your Quartus II software project.

Right-click any IP component name in IP Catalog to display details about supported devices, installation location, version, and links to documentation. The IP Catalog maintains multiple versions of a component, and you can choose which version you want to include in your design.

To locate a specific type of component, you can type some or all of the component's name in the IP Catalog search box. For example, you can type **memory** to locate memory-mapped IP components, or **axi** to locate AXI IP components. You can also filter the IP Catalog display with options on the right-click menu.

Figure 5-1: IP Catalog

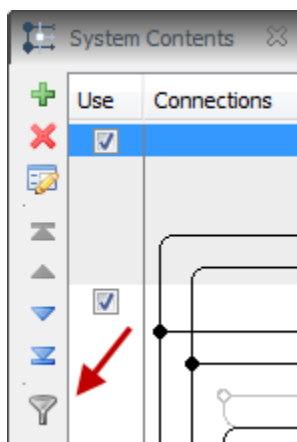


Filtering the Display of the System Contents tab

You can use the **Filters** dialog box to filter the display of your system by interface type, instance name, or by using custom tags.

For example, in the **System Contents** tab, you can show only instances that include memory-mapped interfaces, instances that are connected to a particular Nios II processor, or temporarily hide clock and reset interfaces to simplify the display.

Figure 5-2: Filter Icon in the System Contents Tab



Related Information

[Filters Dialog Box](#)

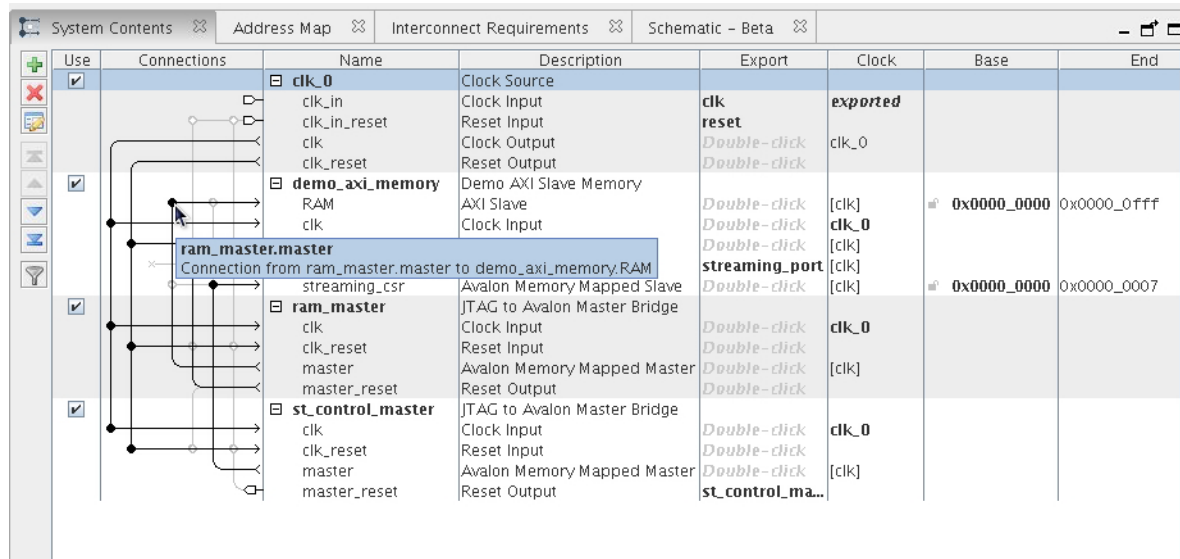
Connecting IP Components

When you add connections to a Qsys system, you can connect the interfaces of the modules in the **System Contents** tab. Qsys interconnect connects the individual signals in each interface when the HDL for the system generates. You connect interfaces of compatible types and opposite directions. For example, you can connect a memory-mapped master interface to a slave interface, and an interrupt sender interface to an interrupt receiver interface.

In the **System Contents** tab, possible connections between interfaces appear as gray lines and open circles. To make a connection, click the open circle at the intersection of the two interface names. When you make a connection, Qsys draws the connection line in black, and fills the connection circle. Clicking a filled-in circle removes a connection.

Figure 5-3: Connections Column in the System Contents Tab

When you are done adding connections to your system, you can deselect **Allow Connection Editing** in the right-click menu. This option sets the **Connections** column to read-only and hides the possible connections.



Use	Connections	Name	Description	Export	Clock	Base	End
<input checked="" type="checkbox"/>		clk_0	Clock Source				
<input checked="" type="checkbox"/>		clk_in	Clock Input	clk	exported		
<input checked="" type="checkbox"/>		clk_in_reset	Reset Input	reset			
<input checked="" type="checkbox"/>		clk	Clock Output	Double-click	clk_0		
<input checked="" type="checkbox"/>		clk_reset	Reset Output	Double-click			
<input checked="" type="checkbox"/>		demo_axi_memory	Demo AXI Slave Memory				
<input checked="" type="checkbox"/>		RAM	AXI Slave	Double-click	[clk]	# 0x0000_0000	0x0000_0fff
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click	clk_0		
<input checked="" type="checkbox"/>		ram_master.master	Avalon Memory Mapped Slave	Double-click	[clk]		
<input checked="" type="checkbox"/>		streaming_csr	JTAG to Avalon Master Bridge	Double-click	[clk]		
<input checked="" type="checkbox"/>		ram_master	Avalon Memory Mapped Master	Double-click	clk_0		
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click	[clk]		
<input checked="" type="checkbox"/>		clk_reset	Reset Input	Double-click			
<input checked="" type="checkbox"/>		master	Avalon Memory Mapped Master	Double-click	[clk]		
<input checked="" type="checkbox"/>		master_reset	Reset Output	Double-click			
<input checked="" type="checkbox"/>		st_control_master	JTAG to Avalon Master Bridge	Double-click	clk_0		
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click	[clk]		
<input checked="" type="checkbox"/>		clk_reset	Reset Input	Double-click			
<input checked="" type="checkbox"/>		master	Avalon Memory Mapped Master	Double-click	[clk]		
<input checked="" type="checkbox"/>		master_reset	Reset Output	Double-click			

The **Connections** tab shows a list of current and possible connections for the selected instances or interfaces in the **Hierarchy** or **System Contents** tabs. You can add and remove connections by clicking the checkbox next to each connection in the list. There are also reporting columns, which may provide some information about each connection. The reporting columns: **Clock Crossing**, **Data Width**, and **Burst**, provide information after system generation when the Qsys interconnect adds additional adapters, which may result in slower f_{Max} , or larger area.

Related Information

Connecting Components

Specifying the Target Device

The **Device Family** tab allows you to select the device family and device for your Qsys system. IP components availability, parameters, and output options reflect the target Qsys-selected device.

Qsys saves device settings in the **.qsys** file.

In Qsys, when you generate your system, the generated output is for the Qsys-selected device.

The Quartus II software always uses the device specified in the Quartus II project settings, even if you have already generated your Qsys system with the Qsys-selected device.

Note: Qsys generates a warning message if the Qsys selected device family and device do not match the Quartus II project settings. Also, when you open Qsys from within the Quartus II software, the Quartus II project device settings apply.

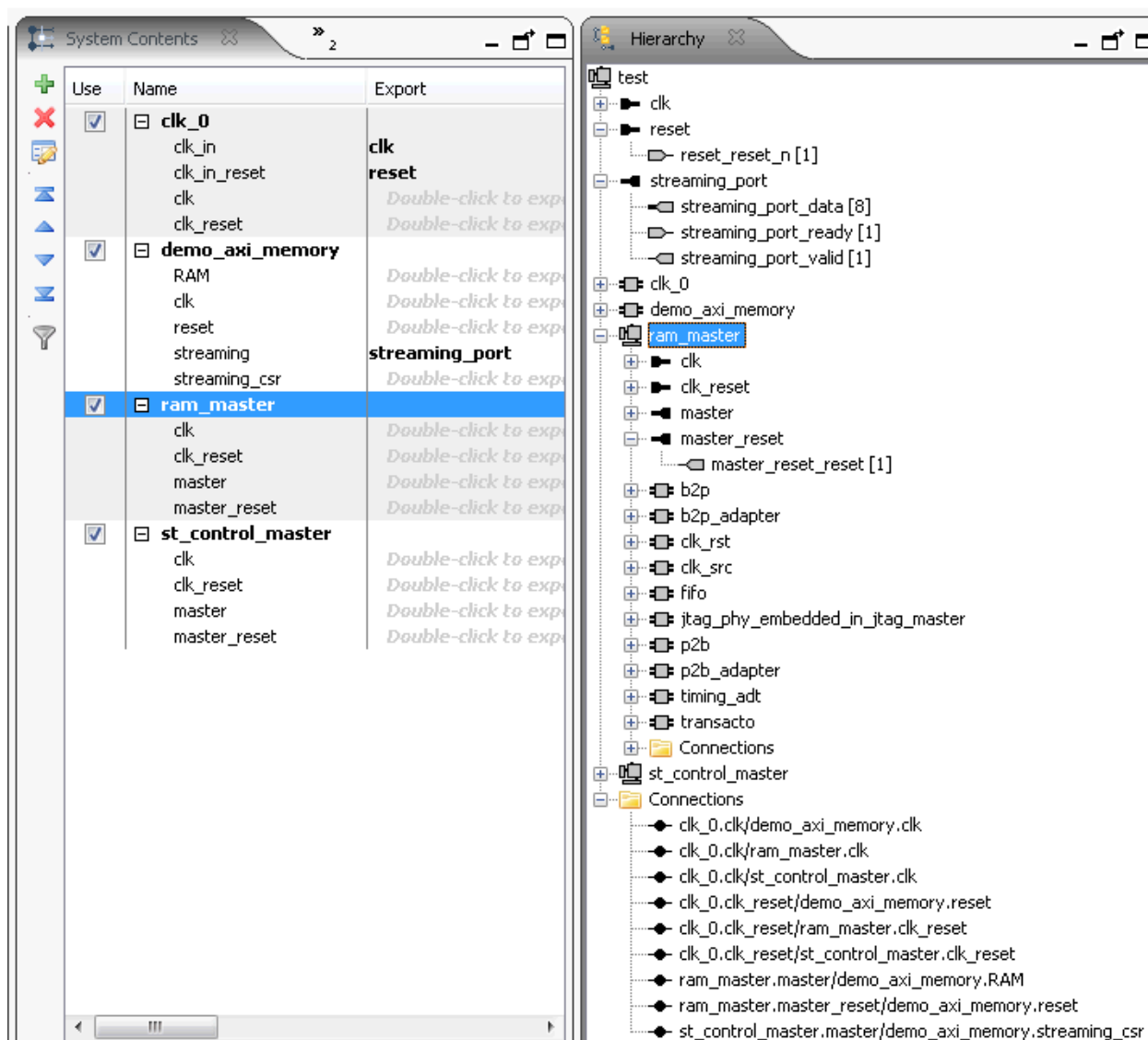
Navigating Your Qsys System

The **Hierarchy** tab is a full system hierarchical navigator, which expands the system contents to show modules, interfaces, signals, subsystems, and connections.

Figure 5-4: Expanding Elements in the Hierarchy Tab

The **Hierarchy** tab displays a unique icon for each element in the system, including interfaces, directional pins, IP blocks, and system icons that show exported interfaces and the instances of IP components that make up the system.

To facilitate design development and debugging, you can use the context sensitivity feature between tabs. For example, when you select an element in one tab, Qsys selects the same element in other open tabs, which allows you to interact with your system in more detail. Below, the `ram_master` selection is highlighted in both the **System Contents** and **Hierarchy** tabs.



You can use the **Hierarchy** tab to browse, connect, parameterize IP, and drive changes in other tabs. Expanding each interface in the **Hierarchy** tab allows you to view sub-components, associated elements, and signals for each interface. You can focus on a particular area of your system by coordinating selections in the **Hierarchy** tab with other open tabs in your workspace.

Reviewing your system using the **Hierarchy** tab in conjunction with relevant tabs is useful during the debugging phase because you can contain and focus your debugging efforts to a single element in your system.

The **Hierarchy** tab provides the following information and functionality:

- Connections between signals.
- Names of signals in exported interfaces.
- Right-click menu to connect, edit, add, remove, or duplicate elements in the hierarchy.
- Internal connections of Qsys subsystems that are included as IP components. In contrast, the **System Contents** tab displays only the exported interfaces of Qsys subsystems.

Setting the Generation ID

You can set the **Generation Id** parameter by selecting the top-level system in the **Hierarchy** tab, and then locating the parameter in the open **Parameters** tab.

The **Generation Id** parameter is a unique integer value that is set to a timestamp just before Qsys system generation. System tools, such as NIOS II or HPS (Hard Processor System) use the **Generation ID** to ensure software-build compatibility with the Qsys system.

Specifying IP Component Parameters

The **Parameters** tab allows you to specify parameters that define the IP component's functionality in order to satisfy your system's design requirements.

When you add a component to your system, or when you double-click a component in the **System Contents** tab, the parameter editor opens. Many IP components in the IP Catalog have parameters that you can configure when you add the component to your system. If you create your own IP components, use the Hardware Component Description File (**_hw.tcl**) to specify the configurable parameters.

With the **Parameters** tab open, when you select an element in the **Hierarchy** tab, Qsys shows the same element in the **Parameters** tab. You can then make changes to the parameters that appear in the parameter editor, including changing the name for top-level instance that appears in the **System Contents** tab. Changes that you make in the **Parameters** tab are immediately reflected in other open tabs in your workspace.

In the parameter editor, the **Documentation** button provides information for a component's parameter, including the version.

At the top of the parameter editor, Qsys shows the hierarchical path for the component and its elements. This feature is useful when you are navigating deep within your system using the **Hierarchy** tab, with the **Parameters** tab also open in your workspace. When you select an interface in your system in the **Hierarchy** tab, the **Parameters** tab also allows you to review the timing for that interface. Qsys displays the read and write waveforms at the bottom of the **Parameters** tab.

Working With Presets

The **Presets** tab allows you to apply a set of parameter values to your IP component. Some preset values support a specific application. The **Presets** tab opens the preset editor and allows you to create, modify, and save custom component parameter values as a preset file. You can apply the preset parameter values to the currently selected component. Not all IP components have preset files.

When you add a new component to your system, if there are preset values available for the component, the preset editor appears in the parameter editor window and lists preset files that you can apply the component.

The name of each preset file describes a particular protocol and contains the required parameter values for that protocol.

You can search for text to filter the **Presets** list. For example, if you add the **DDR3 SDRAM Controller with UniPHY** component to your system, and type `1g micron 256` in the search box, the **Presets** list shows only those parameter values that apply to the `1g micron 256` filter request. Presets whose parameter values match the current parameter settings appear in bold.

In the preset editor, if the available preset files do not meet the requirements of your design, you can create a new preset file. In the presets editor, clicking **New** opens the **New Preset** dialog box, where you specify the new custom preset name, component version, description of the preset, and which parameters to include in the preset. You can also specify where you want to save the preset file. If the file location that you specify is not already in the IP search path, Qsys adds the location of the new preset file to the IP search path.

In the preset editor, clicking **Update** allows you to update parameter values for a custom preset. The **Update Preset** dialog box displays the default value, which you can edit, and the current value, which is static. You can remove a custom preset from the **Presets** list by clicking **Delete**.

When you access the presets editor by clicking **View > Presets**, you can apply the available presets to the currently selected component.

Related Information

[Presets Editor](#)

Creating Connections Between Masters and Slaves

The **Address Map** tab provides the address range that each memory-mapped master uses to connect to each slave in your system.

Qsys shows the slaves on the left, and masters across the top, with the address span of the connection in each cell. If there is no connection between a master and a slave, the table cell is empty.

You can design a system where two masters access a slave at different addresses. If you use this feature, Qsys labels the **Base** and **End** address columns in the **System Contents** tab as "mixed" rather than providing the address range.

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

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

Note: The base address of a slave component must be a multiple of the address span of the component. This restriction is a requirement of the Qsys interconnect and results in efficient address decoding logic, and allows Qsys to achieve the best possible f_{MAX} .

Specifying Clock Settings

Use the **Clocks** tab to define the **Name**, **Source**, and frequency (**MHz**) of each clock in your system.

Reporting Connections that Cross Clock Domains

The **Clock Domains** tab reports all connections that cross clock domains in a Qsys system.

For each clock domain in the system, the **Clock Domains** tab lists interfaces and connections.

Displaying the Block Symbol

When the Block Symbol tab is open, Qsys displays a graphical representation of the element selected in the **Hierarchy** or **System Contents** tabs. You can view selected component's interfaces or signals in the **Block Symbol** tab. The **Show signals** options allows you to turn on or off signal graphics.

The **Block Symbol** tab appears by default in the parameter when you add a new component to your system, and reflects changes that you make in other tabs.

Specifying Interconnect Requirements

The **Interconnect Requirements** tab allows you to apply system-wide `$system`, and interface interconnect requirements for IP components in your system.

Specifying System-Wide Interconnect Requirements

You can apply the following system-wide interconnect requirements in the **Interconnect Requirements** tab:

Table 5-2: Specifying System-Wide Interconnect Requirements

Option	Description
Limit interconnect pipeline stages to	Specifies the maximum number of pipeline stages that Qsys may insert in each command and response path 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 data path. Choosing 3 or 4 pipeline stages may significantly increase the logic utilization of the system. This setting is specific for each Qsys system or subsystem, meaning that each subsystem can have a different setting. Additional latency is added once on the command path, and once on the response path. You can manually adjust this setting in the Memory-Mapped Interconnect tab. Access this tab by clicking Show System With Qsys Interconnect command on the System menu.

Option	Description
Clock crossing adapter type	<p>Specifies the default implementation for automatically inserted clock crossing adapters:</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. However, the FIFO-based adapter can sustain higher throughput because it supports multiple transactions 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.

Specifying \$system Interconnect Requirements

You can apply the following interconnect requirements when you select \$system as the **Identifier** in the **Interconnect Requirements** tab, in the **All Requirements** table. \$system is the current system that is open in Qsys.

Table 5-3: Specifying \$system Interconnect Requirements

Option	Description
Limit interconnect pipeline stages to	Refer to <i>Specifying System-Wide Interconnect Requirements</i> .
Clock crossing adapter type	Refer to <i>Specifying System-Wide Interconnect Requirements</i> .
Automate default slave insertion	Specifies whether you want Qsys to automatically insert a default slave for undefined memory region accesses during system generation.
Enable instrumentation	When you set this option to TRUE , Qsys enables debug instrumentation in the Qsys interconnect, which then monitors interconnect performance in the system console.

Related Information

[Specifying System-Wide Interconnect Requirements](#) on page 5-10

Specifying Interface Interconnect Requirements

You can apply the following interconnect requirements when you select a component interface as the **Identifier** in the **Interconnect Requirements** tab, in the **All Requirements** table:

Table 5-4: Specifying Interface Interconnect Requirements

Option	Value	Description
Security	<ul style="list-style-type: none"> Non-secure Secure Secure ranges TrustZone-aware 	<p>After you establish connections between the masters and slaves, allows you to set the security options, as needed, for each master and slave in your system.</p> <p>Note: You can also set these values in the Security column in the System Contents tab.</p>
Secure address ranges	Accepts valid address range.	Allows you to type in any valid address range.
Multiple threadID support	TRUE or FALSE	Enables an AXI master to send a command with different IDs.
ThreadID buffer depth	integer, integer, ...	<p>Qsys backpressures a master if it sends a second command to a different slave after its first command does not receive a response. When you use the ThreadID buffer depth parameter, Qsys allows sending of the second command, which is stored in the interconnect FIFO. Qsys inserts FIFOs on a per thread basis, and uses the ThreadID buffer depth parameter to configure the depth of the FIFO.</p> <p>Only use the ThreadID buffer depth parameter when the Multiple threadID support parameter is TRUE.</p> <p>The number of integer values must be same as the number of ThreadID mapping ranges. If you do not specify the buffer depth, Qsys uses a default value of 4 for all thread mapping ranges.</p>

Option	Value	Description
ThreadID mapping ranges	<p>String in the format: BxxxxBBB, BBBBxxxx, xxxxBBBB, and so on.</p> <p>Where B is either 0 or 1, and x is the character.</p> <p>The length of one value (number for B and x) depends on the master ID width. For example, if there are 8-bits on the ID signal, then there are 8 B and x altogether.</p>	<p>In HPS, Qsys already uses some bits of the ID signal for some peripherals. For example, the MSB and/or LSB bits of the ID signal for some peripherals may be constant and set to either 0 or 1. Therefore possible threadID mapping ranges are, 0xx1, 1xx0, and so on.</p> <p>For example, for an ID signal width of 4, there are 16 individual thread values possible. You can group some of these values. For example, you can have 4 threads of 0-3, 4-7, 8-11, and 12-15. These groups are translated as 00xx, 01xx, 10xx, and 11xx, respectively.</p> <p>You must only use the ThreadID mapping ranges parameter when the Multiple threadID support parameter is TRUE.</p> <p>For more information, refer to the HPS Peripheral Master Input IDs table in the <i>Cyclone V Device Handbook</i>.</p>
Enable master reorder buffer	TRUE or FALSE	<p>If the Enable master reorder buffer setting is FALSE, a master is backpressured if it sends a second command to a different slave when its first command does not receive a response. When TRUE, the second command is sent successfully.</p> <p>If the second slave responds before the first slave, the response is stored in the reorder buffer until the first response returns. After receiving both the responses, the buffer reorders the responses and the master receives them in order.</p>

For more information about HPS, refer to the *Cyclone V Device Handbook* in volume 3 of the *Hard Processor System Technical Reference Manual*.

Related Information

- [Managing System Security](#) on page 5-30
- [Cyclone V Device Handbook](#)

Defining 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 sub-component in another Qsys system. The higher-level Qsys system can assign values to the instance parameters.

The **Instance Script** on the **Instance Parameters** tab defines how the specified values for the instance parameters should affect the sub-components in your Qsys system. The instance script allows you to create

queries about the instance parameters that you define and set the values of the parameters for the sub-components in your system.

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

To use instance parameters, the IP components or subsystems in your Qsys system must have parameters that can be set when they are instantiated in a higher-level 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.

Related Information

[Working with Instance Parameters in Qsys](#)

Creating an Instance Script

The first command in an instance script must specify the Tcl command version for the script. This command ensures the Tcl commands behave identically in future versions of the tool. Use the following Tcl command to specify the version of the Tcl commands, where *<version>* is the Quartus II software version number, such as 14.0:

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

To use Tcl commands that work with instance parameters in the instance script, you must specify the commands within a Tcl 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}
```

Example 5-1: Instance Script Example

In this example, an instance script uses the `pio_width` parameter to set the `width` parameter of a parallel I/O (PIO) component. The script combines the `get_parameter_value` and `set_instance_parameter_value` commands using brackets.

```
# Request a specific version of the scripting API  
package require -exact qsys 13.1  
  
# 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]  
}
```

Related Information

[Component Interface Tcl Reference](#)

Supported Tcl Commands for Instance Scripts

You can use standard Tcl commands to manipulate parameters in the script, such as the `set` command to create variables, or the `expr` command for mathematical manipulation of the parameter values. Instance scripts also use Tcl commands to query the parameters of a Qsys system, or to set the values of the parameters of the sub-IP-components instantiated in the system.

`get_instance_parameter_value`

Description

Returns the value of a parameter in a child instance.

Usage

```
get_instance_parameter_value <instance> <parameter>
```

Returns

The value of the parameter.

Arguments

instance

The name of the child instance.

parameter

The name of the parameter in the instance.

Example

```
get_instance_parameter_value pixel_converter input_DPI
```


get_instance_parameters

Description

Returns the names of all parameters on a child instance that can be manipulated by the parent. It omits parameters that are derived and those that have the `SYSTEM_INFO` parameter property set.

Usage

```
get_instance_parameters <instance>
```

Returns

A list of parameters in the instance.

Arguments

instance

The name of the child instance.

Example

```
get_instance_parameters instance
```

get_parameter_value

Description

Returns the current value of a parameter defined previously with the `add_parameter` command.

Usage

```
get_parameter_value <parameter>
```

Returns

The value of the parameter.

Arguments

parameter

The name of the parameter whose value is being retrieved.

Example

```
get_parameter_value fifo_width
```

get_parameters

Description

Returns the names of all the parameters in the component.

Usage

```
get_parameters
```

Returns

A list of parameter names.

Arguments

No arguments.

Example

```
get_parameters
```

send_message

Description

Sends a message to the user of the component. The message text is normally interpreted as HTML. The `` element can be used to provide emphasis. If you do not want the message text to be interpreted as HTML, then pass a list like `{ Info Text }` as the message level

Usage

```
send_message <level> <message>
```

Returns

No return value.

Arguments

level

The following message levels are supported:

- `ERROR`--Provides an error message.
- `WARNING`--Provides a warning message.
- `INFO`--Provides an informational message.
- `PROGRESS`--Provides a progress message.
- `DEBUG`--Provides a debug message when debug mode is enabled.

message

The text of the message.

Example

```
send_message ERROR "The system is down!"  
send_message { Info Text } "The system is up!"
```

set_instance_parameter_value

Description

Sets the value of a parameter for a child instance. Derived parameters and `SYSTEM_INFO` parameters for the child instance may not be set using this command.

Usage

```
set_instance_parameter_value <instance> <parameter> <value>
```

Returns

No return value.

Arguments

instance

The name of the child instance.

parameter

The name of the parameter.

value

The new parameter value.

Example

```
set_instance_parameter_value uart_0 baudRate 9600
```

set_module_property

Description

Used to specify the Tcl procedure invoked to evaluate changes in Qsys system instance parameters.

Usage

```
set_module_property <property> <value>
```

Returns

No return value.

Arguments

property

The name of the property. Refer to [Module Properties](#).

value

The new value of the property.

Example

```
set_module_property COMPOSITION_CALLBACK "my_composition_callback"
```

Displaying Your Qsys System

Qsys allows you to change the display of your system to match design development. Each tab on View menu, provides a unique display of your system and allows you to review your design with different perspectives. Some tabs allow you to focus on a particular part of the system, or two different tabs open in your workspace tabs can show the same IP components, with one tab showing more detail, or other relevant information.

Qsys GUI supports global selections and edits. When you make a selection or apply an edit in the **Hierarchy** tab, Qsys updates all other tabs to reflect your action. For example, when you select `cpu_0` in the **Hierarchy** tab, Qsys updates the **Parameters** tab to show the parameters for `cpu_0`.

By default, when you open Qsys, the IP Catalog and **Hierarchy** tab display to the left of the main frame. The **System Contents**, **Address Map**, **Interconnect Requirements**, and **Device Family** tabs display in the main frame.

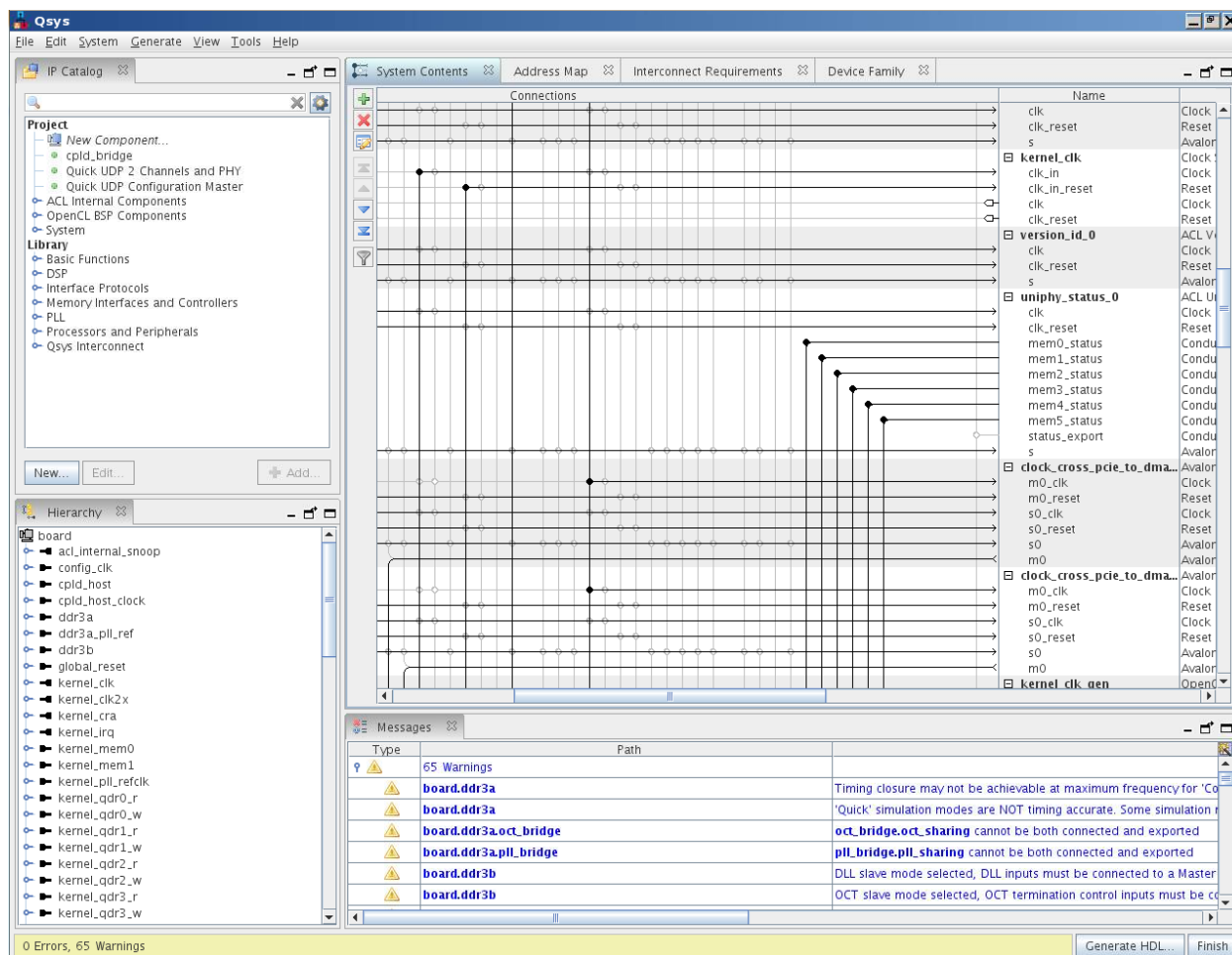
The **Messages** tab displays in the lower portion of Qsys. Double-clicking a message in the **Messages** tab changes focus to the associated element in the relevant tab to facilitate debugging. When the **Messages** tab is closed or not in view in your workspace, error and warning message counts continue to display in the status bar of the Qsys window.

You can dock tabs in the main frame as a group, or individually by clicking the tab control in the upper-right corner of the main frame. You can arrange your system design workspace by dragging and dropping, and then grouping tabs in an order appropriate to your design development, or close or dock tabs that you are not using. Tool tips on the upper-right corner of the tab describe possible workspace arrangements, for example, restoring or disconnecting a tab to or from your workspace. When you save your system, Qsys also saves the current workspace configuration, and when you re-open the saved system, Qsys restores the last saved workspace.

The **Reset to System Layout** command on the View menu restores the workspace to its default configuration for system design. The **Reset to IP Layout** command restores the workspace to its default configuration for defining and generating single IP cores.

Note: Qsys contains some tabs which are not documented and appear on the View menu as "Beta". The purpose in presenting these tabs is to allow designers to explore their usefulness in Qsys system development.

Figure 5-5: Displaying Your Qsys System



Displaying IP Component Parameter Information

The **Details** tab provides descriptions for the selected component's parameters.

As you click through the parameter in the parameter editor, Qsys displays the description of the parameter in the **Details** tab. To return to the complete description for the component, click the header in the **Parameters** tab.

Managing IP Settings in the Quartus II Software

To specify the following IP Settings in the Quartus II software, click **Tools > Option > IP Settings**:

Table 5-5: IP Settings

Setting	Description
Maximum Qsys memory usage	Allows you to increase memory usage for Qsys if you experience slow processing for large systems, or if Qsys reports an Out of Memory error.
IP generation HDL preference	The Quartus II software uses this setting when the .qsys file appears in the Files list for the current project in the Settings dialog box and you run Analysis & Synthesis. Qsys uses this setting when you generate HDL files.
Automatically add Quartus II IP files to all projects	The Quartus II software uses this setting when you create an IP core file variation with options in the Quartus II IP Catalog and parameter editor. When turned on, the Quartus II software adds the IP variation files to the project currently open.
IP Search Locations	<p>The Quartus II software uses the settings that you specify for global and project search paths under IP Search Locations, in addition to the IP Search Path in Qsys (Tools > Options), to populate the Quartus II software IP Catalog.</p> <p>Qsys uses the settings that you specify for global search paths under IP Search Locations to populate the Qsys IP Catalog. Qsys uses the project search path settings to populate the Qsys IP Catalog when you open Qsys from within the Quartus II software (Tools > Qsys), but not when you open Qsys from the command-line.</p>

Note: You can also access **IP Settings** by clicking **Assignments > Settings > IP Settings**. This access is available only when you have a Quartus II project open. This allows you access to **IP Settings** when you want to create IP cores independent of a Quartus II project. Settings that you apply or create in either location are shared.

Opening Qsys with Additional Memory

If your Qsys system requires more than the 512 megabytes of default memory, you can increase the amount of memory either in the Quartus II software **Options** dialog box, or at the command-line.

- When you open Qsys from within the Quartus II software, you can increase memory for your Qsys system, by clicking **Tools > Options > IP Settings**, and then selecting the appropriate amount of memory with the **Maximum Qsys memory usage** option.
- When you open Qsys from the command-line, you can add an option to increase the memory. For example, the following `qsys-edit` command allows you to open Qsys with 1 gigabytes of memory.

```
qsys-edit --jvm-max-heap-size=1g
```

Upgrading Outdated IP Components

When you open a Qsys system that contains outdated IP components, Qsys automatically attempts to upgrade the IP components if it cannot locate the requested version. IP components that Qsys successfully updates appear in the **Upgrade IP Cores** dialog box with a green checkmark. Most Qsys IP components support automatic upgrade.

You can include a path to older IP components in the IP Search Path, which Qsys uses even if updated versions are available. However, older versions of IP components may not work in newer version of Qsys.

Note: If your Qsys system includes an IP component(s) outside of the project directory, the directory of the **.qsys** file, or other any other directory location, you must add these dependency paths to the Qsys IP Search Path (**Tools > Options**).

1. With your Qsys system open, click **System > Upgrade IP Cores**.
Only IP Components that are associated with the open Qsys system, and that do not support automatic upgrade appear in **Upgrade IP Cores** dialog box.
2. In the **Upgrade IP Cores** dialog box, click one or multiple IP components, and then click **Upgrade**.
A green checkmark appears for the IP components that Qsys successfully upgrades.
3. Generate your Qsys system.

Note: Qsys supports command-line upgrade for IP components with the following command:

```
qsys-generate --upgrade-ip-cores <qsys_file>
```

The `<qsys_file>` variable accepts a path to the **.qsys** file so that you are not constrained to running this command in the same directory as the **.qsys** file. Qsys reports the start and finish of the command-line upgrade, but does not name the particular IP component(s) upgraded.

For device migration information, refer to *Introduction to Altera IP Cores*.

Related Information

[Adding IP Component Files to the IP Catalog](#) on page 5-48

[Introduction to Altera IP Cores](#)

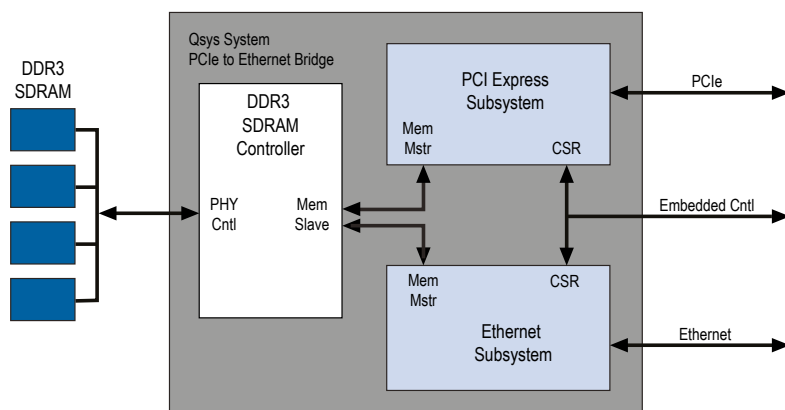
Creating Hierarchical Systems

Qsys supports team-based and hierarchical system design.

You can include any Qsys system as a component in another Qsys system. In a team-based design flow, you can have one or more systems in your design developed simultaneously by other team members, decreasing time-to-market for the complete design.

Figure 5-6: Top-Level of a Hierarchical Design for a PCI Express to Ethernet Bridge

For example, you can use Qsys to combine separate PCI Express and Ethernet subsystems with Altera's DDR3 SDRAM Controller and UniPHY IP cores.



Hierarchical system design in Qsys offers the following advantages:

- Team-based, modular design by dividing large designs into subsystems.
- Design reuse, by allowing you to use any Qsys system as a component.
- Scalability, by allowing you to instantiate multiple instances of a Qsys system.

Adding Systems to the IP Catalog

Any Qsys system is available for use as an IP component in other Qsys systems.

Figure 5-7: IP Catalog Systems



Creating an IP Component Based on a System

The **Export System as hw.tcl Component** command on the File menu allows you to save the system currently open in Qsys as an `_hw.tcl` file to the current project directory. The saved system displays as a new component under the **System** category in the IP Catalog.

Hierarchical System Using Instance Parameters Example

You can use an instance parameter to control the implementation of system IP components from a higher-level Qsys system. You define instance parameters on the **Instance Parameters** tab in Qsys.

The example below shows two instances of the same IP component, `My_IP`, which is a Qsys component with a system identification parameter called `MY_SYSTEM_ID`. In this example, the design requirement specifies that each instantiation of `my_system.qsys` must have unique system ID values to identify the different instances of the system. The value specified by the top-level system is designated `top_id`. In `my_system.qsys`, the component instance `comp0` requires `MY_SYSTEM_ID` set to `top_id + 1`. Instance `comp1` requires `MY_SYSTEM_ID` set to `top_id + 2`.

Example 5-2: Defining the MY_SYSTEM_ID in the My_IP Component

```
add_parameter MY_SYSTEM_ID int 8

set_parameter_property MY_SYSTEM_ID display_name "System ID"

set_parameter_property MY_SYSTEM_ID UNITS None
```

To satisfy the design requirements for this example, define an instance parameter in **my_system.qsys** that is set by the higher-level system. Next, 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.

To do this, in Qsys, open the **my_system.qsys** Qsys system that instantiates the two instances of the **My_IP** 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.

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` sub-components in **my_system.qsys**.

Example 5-3: Hierarchical System Using Instance Parameters

In this example, Qsys gets the value of the parameter `system_id` from the top-level system and saves it as `top_id`. Then Qsys 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.

```
package require qsys 13.1
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 sub-IP-components
    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"
}
```

Click **Preview Instance** to modify the parameter value interactively. View the effect of the scripts in the message panel, which is 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.

Related Information**[Working with Instance Parameters in Qsys](#)**

Managing System Security

TrustZone is the security extension of the ARM[®]-based architecture. It includes secure and non-secure transactions designations, and a protocol for processing between the designations. TrustZone security support is a part of the Qsys interconnect.

The AXI `AXPROT` protection signal specifies a secure or non-secure transaction. When an AXI master sends a command, the `AXPROT` signal specifies whether the command is secure or non-secure. When an AXI slave receives a command, the `AXPROT` signal determines whether the command is secure or non-secure. Determining the security of a transaction while sending or receiving a transaction is a run-time protocol.

The Avalon specification does not include a protection signal as part of its specification. When an Avalon master sends a command, it has no embedded security and Qsys recognizes the command as non-secure. When an Avalon slave receives a command, it also has no embedded security, and the slave always accepts the command and responds.

AXI masters and slaves can be TrustZone-aware. All other master and slave interfaces, such as Avalon-MM interfaces, are non-TrustZone-aware. You can set compile-time security support for all components (except AXI masters, including AXI3, AXI4, and AXI4-Lite) in the **Security** column in the **System Contents** tab, or in the **Interconnect Requirements** tab under the **Identifier** column for the master or slave interface. To begin creating a secure system, you must first add masters and slaves to your system, and the connections between them. After you establish connections between the masters and slaves, you can then set the security options, as needed.

An example of when you may need to specify compile-time security support is when an Avalon master needs to communicate with a secure AXI slave, and you can specify whether the connection point is secure or non-secure. You can specify a compile-time secure address ranges for a memory slave if an interface-level security setting is not sufficient.

Related Information

- [Qsys Interconnect](#)
- [Qsys System Design Components](#)

Understanding Security Settings Between Interfaces

The AXI `AXPROT` signal specifies a transaction as secure or non-secure at runtime when a master sends a transaction. Qsys identifies AXI master interfaces as TrustZone-aware. You can configure AXI slaves as Trustzone-aware, secure, non-secure, or secure ranges.

Table 5-6: Compile-Time Security Options

For non-TrustZone-aware components, compile-time security support options are available in Qsys on the **System Contents** tab, or on the **Interconnect Requirements** tab.

Compile-Time Security Options	Description
Non-secure	Master sends only non-secure transactions, and the slave receives any transaction, secure or non-secure.

Compile-Time Security Options	Description
Secure	Master sends only secure transactions, and the slave receives only secure transactions.
Secure ranges	Applies to only the slave interface. The specified address ranges within the slave's address span are secure, all other address ranges are not. The format is a comma-separated list of inclusive-low and inclusive-high addresses, for example, 0x0:0xffff, 0x2000:0x20ff.

After setting compile-time security options for non-TrustZone-aware master and slave interfaces, you must identify those masters that require a default slave before generation. To designate a slave interface as the default slave, turn on **Default Slave** in the **System Contents** tab. A master can have only one default slave.

Note: The **Security** and **Default Slave** columns in the **System Contents** tab are hidden by default. Right-click the **System Contents** header to select which columns you want to display.

The following are descriptions of security support for master and slave interfaces. These description can guide you in your design decisions when you want to create secure systems that have mixed secure and non-TrustZone-aware components:

- All AXI, AXI4, and AXI4-Lite masters are TrustZone-aware.
- You can set AXI, AXI4, and AXI4-Lite slaves as Trust-Zone-aware, secure, non-secure, or secure range ranges.
- You can set non-AXI master interfaces as secure or non-secure.
- You can set non-AXI slave interfaces as secure, non-secure, or secure address ranges.

Specifying a Default Slave in a Qsys System

If a master issues "per-access" or "not allowed" transactions, your design must contain a default slave. Per-access refers to the ability of a TrustZone-aware master to allow or disallow access or transactions. A transaction that violates security is rerouted to the default slave and subsequently responds to the master with an error. You can designate any slave as the default slave.

You can share a default slave between multiple masters. You should have one default slave for each interconnect domain. An interconnect domain is a group of connected memory-mapped masters and slaves that share the same interconnect. The `altera_axi_default_slave` component includes the required TrustZone features.

You can achieve an optimized secure system by partitioning your design and carefully designating secure or non-secure address maps to maintain reliable data. Avoid a design where, under the same hierarchy, a non-secure master initiates transactions to a secure slave resulting in unsuccessful transfers.

Table 5-7: Secure and Non-Secure Access Between Master, Slave, and Memory Components

Transaction Type	TrustZone-aware Master	Non-TrustZone-aware Master Secure	Non-TrustZone-aware Master Non-Secure
TrustZone-aware slave/ memory	OK	OK	OK

Transaction Type	TrustZone-aware Master	Non-TrustZone-aware Master	Non-TrustZone-aware Master
		Secure	Non-Secure
Non-TrustZone-aware slave (secure)	Per-access	OK	Not allowed
Non-TrustZone-aware slave (non-secure)	OK	OK	OK
Non-TrustZone-aware memory (secure region)	Per-access	OK	Not allowed
Non-TrustZone-aware memory (non-secure region)	OK	OK	OK

Accessing Undefined Memory Regions

When a transaction from a master targets a memory region that is not specified in the slave memory map, it is known as an "access to an undefined memory region." To ensure predictable response behavior when this occurs, you must add a default slave to the design. Undefined memory region accesses are then routed to the default slave, which then terminates the transaction with an error response.

You can designate any memory-mapped slave as a default slave. Altera recommends that you have only one default slave for each interconnect domain in your system. Accessing undefined memory regions can occur in the following cases:

- When there are gaps within the accessible memory map region that are within the addressable range of slaves, but are not mapped.
- Accesses by a master to a region that does not belong to any slaves that is mapped to the master.
- When a non-secured transaction is accessing a secured slave. This applies to only slaves that are secured at compilation time.
- When a read-only slave is accessed with a write command, or a write-only slave is accessed with a read command.

To designate a slave as the default slave, for the selected component, turn on **Default Slave** in the **Systems Content** tab.

Note: If you do not specify the default slave, Qsys automatically assigns the slave at the lowest address within the memory map for the master that issues the request as the default slave.

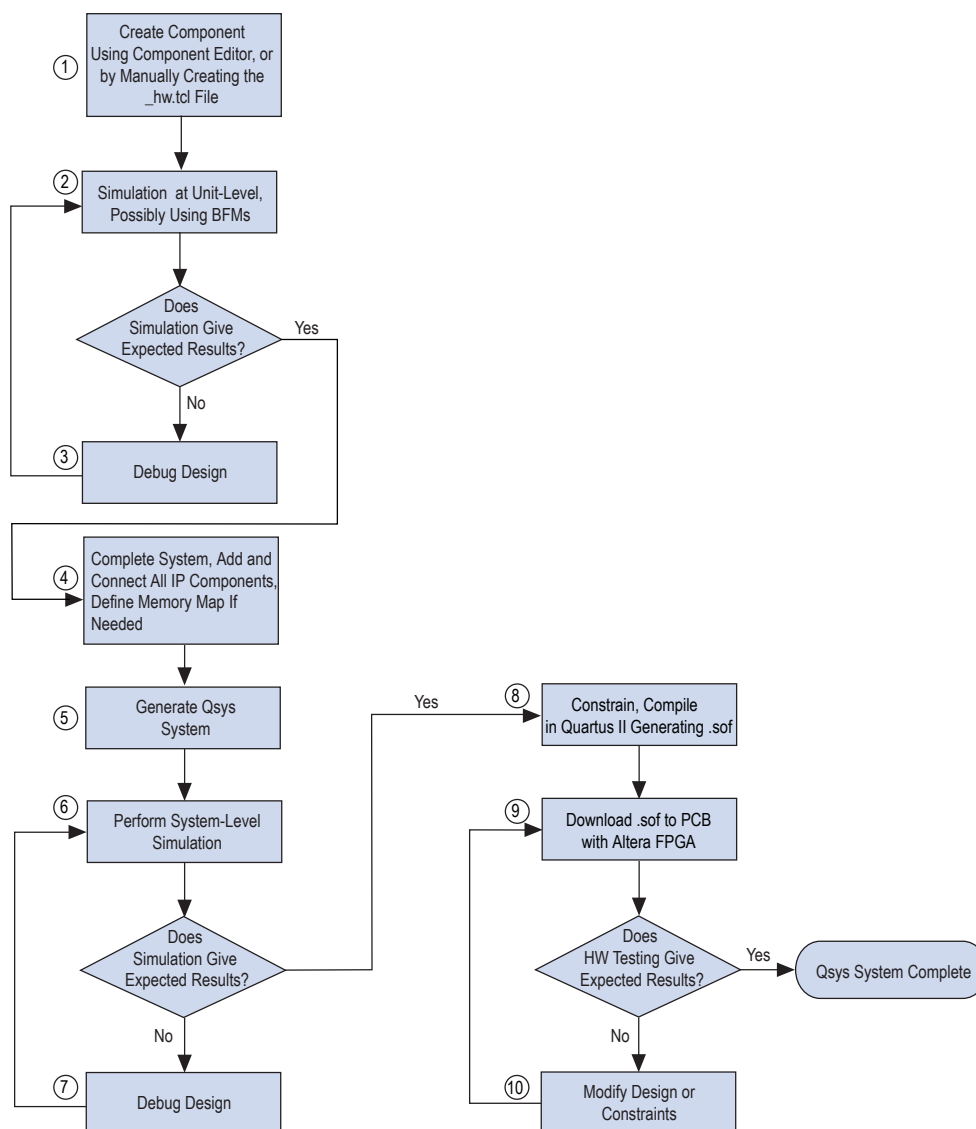
Related Information

[Qsys System Design Components](#)

Qsys Design Flow and Quartus II Software Integration

Figure 5-8: Qsys Design Flow

The design flow below describes how you can create a custom IP component using the Qsys Component Editor, or manually create a `_hw.tcl` file, simulate your custom IP, and then integrate it with other IP components to create a Qsys system and complete Quartus II project.



Note: For information on how to define and generate single IP cores for use in your Quartus II software projects, refer to *Introduction to Altera IP Cores*.

Related Information

- [Creating Qsys Components](#)
- [Introduction to Altera IP Cores](#)

- [Managing Quartus II Projects](#)

Generating a Qsys System

In Qsys, you can choose options for generation of synthesis, simulation and testbench files for your Qsys system.

Qsys system generation creates the interconnect between IP components and generates synthesis and simulation HDL files. You can generate a testbench system that adds Bus Functional Models (BFMs) that interact with your system in a simulator.

When you make changes to a system, Qsys gives you the option to exit without generating. If you choose to generate your system before you exit, the **Generation** dialog box opens and allows you to select generation options.

The **Generate HDL** button in the lower-right of the Qsys window allows you to quickly generate synthesis and simulation files for your system.

Related Information

- [Avalon Verification IP Suite User Guide](#)
- [Mentor Verification IP \(VIP\) Altera Edition \(AE\)](#)

Generating Files for Synthesis and Simulation

The Quartus II software uses Qsys-generated synthesis HDL files during compilation.

In Qsys, you can generate simulation HDL files (**Generate > Generate HDL**), which can include simulation-only features targeted towards your simulator. You can generate simulation files as Verilog, VHDL, or as a mixed-language simulation for use in your simulation environment.

Note: For a list of Altera-supported simulators, refer to *Simulating Altera Designs*.

Qsys supports standard and legacy device generation. Standard device generation refers to generating files for the Arria 10 device, and later device families. Legacy device generation refers to generating files for device families prior to the release of the Arria 10 device, including Max 10 devices.

The **Output Directory** option applies to both synthesis and simulation generation. By default, Qsys generates output files to the Qsys project directory. You can change the default directory in the **Generation** dialog box for legacy devices. For standard devices, the generation directory is fixed to the Qsys project directory.

Note: If you need to change top-level I/O pin or instance names, create a top-level HDL file that instantiates the Qsys system. The Qsys-generated output is then instantiated in your design without changes to the Qsys-generated output files.

The following options in the **Generation** dialog box (**Generate** > **Generate HDL**) allow you to generate synthesis and simulation files:

Option	Description
Create HDL design files for synthesis	Generates Verilog HDL or VHDL design files for the system's top-level definition and child instances for the selected target language. Synthesis file generation is optional.
Create timing and resource estimates for third-party EDA synthesis tools	Generates a non-functional Verilog Design File (.v) for use by some third-party EDA synthesis tools. Estimates timing and resource usage for your IP component. The generated netlist file name is <your_ip_component_name>_syn.v.
Create Block Symbol File (.bsf)	Allows you to optionally create a (.bsf) file to use in a schematic Block Diagram File (.bdf).
Create simulation model	Allows you to optionally generate Verilog HDL or VHDL simulation model files, and simulation scripts.
Allow mixed-language simulation	Generates a simulation model that contains both Verilog and VHDL as specified by the individual IP cores. Using this option, each IP core produces their HDL using it's native implementation, which results in simulation HDL that is easier to understand and faster to simulate. You must have a simulator that supports mixed language simulation. When turned off, Qsys generates all simulation files in the selected simulation model language.

Related Information

- [Simulating Altera Designs](#)

Files Generated for Qsys IP Components

Qsys generates the following files for your Qsys system that are used during synthesis and simulation.

Table 5-8: Qsys-Generated IP Component Files

File Name	Description
<system>.qsys	The Qsys system file. <system> is the name that you give your system.

File Name	Description
<system>.sopcinfo	<p>Describes the connections and IP component parameterizations in your Qsys system. You can parse its contents to get requirements when you develop software drivers for IP components.</p> <p>Downstream tools such as the Nios II tool chain use this file. The .sopcinfo 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.</p>
<system>.cmp	The VHDL Component Declaration (.cmp) file is a text file that contains local generic and port definitions that you can use in VHDL design files.
<system>.html	A system report that contains connection information, a memory map showing the address of each slave with respect to each master to which it is connected, and parameter assignments.
<system>_generation.rpt	Qsys generation log file. A summary of the messages that Qsys issues during system generation.
<system>.debuginfo	Contains post-generation information. Used to pass System Console and Bus Analyzer Toolkit information about the Qsys interconnect. The Bus Analysis Toolkit uses this file to identify debug components in the Qsys interconnect.
<system>.qip	Contains all the required information about the IP component or system to integrate and compile the component in the Quartus II software.
<system>.bsf	A Block Symbol File (.bsf) representation of the top-level Qsys system for use in Quartus II Block Diagram Files (.bdf).
<system>.spd	Input file for <code>ip-make-simscript</code> to generate simulation scripts for supported simulators. The .spd file contains a list of files generated for simulation, including memory initialization files.
<system>.ppf	The Pin Planner File (.ppf) stores the port and node assignments for IP components created for use with the Pin Planner.
<system>_bb.v	You can use the Verilog black-box (_bb.v) file as an empty module declaration for use as a black box.
<system>.sip	Contains information required for NativeLink simulation of IP components. You must add the .sip file to your Quartus II project.
<system>_inst.v or _inst.vhd	HDL example instantiation template. You can copy and paste the contents of this file into your HDL file to instantiate a Qsys system.

File Name	Description
<system>.regmap	If IP in the system contain register information, Qsys generates a .regmap file. The .regmap file describes the register map information of master and slave interfaces. This file complements the .sopcinfo file by providing more detailed register information about the system. This enables register display views and user customizable statistics in the System Console.
<system>.svd	Allows HPS System Debug tools to view the register maps of peripherals connected to HPS within a Qsys system. During synthesis, the .svd files for slave interfaces visible to System Console masters are stored in the .sof file in the debug section. System Console reads this section, which Qsys can query for register map information. For system slaves, Qsys can access the registers by name.
<system>.v or <system>.vhd	HDL files that instantiate each submodule or child IP core in the system for synthesis or simulation.
mentor/	Contains a ModelSim® script msim_setup.tcl to set up and run a simulation.
aldec/	Contains a Riviera-PRO script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script vcs_setup.sh to set up and run a VCS® simulation. Contains a shell script vcsmx_setup.sh and synopsys_sim.setup file 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.
/submodules	Contains HDL files for the submodule of the Qsys system.
<child IP cores>/	For each generated child IP core directory, Qsys generates /synth and /sim subdirectories.

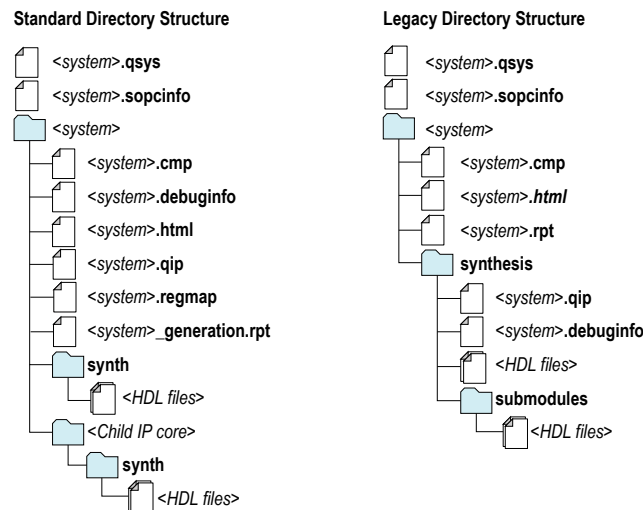
Related Information

- [Qsys Synthesis Standard and Legacy Device Output Directories](#) on page 5-37
- [Qsys Simulation Standard and Legacy Device Output Directories](#) on page 5-38

Qsys Synthesis Standard and Legacy Device Output Directories

The **/synth** or **/synthesis** directories contain the Qsys-generated files that the Quartus II software uses to synthesize your design.

Figure 5-9: Qsys Synthesis Output Directories

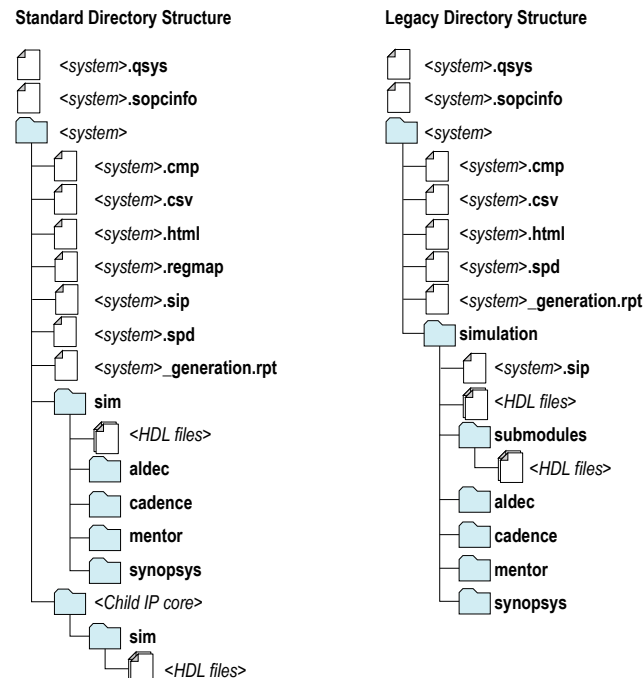
**Related Information**

[Files Generated for Qsys IP Components](#) on page 5-35

Qsys Simulation Standard and Legacy Device Output Directories

The /sim and /simulation directories contain the Qsys-generated output files to simulate your Qsys system.

Figure 5-10: Qsys Simulation Output Directories



Related Information

[Files Generated for Qsys IP Components](#) on page 5-35

Generating Files for a Testbench Qsys System

Qsys testbench is a new system that instantiates the current Qsys system by adding BFM s to drive the top-level interfaces. BFM s interact with the system in the simulator. You can use options in the **Generation** dialog box (**Generate** > **Generate Testbench System**) to generate a testbench Qsys system.

You can generate a standard or simple testbench system with BFM or Mentor Verification IP (for AXI3/AXI4) IP components that drive the external interfaces of your system. Qsys generates a Verilog HDL or VHDL simulation model for the testbench system to use in your simulation tool. You should first generate a testbench system, and then modify the testbench system in Qsys before generating its simulation model. In most cases, you should select only one of the simulation model options.

The **Output Directory** option in the **Generation** dialog box allows you to browse and locate an alternate directory than the project directory for the testbench system.

The following options are available for generating a Qsys testbench system:

Option	Description
Create testbench Qsys system	<ul style="list-style-type: none"> • Standard, BFM s for standard Qsys Interconnect—Creates a testbench Qsys system with BFM IP components attached to exported Avalon and AXI3/AXI4 interfaces. Includes any simulation partner modules specified by IP components in the system. The testbench generator supports AXI interfaces and can connect AXI3/AXI4 interfaces to Mentor Graphics AXI3/AXI4 master/slave BFM s. However, BFM s support address widths only up to 32-bits. • Simple, BFM s for clocks and resets—Creates a testbench Qsys system with BFM IP components driving only clock and reset interfaces. Includes any simulation partner modules specified by IP components in the system.
Create testbench simulation model	Creates Verilog HDL or VHDL simulation model files and simulation scripts for the testbench Qsys system currently open in your workspace. Use this option if you do not need to modify the Qsys-generated testbench before running the simulation.
Allow mixed-language simulation	Generates a mixed-language simulation model. If you have a mixed-language simulator license, generating for mixed-language simulation can shorten the generation time, and produce files that can simulate faster. When turned off, all simulation files are generated in the selected simulation model language.

Files Generated for Qsys Testbench

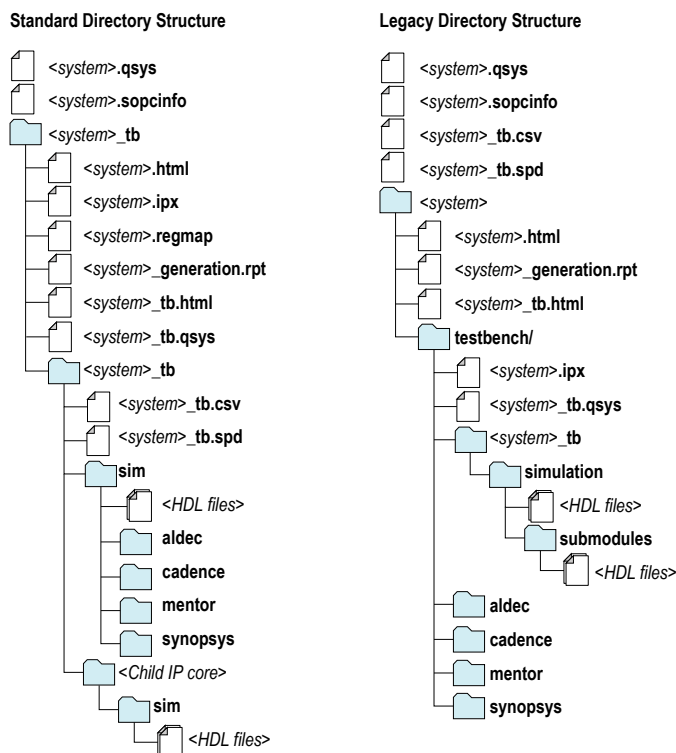
Table 5-9: Qsys-Generated Testbench Files

File Name or Directory Name	Description
<system>_tb.qsys	The Qsys testbench system.
<system>_tb.v or <system>_tb.vhd	The top-level testbench file that connects BFM to the top-level interfaces of <system>_tb.qsys.
<system>_tb.spd	Required input file for ip-make-simscript to generate simulation scripts for supported simulators. The .spd file contains a list of files generated for simulation and information about memory that you can initialize.
<system>.html and <system>_tb.html	A system report that contains connection information, a memory map showing the address of each slave with respect to each master to which it is connected, and parameter assignments.
<system>_generation.rpt	Qsys generation log file. A summary of the messages that Qsys issues during testbench system generation.
<system>.ipx	The IP Index File (.ipx) lists the available IP components, or a reference to other directories to search for IP components.
<system>.svd	Allows HPS System Debug tools to view the register maps of peripherals connected to HPS within a Qsys system. Similarly, during synthesis the .svd files for slave interfaces visible to System Console masters are stored in the .sof file in the debug section. System Console reads this section, which Qsys can query for register map information. For system slaves, Qsys can access the registers by name.
mentor/	Contains a ModelSim script msim_setup.tcl to set up and run a simulation
aldec/	Contains a Riviera-PRO script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script vcs_setup.sh to set up and run a VCS simulation. Contains a shell script vcsmx_setup.sh and synopsys_sim.setup file 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.
/submodules	Contains HDL files for the submodule of the Qsys testbench system.
<child IP cores>/	For each generated child IP core directory, Qsys testbench generates /synth and /sim subdirectories.

Qsys Testbench Simulation Standard and Legacy Device Output Directories

The `/sim` and `/simulation` directories contain the Qsys-generated output files to simulate your Qsys testbench system.

Figure 5-11: Qsys Simulation Testbench Directory Structure



Generate and Modify a Qsys Testbench System

You can use the following steps to create a Qsys testbench system of your Qsys system.

1. Create a Qsys system.
2. Generate a testbench system in the Qsys **Generation** dialog box (**Generate** > **Generate Testbench System**).
3. Open the testbench system in Qsys. Make changes to the BFM, as needed, such as changing the instance names and **VHDL ID** value. For example, you can modify the **VHDL ID** value in the **Altera Avalon Interrupt Source IP** component.
4. If you modify a BFM, re-generate the simulation model for the testbench system.
5. Create a custom test program for the BFM.
6. Compile and load the Qsys system and testbench into your simulator, and then run the simulation.

Simulation Scripts

Qsys generates simulation scripts to set up the simulation environment for Mentor Graphics Modelsim[®] and Questasim[®], Synopsys VCS[®] and VCS MX[®], Cadence Incisive Enterprise Simulator[®] (NCSIM), and the Aldec Riviera-PRO[®] Simulator.

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 system for simulation.

Table 5-10: Simulation Script Variables

The simulation scripts provide variables that allow flexibility in your simulation environment.

Variable	Description
TOP_LEVEL_NAME	If the testbench Qsys 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.
QUARTUS_INSTALL_DIR	Points to the Quartus installation directory that contains the device family library.

Example 5-4: Top-Level Simulation HDL File for a Testbench System

The example below shows the `pattern_generator_tb` generated for a Qsys system called `pattern_generator`. The `top.sv` file defines the top-level module that instantiates the `pattern_generator_tb` simulation model, as well as a custom SystemVerilog test program with BFM transactions, called `test_program`.

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

Note: The VHDL version of the Altera Tristate Conduit BFM is not supported in Synopsys VCS, NCSim, and Riviera-PRO in the Quartus II software version 14.0. These simulators do not support the VHDL protected type, which is used to implement the BFM. For a workaround, use a simulator that supports the VHDL protected type.

Related Information

- [ModelSim-Altera software, Mentor Graphics ModelSim support](#)
- [Synopsys VCS and VCS MX support](#)
- [Cadence Incisive Enterprise Simulator \(IES\) support](#)
- [Aldec Active-HDL and Riviera-PRO support](#)

Simulating Software Running on a Nios II Processor

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

1. In the **Generation** dialog box (**Generate > Generate Testbench System**), select **Simple, BFM for clocks and resets**.
2. For the **Create testbench simulation model** option select **Verilog** or **VHDL**.
3. Click **Generate**.

4. Open the **Nios II Software Build Tools for Eclipse**.
5. Set up an application project and board support package (BSP) for the `<system>` **.sopcinfo** file.
6. To simulate, right-click the application project in Eclipse, and then click **Run as > Nios II ModelSim**. Sets up the ModelSim simulation environment, and compiles and loads the Nios II software simulation.
7. To run the simulation in ModelSim, type `run -all` in the ModelSim transcript window.
8. Set the ModelSim settings and select the Qsys Testbench Simulation Package Descriptor (**.spd**) file, `<system> > _tb.spd`. The **.spd** file is generated with the testbench simulation model for Nios II designs and specifies the files required for Nios II simulation.

Related Information

- [Getting Started with the Graphical User Interface \(Nios II\)](#)
- [Getting Started from the Command-Line \(Nios II\)](#)

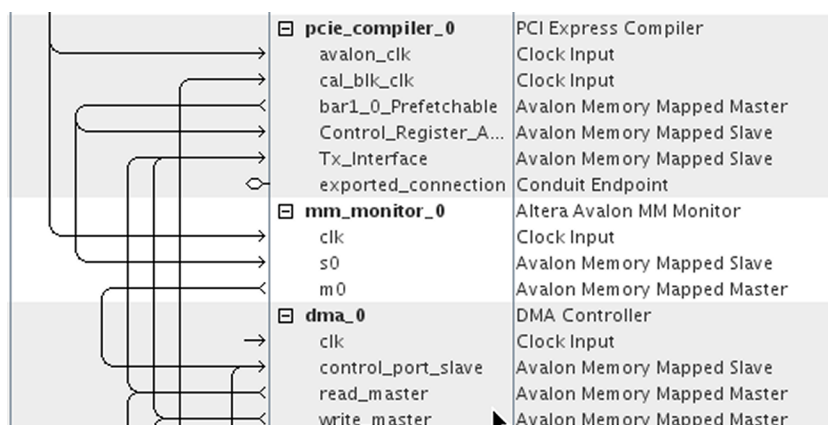
Adding Assertion Monitors for Simulation

You can add monitors to Avalon-MM, AXI, and Avalon-ST interfaces in your system to verify protocol and test coverage with a simulator that supports SystemVerilog assertions.

Note: Modelsim Altera Edition does not support SystemVerilog assertions. If you want to use assertion monitors, you must use a supported third-party simulators such as Mentor Questasim, Synopsys VCS, or Cadence Incisive. For more information, refer to *Introduction to Altera IP Cores*.

Figure 5-12: Inserting an Avalon-MM Monitor Between an Avalon-MM Master and Slave Interface

This example demonstrates the use of a monitor with an Avalon-MM monitor between the `pcie_compiler_bar1_0_Prefetchable` Avalon-MM master interface, and the `dma_0_control_port_slave` Avalon-MM slave interface.



Similarly, you can insert an Avalon-ST monitor between Avalon-ST source and sink interfaces.

Related Information

[Introduction to Altera IP Cores](#)

CMSIS Support for HPS IP Component

Qsys systems that contain an HPS IP component generate a System View Description (**.svd**) file that lists peripherals connected to the ARM processor.

The **.svd** (or CMSIS-SVD) file format is an XML schema specified as part of the Cortex Microcontroller Software Interface Standard (CMSIS) provided by ARM. The **.svd** file allows HPS system debug tools (such as the DS-5 Debugger) to view the register maps of peripherals connected to HPS in a Qsys system.

Related Information

- [Component Interface Tcl Reference](#)
- [CMSIS - Cortex Microcontroller Software](#)

Displaying Qsys Interconnect

The System with Qsys Interconnect window allows you to see the contents of the Qsys interconnect before you generate your system. In this display of your system, you can review a graphical representation of the generated interconnect. Qsys converts connections between interfaces to interconnect logic during system generation.

You access the System with Qsys Interconnect window by clicking **Show System With Qsys Interconnect** command on the System menu.

The System with Qsys Interconnect window has the following tabs:

- **System Contents**—Displays the original instances in your system, as well as the inserted interconnect instances. Connections between interfaces are replaced by connections to interconnect where applicable.
- **Hierarchy**—Displays a system hierarchical navigator, expanding the system contents to show modules, interfaces, signals, contents of subsystems, and connections.
- **Parameters**—Displays the parameters for the selected element in the **Hierarchy** tab.
- **Memory-Mapped Interconnect**—Allows you to select a memory-mapped interconnect module and view its internal command and response networks. You can also insert pipeline stages to achieve timing closure.

The **System Contents**, **Hierarchy**, and **Parameters** tabs are read-only. Edits that you apply on the **Memory-Mapped Interconnect** tab are automatically reflected on the **Interconnect Requirements** tab.

Using the Memory-Mapped Interconnect Tab

The **Memory-Mapped Interconnect** tab in the System with Qsys Interconnect window displays a graphical representation of command and response data paths in your system. Data paths allow you precise control over pipelining in the interconnect. Qsys displays separate figures for the command and response data paths. You can access the data paths by clicking their respective tabs in the **Memory-Mapped Interconnect** tab.

Each node element in a figure represents either a master or slave that communicates over the interconnect, or an interconnect sub-module. Each edge is an abstraction of connectivity between elements, and its direction represents the flow of the commands or responses.

Click **Highlight Mode (Path, Successors, Predecessors)** to identify edges and data paths between modules. Turn on **Show Pipeline Locations** to add greyed-out registers on edges where pipelining is allowed in the interconnect.

Note: You must select more than one module to highlight a path.

Manually Controlling Pipelining in the Qsys Interconnect

The **Memory-Mapped Interconnect** tab allows you to manipulate pipeline connections in the Qsys interconnect. You access the **Memory-Mapped Interconnect** tab by clicking the **Show System With Qsys Interconnect** command on the System menu.

Note: To increase interconnect frequency, you should first try increasing the value of the **Limit interconnect pipeline stages to** option on the **Interconnect Requirements** tab. You should only consider manually pipelining the interconnect if changes to this option do not improve frequency, and you have tried all other options to achieve timing closure, including the use of a bridge. Manually pipelining the interconnect should only be applied to complete systems.

1. In the **Interconnect Requirements** tab, first try increasing the value of the **Limit interconnect pipeline stages to** option until it no longer gives significant improvements in frequency, or until it causes unacceptable effects on other parts of the system.
2. In the Quartus II software, compile your design and run timing analysis.
3. Using the timing report, identify the critical path through the interconnect and determine the approximate mid-point. The following is an example of a timing report:

```
2.800 0.000 cpu_instruction_master|out_shifter[63]|q
3.004 0.204 mm_domain_0|addr_router_001|Equal5~0|datac
3.246 0.242 mm_domain_0|addr_router_001|Equal5~0|combout
3.346 0.100 mm_domain_0|addr_router_001|Equal5~1|dataa
3.685 0.339 mm_domain_0|addr_router_001|Equal5~1|combout
4.153 0.468 mm_domain_0|addr_router_001|src_channel[5]~0|datad
4.373 0.220 mm_domain_0|addr_router_001|src_channel[5]~0|combout
```

4. In Qsys, click **System > Show System With Qsys Interconnect**.
5. In the **Memory-Mapped Interconnect** tab, select the interconnect module that contains the critical path. You can determine the name of the module from the hierarchical node names in the timing report.
6. Click **Show Pipelinable Locations**. Qsys displays all possible pipeline locations in the interconnect. Right-click the possible pipeline location to insert or remove a pipeline stage.
7. Locate the possible pipeline location that is closest to the mid-point of the critical path. The names of the blocks in the memory-mapped interconnect tab correspond to the module instance names in the timing report.
8. Right-click the location where you want to insert a pipeline, and then click **Insert Pipeline**.
9. Regenerate the Qsys system, recompile the design, and then rerun timing analysis. If necessary, repeat the manual pipelining process again until timing requirements are met.

Manual pipelining has the following limitations:

- If you make changes to your original system's connectivity after manually pipelining an interconnect, your inserted pipelines may become invalid. Qsys displays warning messages when you generate your system if invalid pipeline stages are detected. You can remove invalid pipeline stages with the **Remove Stale Pipelines** option in the **Memory-Mapped Interconnect** tab. Altera recommends that you do not make changes to the system's connectivity after manual pipeline insertion.
- Review manually-inserted pipelines when upgrading to newer versions of Qsys. Manually-inserted pipelines in one version of Qsys may not be valid in a future version.

Related Information

- [Specifying \\$system Interconnect Requirements](#) on page 5-11
- [Qsys System Design Components](#)

Integrating a Qsys System with the Quartus II Software

To integrate a Qsys system with your Quartus II project, either the Qsys System File (**.qsys**) or the Quartus II IP File (**.qip**), but never both, must be added to your Quartus II project. Qsys creates the **.qsys** file when you save your Qsys system, and produces the **.qip** file when you generate your Qsys system. Both the **.qsys** and **.qip** files contain the information necessary for compiling your Qsys system within a Quartus II project.

You can choose to include the **.qsys** file automatically in your Quartus II project when you generate your Qsys system by turning on the **Automatically add Quartus II IP files to all projects** option in the Quartus II software (**Tools > Options > IP Settings**). If this option is turned off, the Quartus II software asks you if you want to include the **.qsys** file in your Quartus II project after you exit Qsys.

If you want file generation to occur as part of the Quartus II software's compilation, you should include the **.qsys** file in your Quartus II project. If you want to manually control file generation outside of the Quartus II software, you should include the **.qip** file in your Quartus II project.

Note: The Quartus II software generates an error message during compilation if you add both the **.qsys** and **.qip** files to your Quartus II project.

Does Quartus II Overwrite Qsys-Generated Files During Compilation?

Qsys supports standard and legacy device generation. Standard device generation refers to generating files for the Arria 10 device, and later device families. Legacy device generation refers to generating files for device families prior to the release of the Arria 10 device, including Max 10 devices.

When you integrate your Qsys system with the Quartus II software, if a **.qsys** file is included as a source file, Qsys generates standard device files under **<system>/** next to the location of the **.qsys** file. For legacy devices, if a **.qsys** file is included as a source file, Qsys generates HDL files in the Quartus II project directory under **/db/ip**.

For standard devices, Qsys-generated files are only overwritten during Quartus II compilation if the **.qip** file is removed or missing. For legacy devices, each time you compile your Quartus II project with a **.qsys** file, the Qsys-generated files are overwritten. Therefore, you should not edit Qsys-generated HDL in the **/db/ip** directory; any edits made to these files are lost and never used as input to the Quartus HDL synthesis engine.

Related Information

- [Adding IP Component Files to the IP Catalog](#) on page 5-48
- [Generating a Qsys System](#) on page 5-34
- [Qsys Synthesis Standard and Legacy Device Output Directories](#) on page 5-37
- [Qsys Simulation Standard and Legacy Device Output Directories](#) on page 5-38
- [Introduction to Altera IP Cores](#)
- [Implementing and Parameterizing Memory IP](#)

Integrating a Qsys System and the Quartus II Software With the .qsys File

Use the following steps to integrate your Qsys system and your Quartus II project using the **.qsys** file:

1. In Qsys, create and save a Qsys system.

2. To automatically include the **.qsys** file in the your Quartus II project during compilation, in the Quartus II software, select **Tools > Options > IP Settings**, and turn on **Automatically add Quartus II IP files to all projects**.
3. When the **Automatically add Quartus II IP files to all projects** option is not checked, when you exit Qsys, the Quartus II software displays a dialog box asking whether you want to add the **.qsys** file to your Quartus II project. Click **Yes** to add the **.qsys** file to your Quartus II project.
4. In the Quartus II software, select **Processing > Start Compilation**.

Integrating a Qsys System and the Quartus II Software With the .qip File

Use the following steps to integrate your Qsys system and your Quartus II project using the **.qip** file:

1. In Qsys, create and save a Qsys system.
2. In Qsys, click **Generate HDL**.
3. In the Quartus II software, select **Assignments > Settings > Files**.
4. On the **Files** page, use the controls to locate your **.qip** file, and then add it to your Quartus II project.
5. In the Quartus II software, select **Processing > Start Compilation**.

Setting Clock Constraints

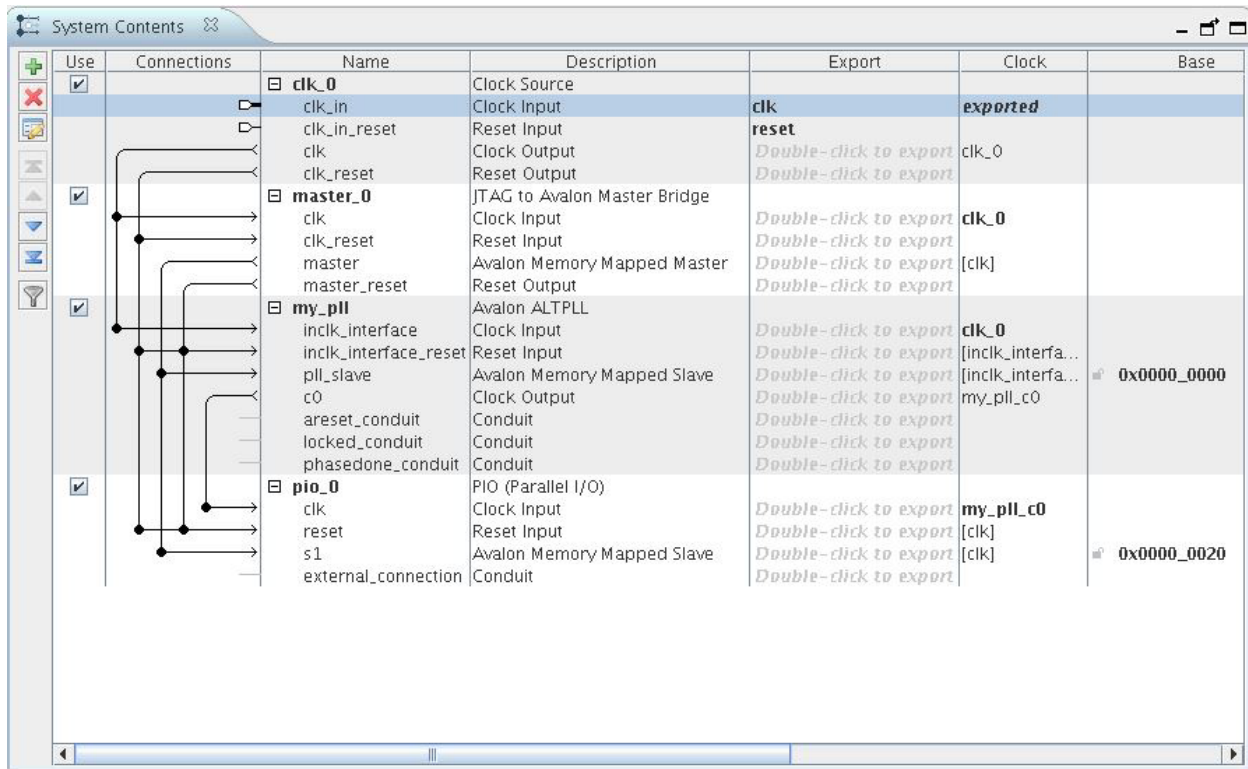
Many IP components include Synopsys Design Constraint (**.sdc**) files that provide timing constraints. Generated **.sdc** files are included in your Quartus II project with the generated **.qip** file. For your top-level clocks and PLLs, you must provide clock and timing constraints in SDC format to direct synthesis and fitting to optimize the design appropriately, and to evaluate performance against timing constraints.

You can specify a base clock assignment for each clock input in the TimeQuest GUI or with the `create_clock` command, and then use the `derive_pll_clocks` command to define the PLL clock output frequencies and phase shifts for all PLLs in the Quartus II project using the **.sdc** file.

Figure 5-13: Single Clock Input Signal

For the case of a single clock input signal called `clk`, and one PLL with a single output, you can use the following commands in your Synopsys Design Constraint (`.sdc`) file:

```
create_clock -name master_clk -period 20 [get_ports {clk}]
derive_pll_clocks
```



Use	Connections	Name	Description	Export	Clock	Base
<input checked="" type="checkbox"/>		clk_0	Clock Source			
		clk_in	Clock Input	clk	exported	
		clk_in_reset	Reset Input	reset		
		clk	Clock Output	Double-click to export	clk_0	
		clk_reset	Reset Output	Double-click to export		
<input checked="" type="checkbox"/>		master_0	JTAG to Avalon Master Bridge			
		clk	Clock Input	Double-click to export	clk_0	
		clk_reset	Reset Input	Double-click to export		
		master	Avalon Memory Mapped Master	Double-click to export	[clk]	
		master_reset	Reset Output	Double-click to export		
<input checked="" type="checkbox"/>		my_pll	Avalon ALTPLL			
		inclk_interface	Clock Input	Double-click to export	clk_0	
		inclk_interface_reset	Reset Input	Double-click to export	[inclk_interfa...	
		pll_slave	Avalon Memory Mapped Slave	Double-click to export	[inclk_interfa...	0x0000_0000
		c0	Clock Output	Double-click to export	my_pll_c0	
		areset_conduit	Conduit	Double-click to export		
		locked_conduit	Conduit	Double-click to export		
		phasedone_conduit	Conduit	Double-click to export		
<input checked="" type="checkbox"/>		pio_0	PIO (Parallel I/O)			
		clk	Clock Input	Double-click to export	my_pll_c0	
		reset	Reset Input	Double-click to export	[clk]	
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0020
		external_connection	Conduit	Double-click to export		

Related Information

[The Quartus II TimeQuest Timing Analyzer](#)

Adding IP Component Files to the IP Catalog

The IP Catalog lists IP components available for use in Qsys systems. IP components can include Altera-provided IP components, third-party IP components, and custom IP components that you provide.

Because Qsys supports hierarchical design, previously created Qsys systems also appear in the IP Catalog. You can connect any interfaces exported from a Qsys system within a parent Qsys system. Additionally, instance parameters set on a Qsys system, can be parameterized within a parent Qsys system.

Altera and third-party developers provide ready-to-use IP components, which are installed with the Quartus II software and appear in the IP Catalog. The Qsys IP Catalog includes the following IP component types:

- Microprocessors, such as the Nios[®] II processor
- DSP IP components, such as the Reed Solomon Decoder 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) IP components, such as the Avalon-ST Multiplexer
- Qsys Interconnect IP components
- Verification IP (VIP) Bus Functional Models (BFMs)

You can use the **IP Search Path** option (**Tools > Options**) to include custom and third-party IP components in the IP Catalog. The IP Catalog displays all IP cores that are found in the IP search path.

Qsys searches the directories listed in the IP search path for the following file types:

- ***_hw.tcl**—Defines a single IP component.
- ***.ipx**—Each IP Index File (**.ipx**) indexes a collection of available IP components, or a reference to other directories to search. In general, **.ipx** files facilitate faster startup for Qsys because Qsys does not traverse directories below the **.ipx** file. Additionally, the **.ipx** file can contain information about the IP component, which eliminates the need for Qsys to read the **_hw.tcl** file to understand the basic attributes of the IP core.

Qsys recursively searches directories specified in the IP search path until it finds a **_hw.tcl** file. When Qsys finds a **_hw.tcl** file in a directory, Qsys does not search subdirectories for additional **_hw.tcl** files.

When you specify an IP search location in Qsys, a recursive descent is annotated by **. A single * signifies a match against any file within the specified directory. However, even if a recursive descent is specified, if Qsys finds a **_hw.tcl** or **.ipx** file, it does not search any subdirectories beyond that level.

Note: When you add a component to the search path, you must refresh your system by clicking **File > Refresh** to update the IP Catalog.

Table 5-11: IP Search Locations

Location	Description
PROJECT_DIR/*	Finds IP components and index files in the Quartus II project directory.
PROJECT_DIR/ip/**/*	Finds IP components and index files in any subdirectory of the /ip subdirectory of the Quartus project directory.

Saving IP Components to the Default Search Directory

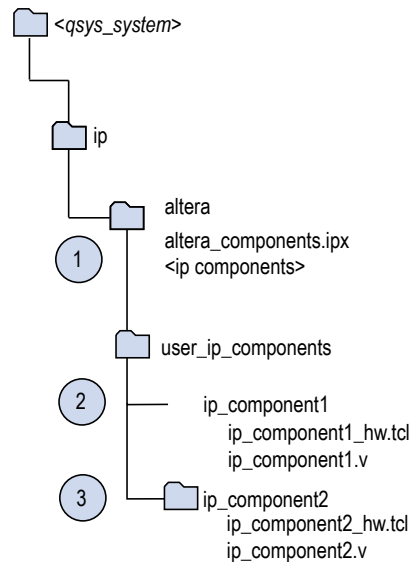
The simplest method to add an IP component to the IP Catalog is to copy the IP component files into an IP search path location.

When you create a component in the Qsys Component Editor, you can save the IP component in your Qsys project directory, or in the **/ip** subdirectory of your Qsys project directory. These methods are useful if you want to associate your IP component with a specific Qsys project.

You can also use the global IP search path in the Quartus II software (**Tools > Options > IP Catalog Search Locations**) to modify the IP search path across all Quartus II projects, Qsys systems, and Altera command line tools.

Figure 5-14: User IP Component Search

In this example, **user_ip_components** is a user-created directory for custom components.



Qsys performs the IP component search algorithm to locate **.ipx** and **_hw.tcl** files, as follows:

1. Qsys recursively searches the **<qsys_system>/ip** directory by default. The recursive search stops when Qsys discovers an **.ipx** file.

Because of this traversal, if changes to the **ip_component1_hw.tcl** file are made, but the **.ipx** file is not rebuilt using **ip-make-ipx**, those changes are not reflected in Qsys because the **.ipx** file contains old information about the **ip_component1** directory.

2. If the **.ipx** file is removed, Qsys discovers both the **ip_component1** and **ip_component2** directories, which contain **_hw.tcl** files.

Note: If you save your **_hw.tcl** file in the **<qsys_system>/ip** directory, Qsys finds your **_hw.tcl** file and will not search subdirectories adjacent to the **_hw.tcl** file. For more information about the IP search path, refer to *Introduction to Altera IP Cores*.

Related Information

[Introduction to Altera IP Cores](#)

Using the IP Index File (.ipx) to Search for IP Components

An IP Index File (**.ipx**) is an XML file that contains a search path that Qsys uses to search for IP components that are available for a Qsys system. You can use the **ip-make-ipx** command to create an **.ipx** file for a directory tree, which can reduce the startup time for Qsys.

You can specify a search path in a **user_components.ipx** file located anywhere in a customizable search path either in Qsys (**Tools > Options**) or the Quartus II software (**Tools > Options > IP Catalog Search Locations**). This method of discovering IP components allows you to add a location that is independent of the default search path. The **user_components.ipx** file directs Qsys to the location of each IP component or directory to search.

A `<path>` element in the `.ipx` file specifies a directory where multiple IP components may be found. A `<component>` entry specifies the path to a single component. A `<path>` element can use wildcards in its definition. An asterisk matches any file name. If you use an asterisk as a directory name, it matches any number of subdirectories.

Example 5-5: Path Element in an .ipx File

```
<library>
  <path path="...<user directory>" />
  <path path="...<user directory>" />
  ...
  <component ... file="...<user directory>" />
  ...
</library>
```

A `<component>` element in an `.ipx` file contains several attributes to define a component. If you provide the required details for each component in an `.ipx` file, the startup time for Qsys is less than if Qsys must discover the files in a directory. The example below shows two `<component>` elements. Note that the paths for file names are specified relative to the `.ipx` file.

Example 5-6: Component Element in an .ipx File

```
<library>
  <component
    name="A Qsys Component"
    displayName="Qsys FIR Filter Component"
    version="2.1"
    file="./components/qsys_filters/fir_hw.tcl"
  />
  <component
    name="rgb2cmymk_component"
    displayName="RGB2CMYK Converter(Color Conversion Category!)"
    version="0.9"
    file="./components/qsys_converters/color/rgb2cmymk_hw.tcl"
  />
</library>
```

Note: You can verify that IP components are available with the `ip-catalog` command.

Integrating Third-Party IP Components

You can use IP components created by Altera partners in your Qsys systems. These IP components have interfaces that are supported by Qsys, such as Avalon-MM or AXI. Additionally, some include timing and placement constraints, software drivers, simulation models, and reference designs.

To locate supported third-party IP components on Altera's web page, navigate to the *Intellectual Property & Reference Designs* page, type Qsys Certified in the **Search** box, select **IP Core & Reference Designs**, and then press **Enter**.

Refer to Altera's *Intellectual Property & Reference Designs* page for more information.

Related Information

[Intellectual Property & Reference Designs](#)

Qsys 64-Bit Addressing Support

Qsys interconnect supports up to 64-bit addressing for all Qsys interfaces and IP components, with a range of: 0x0000 0000 0000 0000 to 0xFFFF FFFF FFFF FFFF, inclusive.

Address parameters appear in the **Base** and **End** columns in the **System Contents** tab, on the **Address Map** tab, in the parameter editor, and in validation messages. Qsys displays as many digits as needed in order to display the top-most set bit, for example, 12 hex digits for a 48-bit address.

A Qsys system can have multiple 64-bit masters, with each master having its own address space. You can share slaves between masters and masters can map slaves to different addresses. For example, one master can interact with slave 0 at base address 0000_0000_0000, and another master can see the same slave at base address c000_000_000.

Quartus II debugging tools provide access to the state of an addressable system via the Avalon-MM interconnect. These are also 64-bit compatible, and process within a 64-bit address space, including a JTAG to Avalon master bridge.

For more information on design practices when using slaves with large address spans, refer to *Address Span Extender* in volume 1 of the *Quartus II Handbook*.

Related Information

[Qsys System Design Components](#)

Support for Avalon-MM Non-Power of Two Data Widths

Qsys requires that you connect all multi-point Avalon-MM connections to interfaces with data widths that are equal to powers of two.

Qsys issues a validation error if an Avalon-MM master or slave interface on a multi-point connection is parameterized with a non-power of two data width.

Note: Avalon-MM point-to-point connections between an Avalon-MM master and an Avalon-MM slave are an exception and may set their data widths to a non-power of two.

Viewing the HDL Example

Click **Generate > HDL Example** to generate a template for the top-level HDL definition of your Qsys system in either Verilog HDL or VHDL. The HDL template displays the IP component declaration.

You can copy and paste the example into a top-level HDL file that instantiates the Qsys system, if the Qsys system is not the top-level module in your Quartus II project.

Qsys System Example Designs

Click the **Example Design** button in the parameter editor to generate an example design.

If there are multiple example designs for an IP component, then there is a button for each example in the parameter editor. When you click the **Example Design** button, the **Select Example Design Directory** dialog box appears, where you can select the directory to save the example design.

The **Example Design** button does not appear in the parameter editor if there is no example. For some IP components, you can click **Generate > Example Designs** to access an example design.

The following Qsys system example designs demonstrate various design features and flows that you can replicate in your Qsys system.

Related Information

- [NIOS II Qsys Example Design](#)
- [PCI Express Avalon-ST Qsys Example Design](#)
- [Triple Speed Ethernet Qsys Example Design](#)

Using Qsys Command-Line Utilities

You can perform many of the functions available in the Qsys GUI from the command-line with the `qsys-edit`, `qsys-generate`, and `qsys-script` utilities.

You run Qsys command-line executables from the Quartus II installation directory:

```
<Quartus II installation directory>\quartus\sopc_builder\bin
```

You can use `qsys-generate` to generate a Qsys system or IP variation outside of the Qsys GUI. You can use `qsys-script` to create, manipulate or manage a Qsys system with command-line scripting.

For command-line help listing all options for these executables, type the following command:

```
<Quartus II installation directory>\quartus\sopc_builder\bin\<executable name> --help
```

Example 5-7: Qsys Command-Line Scripting Example

```
qsys-script --script=my_script.tcl \  
--system-file=fancy.qsys my_script.tcl contains:  
package require -exact qsys 13.1  
# get all instance names in the system and print one by one  
set instances [ get_instances ]  
foreach instance $instances {  
    send_message Info "$instance"  
}
```

Note: You must add `$QUARTUS_ROOTDIR/sopc_builder/bin/` to the `PATH` variable to access command-line utilities. Once you add this `PATH` variable, you can launch the utilities from any directory location.

Related Information

- [Working with Instance Parameters in Qsys](#)
- [Altera Wiki Qsys Scripts](#)

Running the Qsys Editor with `qsys-edit`

You can use the `qsys-edit` utility to run the Qsys editor from the command-line.

You can use the following options with the `qsys-edit` utility:

Table 5-12: qsys-edit Command-Line Options

Option	Usage	Description
<i>1st arg file</i>	Optional	The name of the .qsys system or .qvar variation file to edit.
<code>--search-path[=<value>]</code>	Optional	If omitted, Qsys uses a standard default path. If provided, Qsys searches a comma-separated list of paths. To include the standard path in your replacement, use "\$", for example: <code>/extra/dir.\$</code> .
<code>--project-directory=<directory></code>	Optional	Allows you to find IP components in certain locations relative to the project, if any. By default, the current directory is: <code>'.'</code> . To exclude any project directory, use <code>''</code> .
<code>--new-component-type=<value></code>	Optional	Allows you to specify the kind of instance that is parameterized in a variation.
<code>--debug</code>	Optional	Enables debugging features and output.
<code>--host-controller</code>	Optional	Launches the application with an XML host controller interface on standard input/output.
<code>--jvm-max-heap-size=<value></code>	Optional	The maximum memory size Qsys uses for allocations when running <code>qsys-edit</code> . You specify this value as <code><size><unit></code> , where unit is <code>m</code> (or <code>M</code>) for multiples of megabytes, or <code>g</code> (or <code>G</code>) for multiples of gigabytes. The default value is 512m.
<code>--help</code>	Optional	Display help for <code>qsys-edit</code> .

Generating a Qsys System or IP Variation with qsys-generate

You can use the `qsys-generate` utility to generate RTL for your Qsys system, IP core variation, or simulation models and scripts. You can create testbench systems for testing your Qsys system in a simulator using bus functional models (BFMs). Output from the `qsys-generate` command is the same as when generating using the Qsys GUI.

Table 5-13: qsys-generate Command-Line Options

Option	Usage	Description
<i><1st arg file></i>	Required	The name of the .qsys system file to generate.

Option	Usage	Description
<code>--synthesis=<VERILOG VHDL></code>	Optional	Creates synthesis HDL files that Qsys uses to compile the system in a Quartus II project. You must specify the preferred generation language for the top-level RTL file for the generated Qsys system.
<code>--block-symbol-file</code>	Optional	Creates a Block Symbol File (.bsf) for the Qsys system.
<code>--simulation=<VERILOG VHDL></code>	Optional	Creates a simulation model for the Qsys system. The simulation model contains generated HDL files for the simulator, and may include simulation-only features. You must specify the preferred simulation language.
<code>--testbench=<SIMPLE STANDARD></code>	Optional	Creates a testbench system that instantiates the original system, adding bus functional models (BFMs) to drive the top-level interfaces. When you generate the system, the BFMs interact with the system in the simulator.
<code>--testbench-simulation=<VERILOG VHDL></code>	Optional	After you create the testbench system, you can create a simulation model for the testbench system.
<code>--search-path=<value></code>	Optional	If you omit this command, Qsys uses a standard default path. If you provide this command, Qsys searches a comma-separated list of paths. To include the standard path in your replacement, use "\$", for example, "/extra/dir,\$".
<code>--jvm-max-heap-size=<value></code>	Optional	The maximum memory size that Qsys uses for allocations when running <code>qsys-generate</code> . You specify the value as <code><size> <unit></code> , where <code>unit</code> is m (or M) for multiples of megabytes or g (or G) for multiples of gigabytes. The default value is 512m.
<code>--family=<value></code>	Optional	Specifies the device family.
<code>--part=<value></code>	Optional	Specifies the device part number. If set, this option overrides the <code>--family</code> option.



Option	Usage	Description
<code>--allow-mixed-language-simulation</code>	Optional	Enables a mixed language simulation model generation. If true, if a preferred simulation language is set, Qsys uses a <code>fileset</code> of the component for the simulation model generation. When false, which is the default, Qsys uses the language specified with <code>--file-set=<value></code> for all components for simulation model generation.

Displaying Available IP Components with ip-catalog

The `ip-catalog` command displays a list of available IP components relative to the current Quartus II project directory. Use the following format for the `ip-catalog` command:

```
ip-catalog
[--project-dir=<directory>]
[--name=<value>]
[--verbose]
[--xml]
[--help]
```

Table 5-14: ip-catalog Command-Line Options

Option	Usage	Description
<code>--project-dir= <directory></code>	Optional	Finds IP components relative to the Quartus II project directory. By default, Qsys uses '.' as the current directory. To exclude a project directory, leave the value empty.
<code>--name=<value></code>	Optional	Provides a pattern to filter the names of the IP components found. To show all IP components, use a '*' or '.'. By default, Qsys shows all IP components. The argument is not case sensitive.
<code>--verbose</code>	Optional	Reports the progress of the command.
<code>--xml</code>	Optional	Generates the output in XML format, in place of colon-delimited format.
<code>--help</code>	Optional	Displays help for the <code>ip-catalog</code> command.

Creating an .ipx File with ip-make-ipx

The `ip-make-ipx` command creates an **.ipx** file and is a convenient way to include a collection of IP components from an arbitrary directory. You can edit the **.ipx** file to disable visibility of one or more IP components in the IP Catalog. Use the following format for the `ip-make-ipx` command:

```
ip-make-ipx
[--source-directory=<directory>]
```



```
[--output=<file>]  
[--relative-vars=<value>]  
[--thorough-descent]  
[--message-before=<value>]  
[--message-after=<value>]  
[--help]
```

Table 5-15: ip-make-ipx Command-Line Options

Option	Usage	Description
--source-directory=<directory>	Optional	Specifies the root director(ies) that Qsys uses to find the IP component files. The default directory is “.”. You can provide a comma-separated list of directories.
--output=<file>	Optional	Specifies the name of the file to generate. The default name is /component.ipx.
--relative-vars=<value>	Optional	Causes the output file to include references relative to the specified variable(s) where possible. You can specify multiple variables as a comma-separated list.
--thorough-descent	Optional	If set, a component or .ipx file in a directory does not stop Qsys from searching subdirectories.
--message-before=<value>	Optional	Prints a message: stdout when indexing begins.
--message-after=<value>	Optional	Sends a message: stdout when indexing is done.
--help	Optional	Displays help for the ip-make-ipx command.

Generating a Qsys System with qsys-script

You can use the `qsys-script` utility to create and manipulate a Qsys system with Tcl scripting commands.

Note: You must provide a package version for the `qsys-script`. If you do not specify the `--package-version=<value>` command, you must then provide a Tcl script and request the system scripting API directly with the `package require -exact qsys <version>` command.

Table 5-16: qsys-script Command-Line Options

Option	Usage	Description
--system-file=<file>	Optional	Specifies the path to a .qsys file. Qsys loads the system before running scripting commands.
--script=<file>	Optional	A file that contains Tcl scripting commands that you can use to create or manipulate Qsys systems. If you specify both <code>--cmd</code> and <code>--script</code> , Qsys runs the <code>--cmd</code> commands before the script specified by <code>--script</code> .

Option	Usage	Description
<code>--cmd=<value></code>	Optional	A string that contains Tcl scripting commands that you can use to create or manipulate a Qsys system. If you specify both <code>--cmd</code> and <code>--script</code> , Qsys runs the <code>--cmd</code> commands before the script specified by <code>--script</code> .
<code>--package-version=<value></code>	Optional	Specifies which Tcl API scripting version to use and determines the functionality and behavior of the Tcl commands. The Quartus II software supports Tcl API scripting commands. If you do not specify the version on the command-line, your script must request the scripting API directly with the <code>package require -exact qsys <version></code> command.
<code>--search-path=<value></code>	Optional	If you omit this command, a Qsys uses a standard default path. If you provide this command, Qsys searches a comma-separated list of paths. To include the standard path in your replacement, use "\$", for example, <code>/<directory path>/dir,\$</code> . Separate multiple directory references with a comma.
<code>--jvm-max-heap-size=<value></code>	Optional	The maximum memory size that the <code>qsys-script</code> tool uses. You specify this value as <code><size><unit></code> , where unit is <code>m</code> (or <code>M</code>) for multiples of megabytes, or <code>g</code> (or <code>G</code>) for multiples of gigabytes.
<code>--help</code>	Optional	Displays help for the <code>qsys-script</code> utility.

Qsys Scripting Command Reference

The following are Qsys scripting commands:

[add_connection](#) on page 5-62

[add_instance](#) on page 5-63

[add_interface](#) on page 5-64

[apply_preset](#) on page 5-65

[auto_assign_base_addresses](#) on page 5-66

[auto_assign_system_base_addresses](#) on page 5-67

[auto_assign_irqs](#) on page 5-68

[auto_connect](#) on page 5-69

[create_system](#) on page 5-70

[export_hw_tcl](#) on page 5-71

[get_composed_connection_parameter_value](#) on page 5-72

[get_composed_connection_parameters](#) on page 5-73

[get_composed_connections](#) on page 5-74

[get_composed_instance_assignment](#) on page 5-75

[get_composed_instance_assignments](#) on page 5-76

[get_composed_instance_parameter_value](#) on page 5-77

[get_composed_instance_parameters](#) on page 5-78

[get_composed_instances](#) on page 5-79

[get_connection_parameter_property](#) on page 5-80

[get_connection_parameter_value](#) on page 5-81

[get_connection_parameters](#) on page 5-82

[get_connection_properties](#) on page 5-83

[get_connection_property](#) on page 5-84

[get_connections](#) on page 5-85

[get_instance_assignment](#) on page 5-86

[get_instance_assignments](#) on page 5-87

[get_instance_documentation_links](#) on page 5-88

[get_instance_interface_assignment](#) on page 5-89

[get_instance_interface_assignments](#) on page 5-90

[get_instance_interface_parameter_property](#) on page 5-91

[get_instance_interface_parameter_value](#) on page 5-92

[get_instance_interface_parameters](#) on page 5-93

[get_instance_interface_port_property](#) on page 5-94

[get_instance_interface_ports](#) on page 5-95

[get_instance_interface_properties](#) on page 5-96

[get_instance_interface_property](#) on page 5-97

[get_instance_interfaces](#) on page 5-98

[get_instance_parameter_property](#) on page 5-99

[get_instance_parameter_value](#) on page 5-100

[get_instance_parameters](#) on page 5-101

[get_instance_port_property](#) on page 5-102

[get_instance_properties](#) on page 5-103

[get_instance_property](#) on page 5-104

[get_instances](#) on page 5-105

[get_interconnect_requirement](#) on page 5-106

[get_interconnect_requirements](#) on page 5-107

[get_interface_port_property](#) on page 5-108

[get_interface_ports](#) on page 5-109

[get_interface_properties](#) on page 5-110

[get_interface_property](#) on page 5-111

[get_interfaces](#) on page 5-112

[get_module_properties](#) on page 5-113

[get_module_property](#) on page 5-114

[get_parameter_properties](#) on page 5-115

[get_port_properties](#) on page 5-116

[get_project_properties](#) on page 5-117

[get_project_property](#) on page 5-118

[load_system](#) on page 5-119

[lock_avalon_base_address](#) on page 5-120

[remove_connection](#) on page 5-121

[remove_dangling_connections](#) on page 5-122

[remove_instance](#) on page 5-123

[remove_interface](#) on page 5-124

[save_system](#) on page 5-125

[send_message](#) on page 5-126

[set_connection_parameter_value](#) on page 5-127

[set_instance_parameter_value](#) on page 5-128

[set_instance_property](#) on page 5-129

[set_interconnect_requirement](#) on page 5-130

[set_interface_property](#) on page 5-131

[set_module_property](#) on page 5-132

[set_project_property](#) on page 5-133

[set_use_testbench_naming_pattern](#) on page 5-134

[set_validation_property](#) on page 5-135

[unlock_avalon_base_address](#) on page 5-136

[validate_connection](#) on page 5-137

[validate_instance](#) on page 5-138

[validate_instance_interface](#) on page 5-139

[validate_system](#) on page 5-140

`add_connection`

Description

Connects the named interfaces using an appropriate connection type. Both interface names consist of a child instance name, followed by the name of an interface provided by that module. For example, `mux0.out` is the interface named `out` on the instance named `mux0`. Be careful to connect the start to the end, and not the reverse.

Usage

```
add_connection <start> [<end>]
```

Returns

No return value.

Arguments

start

The start interface that is connected, in `<instance_name>.<interface_name>` format. If the end argument is omitted, the connection must be of the form `<instance1>.<interface>/<instance2>.<interface>`.

end (optional)

The end interface that is connected, `<instance_name>.<interface_name>`.

Example

```
add_connection dma.read_master sdram.s1
```

Related Information

- [get_connection_parameter_value](#) on page 5-81
- [get_connection_property](#) on page 5-84
- [get_connections](#) on page 5-85
- [remove_connection](#) on page 5-121
- [set_connection_parameter_value](#) on page 5-127

add_instance

Description

Adds an instance of a component, referred to as a *child* or *child instance*, to the system.

Usage

```
add_instance <name> <type> [<version>]
```

Returns

No return value.

Arguments

name

Specifies a unique local name that you can use to manipulate the instance. Qsys uses this name in the generated HDL to identify the instance.

type

Refers to a kind of instance available in the IP Catalog, for example `altera_avalon_uart`.

version (optional)

The required version of the specified instance type. If no version is specified, Qsys uses the latest version.

Example

```
add_instance uart_0 altera_avalon_uart
```

Related Information

- [get_instance_parameter_value](#) on page 5-100
- [get_instance_property](#) on page 5-104
- [get_instances](#) on page 5-105
- [remove_instance](#) on page 5-123
- [set_instance_parameter_value](#) on page 5-128

add_interface

Description

Adds an interface to your system, which Qsys uses to export an interface from within the system. You specify the exported internal interface with `set_interface_property <interface> EXPORT_OF instance.interface`.

Usage

```
add_interface <name> <type> <direction>.
```

Returns

No return value.

Arguments

name

The name of the interface that Qsys exports from the system.

type

The type of interface.

direction

The interface direction.

Example

```
add_interface my_export conduit end  
set_interface_property my_export EXPORT_OF uart_0.external_connection
```

Related Information

- [get_interface_ports](#) on page 5-109
- [get_interface_properties](#) on page 5-110
- [get_interface_property](#) on page 5-111
- [set_interface_property](#) on page 5-131

apply_preset

Description

Applies the settings in a preset to the specified instance.

Usage

```
apply_preset <instance> <preset_name>
```

Returns

No return value.

Arguments

instance

The name of the instance.

preset_name

The name of the preset.

Example

```
apply_preset cpu_0 "Custom Debug Settings"
```

auto_assign_base_addresses

Description

Assigns base addresses to all memory mapped interfaces on an instance in the system. Instance interfaces that are locked with `lock_avalon_base_address` keep their addresses during address auto-assignment.

Usage

```
auto_assign_base_addresses <instance>
```

Returns

No return value.

Arguments

instance

The name of the instance with memory mapped interfaces.

Example

```
auto_assign_base_addresses sdram
```

Related Information

- [auto_assign_system_base_addresses](#) on page 5-67
- [lock_avalon_base_address](#) on page 5-120
- [unlock_avalon_base_address](#) on page 5-136

auto_assign_system_base_addresses

Description

Assigns legal base addresses to all memory mapped interfaces on all instances in the system. Instance interfaces that are locked with `lock_avalon_base_address` keep their addresses during address auto-assignment.

Usage

```
auto_assign_system_base_addresses
```

Returns

No return value.

Arguments

No arguments.

Example

```
auto_assign_system_base_addresses
```

Related Information

- [auto_assign_base_addresses](#) on page 5-66
- [lock_avalon_base_address](#) on page 5-120
- [unlock_avalon_base_address](#) on page 5-136

auto_assign_irqs

Description

Assigns interrupt numbers to all connected interrupt senders on an instance in the system.

Usage

```
auto_assign_irqs <instance>
```

Returns

No return value.

Arguments

instance

The name of the instance with an interrupt sender.

Example

```
auto_assign_irqs uart_0
```

auto_connect

Description

Creates connections from an instance or instance interface to matching interfaces in other instances in the system. For example, Avalon-MM slaves connect to Avalon-MM masters.

Usage

```
auto_connect <element>
```

Returns

No return value.

Arguments

element

The name of the instance interface, or the name of an instance.

Example

```
auto_connect sdram  
auto_connect uart_0.s1
```

Related Information

[add_connection](#) on page 5-62

create_system

Description

Replaces the current system with a new system with the specified name.

Usage

```
create_system [<name>]
```

Returns

No return value.

Arguments

name (optional)

The name of the new system.

Example

```
create_system my_new_system_name
```

Related Information

- [load_system](#) on page 5-119
- [save_system](#) on page 5-125

export_hw_tcl

Description

Allows you to save the currently open system as an **_hw.tcl** file in the project directory. The saved systems appears under the **System** category in the IP Category.

Usage

```
export_hw_tcl
```

Returns

No return value.

Arguments

No arguments

Example

```
export_hw_tcl
```

get_composed_connection_parameter_value

Description

Returns the value of a parameter in a connection in a child instance containing a subsystem.

Usage

```
get_composed_connection_parameter_value <instance> <child_connection> <parameter>
```

Returns

The parameter value.

Arguments

instance

The child instance that contains a subsystem

child_connection

The name of the connection in the subsystem

parameter

The name of the parameter to query on the connection.

Example

```
get_composed_connection_parameter_value subsystem_0 cpu.data_master/memory.s0  
baseAddress
```

Related Information

- [get_composed_connection_parameters](#) on page 5-73
- [get_composed_connections](#) on page 5-74

get_composed_connection_parameters

Description

Returns a list of all connections in the subsystem, for an instance that contains a subsystem.

Usage

```
get_composed_connection_parameters <instance> <child_connection>
```

Returns

A list of parameter names.

Arguments

instance

The child instance containing a subsystem.

child_connection

The name of the connection in the subsystem.

Example

```
get_composed_connection_parameters subsystem_0 cpu.data_master/memory.s0
```

Related Information

- [get_composed_connection_parameter_value](#) on page 5-72
- [get_composed_connections](#) on page 5-74

get_composed_connections

Description

For an instance that contains a subsystem of the Qsys system, returns a list of all connections found in a subsystem.

Usage

```
get_composed_connections <instance>
```

Returns

A list of connection names in the subsystem. These connection names are not qualified with the instance name.

Arguments

instance

The child instance containing a subsystem.

Example

```
get_composed_connections subsystem_0
```

Related Information

- [get_composed_connection_parameter_value](#) on page 5-72
- [get_composed_connection_parameters](#) on page 5-73

get_composed_instance_assignment

Description

For an instance that contains a subsystem of the Qsys system, returns the value of an assignment found on the instance in the subsystem.

Usage

```
get_composed_instance_assignment <instance> <child_instance> <assignment>
```

Returns

The value of the assignment.

Arguments

instance

The child instance containing a subsystem.

child_instance

The name of a child instance found in the subsystem.

assignment

The assignment key.

Example

```
get_composed_instance_assignment subsystem_0 video_0 "embeddedsd.CMacro.colorSpace"
```

Related Information

- [get_composed_instance_assignments](#) on page 5-76
- [get_composed_instances](#) on page 5-79

get_composed_instance_assignments

Description

For an instance that contains a subsystem of the Qsys system, returns a list of assignments found on the instance in the subsystem.

Usage

```
get_composed_instance_assignments <instance> <child_instance>
```

Returns

A list of assignment names.

Arguments

instance

The child instance containing a subsystem.

child_instance

The name of a child instance found in the subsystem.

Example

```
get_composed_instance_assignments subsystem_0 cpu
```

Related Information

- [get_composed_instance_assignment](#) on page 5-75
- [get_composed_instances](#) on page 5-79

get_composed_instance_parameter_value

Description

For an instance that contains a subsystem of the Qsys system, returns the value of a parameters found on the instance in the subsystem.

Usage

```
get_composed_instance_parameter_value <instance> <child_instance> <parameter>
```

Returns

The value of a parameter on the instance in the subsystem.

Arguments

instance

The child instance containing a subsystem.

child_instance

The name of a child instance found in the subsystem.

parameter

The name of the parameter to query on the instance in the subsystem.

Example

```
get_composed_instance_parameter_value subsystem_0 cpu DATA_WIDTH
```

Related Information

- [get_composed_instance_parameters](#) on page 5-78
- [get_composed_instances](#) on page 5-79

get_composed_instance_parameters

Description

For an instance that contains a subsystem of the Qsys system, returns a list of parameters found on the instance in the subsystem.

Usage

```
get_composed_instance_parameters <instance> <child_instance>
```

Returns

A list of parameter names.

Arguments

instance

The child instance containing a subsystem.

child_instance

The name of a child instance found in the subsystem.

Example

```
get_composed_instance_parameters subsystem_0 cpu
```

Related Information

- [get_composed_instance_parameter_value](#) on page 5-77
- [get_composed_instances](#) on page 5-79

get_composed_instances

Description

For an instance that contains a subsystem of the Qsys system, returns a list of child instances found in the subsystem.

Usage

```
get_composed_instances <instance>
```

Returns

A list of instance names found in the subsystem.

Arguments

instance

The child instance containing a subsystem.

Example

```
get_composed_instances subsystem_0
```

Related Information

- [get_composed_instance_assignment](#) on page 5-75
- [get_composed_instance_assignments](#) on page 5-76
- [get_composed_instance_parameter_value](#) on page 5-77
- [get_composed_instance_parameters](#) on page 5-78

get_connection_parameter_property

Description

Returns the value of a property on a parameter in a connection. Parameter properties are metadata about how Qsys uses the parameter.

Usage

```
get_connection_parameter_property <connection> <parameter> <property>
```

Returns

The value of the parameter property.

Arguments

connection

The connection to query.

parameter

The name of the parameter.

property

The property of the connection. Refer to *Parameter Properties*.

Example

```
get_connection_parameter_property cpu.data_master/dma0.csr baseAddress UNITS
```

Related Information

- [get_connection_parameter_value](#) on page 5-81
- [get_connection_property](#) on page 5-84
- [get_connections](#) on page 5-85
- [get_parameter_properties](#) on page 5-115
- [Parameter Properties](#) on page 5-150

get_connection_parameter_value

Description

Returns the value of a parameter on the connection. Parameters represent aspects of the connection that you can modify, such as the base address for an Avalon-MM connection.

Usage

```
get_connection_parameter_value <connection> <parameter>
```

Returns

The value of the parameter.

Arguments

connection

The connection to query.

parameter

The name of the parameter.

Example

```
get_connection_parameter_value cpu.data_master/dma0.csr baseAddress
```

Related Information

- [get_connection_parameters](#) on page 5-82
- [get_connections](#) on page 5-85
- [set_connection_parameter_value](#) on page 5-127

get_connection_parameters

Description

Returns a list of parameters found on a connection.

Usage

```
get_connection_parameters <connection>
```

Returns

A list of parameter names.

Arguments

connection

The connection to query.

Example

```
get_connection_parameters cpu.data_master/dma0.csr
```

Related Information

- [get_connection_parameter_property](#) on page 5-80
- [get_connection_parameter_value](#) on page 5-81
- [get_connection_property](#) on page 5-84

get_connection_properties

Description

Returns a list of properties found on a connection.

Usage

```
get_connection_properties
```

Returns

A list of connection properties.

Arguments

No arguments.

Example

```
get_connection_properties
```

Related Information

- [get_connection_property](#) on page 5-84
- [get_connections](#) on page 5-85

get_connection_property

Description

Returns the value of a property found on a connection. Properties represent aspects of the connection that you can modify, such as the type of connection.

Usage

```
get_connection_property <connection> <property>
```

Returns

The value of a connection property.

Arguments

connection

The connection to query.

property

The name of the connection. property. Refer to *Connection Properties*.

Example

```
get_connection_property cpu.data_master/dma0.csr TYPE
```

Related Information

- [get_connection_properties](#) on page 5-83
- [Connection Properties](#) on page 5-142

get_connections

Description

Returns a list of all connections in the system if no element is specified. If you specify a child instance, for example `cpu`, Qsys returns all connections to any interface on the instance. If specify an interface on a child instance, for example `cpu.instruction_master`, Qsys returns all connections to that interface.

Usage

```
get_connections [<element>]
```

Returns

A list of connections.

Arguments

element (optional)

The name of a child instance, or the qualified name of an interface on a child instance.

Example

```
get_connections  
get_connections cpu  
get_connections cpu.instruction_master
```

Related Information

- [add_connection](#) on page 5-62
- [remove_connection](#) on page 5-121

get_instance_assignment

Description

Returns the value of an assignment on a child instance. Qsys uses assignments to transfer information about hardware to embedded software tools and applications.

Usage

```
get_instance_assignment <instance> <assignment>
```

Returns

The value of the specified assignment.

Arguments

instance

The name of the child instance.

assignment

The assignment key to query.

Example

```
get_instance_assignment video_0 embeddedsw.CMacro.colorSpace
```

Related Information

[get_instance_assignments](#) on page 5-87

get_instance_assignments

Description

Returns a list of assignment keys for any assignments defined for the instance.

Usage

```
get_instance_assignments <instance>
```

Returns

A list of assignment keys.

Arguments

instance

The name of the child instance.

Example

```
get_instance_assignments sdram
```

Related Information

[get_instance_assignment](#) on page 5-86

[get_instance_assignment](#) on page 5-86

get_instance_documentation_links

Description

Returns a list of all documentation links provided by an instance.

Usage

```
get_instance_documentation_links <instance>
```

Returns

A list of documentation links.

Arguments

instance

The name of the child instance.

Example

```
get_instance_documentation_links cpu_0
```

Notes

The list of documentation links includes titles and URLs for the links. For instance, a component with a single data sheet link may return:

```
{Data Sheet} {http://url/to/data/sheet}
```


get_instance_interface_assignment

Description

Returns the value of an assignment on an interface of a child instance. Qsys uses assignments to transfer information about hardware to embedded software tools and applications.

Usage

```
get_instance_interface_assignment <instance> <interface> <assignment>
```

Returns

The value of the specified assignment.

Arguments

instance

The name of the child instance.

interface

The name of an interface on the child instance.

assignment

The assignment key to query.

Example

```
get_instance_interface_assignment sdram s1 embeddedsw.configuration.isFlash
```

Related Information

[get_instance_interface_assignments](#) on page 5-90

get_instance_interface_assignments

Description

Returns a list of assignment keys for any assignments defined for an interface of a child instance.

Usage

```
get_instance_interface_assignments <instance> <interface>
```

Returns

A list of assignment keys.

Arguments

instance

The name of the child instance.

interface

The name of an interface on the child instance.

Example

```
get_instance_interface_assignments sdram s1
```

Related Information

[get_instance_interface_assignment](#) on page 5-89

get_instance_interface_parameter_property

Description

Returns the value of a property on a parameter in an interface of a child instance. Parameter properties are metadata about how Qsys uses the parameter.

Usage

```
get_instance_interface_parameter_property <instance> <interface> <parameter> <property>
```

Returns

The value of the parameter property.

Arguments

instance

The name of the child instance.

interface

The name of an interface on the child instance.

parameter

The name of the parameter on the interface.

property

The name of the property on the parameter. Refer to *Parameter Properties*.

Example

```
get_instance_interface_parameter_property uart_0 s0 setupTime ENABLED
```

Related Information

- [get_instance_interface_parameters](#) on page 5-93
- [get_instance_interfaces](#) on page 5-98
- [get_parameter_properties](#) on page 5-115
- [Parameter Properties](#) on page 5-150

get_instance_interface_parameter_value

Description

Returns the value of a parameter of an interface in a child instance.

Usage

```
get_instance_interface_parameter_value <instance> <interface> <parameter>
```

Returns

The value of the parameter.

Arguments

instance

The name of the child instance.

interface

The name of an interface on the child instance.

parameter

The name of the parameter on the interface.

Example

```
get_instance_interface_parameter_value uart_0 s0 setupTime
```

Related Information

- [get_instance_interface_parameters](#) on page 5-93
- [get_instance_interfaces](#) on page 5-98

get_instance_interface_parameters

Description

Returns a list of parameters for an interface in a child instance.

Usage

```
get_instance_interface_parameters <instance> <interface>
```

Returns

A list of parameter names for parameters in the interface.

Arguments

instance

The name of the child instance.

interface

The name of an interface on the uart_0 s0.

Example

```
get_instance_interface_parameters instance interface
```

Related Information

- [get_instance_interface_parameter_value](#) on page 5-92
- [get_instance_interfaces](#) on page 5-98

get_instance_interface_port_property

Description

Returns the value of a property of a port found in the interface of a child instance.

Usage

```
get_instance_interface_port_property <instance> <interface> <port> <property>
```

Returns

The value of the port property.

Arguments

instance

The name of the child instance.

interface

The name of an interface on the child instance.

port

The name of the port in the interface.

property

The name of the property of the port. Refer to *Port Properties*.

Example

```
get_instance_interface_port_property uart_0 exports tx WIDTH
```

Related Information

- [get_instance_interface_ports](#) on page 5-95
- [get_port_properties](#) on page 5-116
- [Port Properties](#) on page 5-155

get_instance_interface_ports

Description

Returns a list of ports found in an interface of a child instance.

Usage

```
get_instance_interface_ports <instance> <interface>
```

Returns

A list of port names found in the interface.

Arguments

instance

The name of the child instance.

interface

The name of an interface on the child instance.

Example

```
get_instance_interface_ports uart_0 s0
```

Related Information

- [get_instance_interface_port_property](#) on page 5-94
- [get_instance_interfaces](#) on page 5-98

get_instance_interface_properties

Description

Returns a list of properties that can be queried for an interface in a child instance.

Usage

```
get_instance_interface_properties
```

Returns

A list of property names.

Arguments

No arguments.

Example

```
get_instance_interface_properties
```

Related Information

- [get_instance_interface_property](#) on page 5-97
- [get_instance_interfaces](#) on page 5-98

get_instance_interface_property

Description

Returns the value of a property for an interface in a child instance.

Usage

```
get_instance_interface_property <instance> <interface> <property>
```

Returns

The value of the property.

Arguments

instance

The name of the child instance.

interface

The name of an interface on the child instance.

property

The name of the property of the interface. Refer to *Element Properties*.

Example

```
get_instance_interface_property uart_0 s0 DESCRIPTION
```

Related Information

- [get_instance_interface_properties](#) on page 5-96
- [get_instance_interfaces](#) on page 5-98
- [Element Properties](#) on page 5-145

get_instance_interfaces

Description

Returns a list of interfaces found in a child instance

Usage

```
get_instance_interfaces <instance>
```

Returns

A list of interface names.

Arguments

instance

The name of the child instance.

Example

```
get_instance_interfaces uart_0
```

Related Information

- [get_instance_interface_ports](#) on page 5-95
- [get_instance_interface_properties](#) on page 5-96
- [get_instance_interface_property](#) on page 5-97

get_instance_parameter_property

Description

Returns the value of a property on a parameter in a child instance. Parameter properties are metadata about how Qsys uses the parameter.

Usage

```
get_instance_parameter_property <instance> <parameter> <property>
```

Returns

The value of the parameter property.

Arguments

instance

The name of the child instance.

parameter

The name of the parameter in the instance.

property

The name of the property of the parameter. Refer to *Parameter Properties*.

Example

```
get_instance_parameter_property uart_0 baudRate ENABLED
```

Related Information

- [get_instance_parameters](#) on page 5-101
- [get_parameter_properties](#) on page 5-115
- [Parameter Properties](#) on page 5-150

get_instance_parameter_value

Description

Returns the value of a parameter in a child instance.

Usage

```
get_instance_parameter_value <instance> <parameter>
```

Returns

The value of the parameter.

Arguments

instance

The name of the child instance.

parameter

The name of the parameter in the instance.

Example

```
get_instance_parameter_value uart_0 baudRate
```

Related Information

- [get_instance_parameters](#) on page 5-101
- [set_instance_parameter_value](#) on page 5-128

get_instance_parameters

Description

Returns a list of parameters in a child instance.

Usage

```
get_instance_parameters <instance>
```

Returns

A list of parameters in the instance.

Arguments

instance

The name of the child instance.

Example

```
get_instance_parameters uart_0
```

Related Information

- [get_instance_parameter_property](#) on page 5-99
- [get_instance_interface_parameter_value](#) on page 5-92
- [set_instance_parameter_value](#) on page 5-128

get_instance_port_property

Description

Returns the value of a property of a port contained by an interface in a child instance.

Usage

```
get_instance_port_property <instance> <port> <property>
```

Returns

The value of the property for the port.

Arguments

instance

The name of the child instance.

port

The name of a port in one of the interfaces on the child instance.

property

The name of a property found on the port. Refer to *Port Properties*.

Example

```
get_instance_port_property uart_0 tx WIDTH
```

Related Information

- [get_instance_interface_ports](#) on page 5-95
- [get_port_properties](#) on page 5-116
- [Port Properties](#) on page 5-155

get_instance_properties

Description

Returns a list of properties for a child instance.

Usage

```
get_instance_properties
```

Returns

A list of property names for the child instance.

Arguments

No arguments.

Example

```
get_instance_properties
```

Related Information

[get_instance_property](#) on page 5-104

get_instance_property

Description

Returns the value of a property for a child instance.

Usage

```
get_instance_property <instance> <property>
```

Returns

The value of the property.

Arguments

instance

The name of the child instance.

property

The name of a property found on the instance. Refer to *Element Properties*.

Example

```
get_instance_property uart_0 ENABLED
```

Related Information

- [get_instance_properties](#) on page 5-103
- [Element Properties](#) on page 5-145

get_instances

Description

Returns a list of the instance names for all child instances in the system.

Usage

```
get_instances
```

Returns

A list of child instance names.

Arguments

No arguments.

Example

```
get_instances
```

Related Information

- [add_instance](#) on page 5-63
- [remove_instance](#) on page 5-123

get_interconnect_requirement

Description

Returns the value of an interconnect requirement for a system or interface on a child instance.

Usage

```
get_interconnect_requirement <element_id> <requirement>
```

Returns

The value of the interconnect requirement.

Arguments

element_id

{*\$system*} for the system, or the qualified name of the interface of an instance, in <*instance*>.<*interface*> format. In Tcl, the system identifier is escaped, for example, {*\$system*}.

requirement

The name of the requirement.

Example

```
get_interconnect_requirement {$system} qsys_mm.maxAdditionalLatency
```

get_interconnect_requirements

Description

Returns a list of all interconnect requirements in the system.

Usage

```
get_interconnect_requirements
```

Returns

A flattened list of interconnect requirements. Every sequence of three elements in the list corresponds to one interconnect requirement. The first element in the sequence is the element identifier. The second element is the requirement name. The third element is the value. You can loop over the returned list with a `foreach` loop, for example:

```
foreach { element_id name value } $requirement_list { loop_body
}
```

Arguments

No arguments.

Example

```
get_interconnect_requirements
```

get_interface_port_property

Description

Returns the value of a property of a port contained by one of the top-level exported interfaces

Usage

```
get_interface_port_property <interface> <port> <property>
```

Returns

The value of the property.

Arguments

interface

The name of a top-level interface on the system.

port

The name of a port found in the interface.

property

The name of a property found on the port. Refer to *Port Properties*.

Example

```
get_interface_port_property uart_exports tx DIRECTION
```

Related Information

- [get_interface_ports](#) on page 5-109
- [get_port_properties](#) on page 5-116
- [Port Properties](#) on page 5-155

get_interface_ports

Description

Returns the names of all of the ports that have been added to a given interface.

Usage

```
get_interface_ports <interface>
```

Returns

A list of port names.

Arguments

interface

The name of a top-level interface on the system.

Example

```
get_interface_ports export_clk_out
```

Related Information

- [get_interface_port_property](#) on page 5-108
- [get_interfaces](#) on page 5-112

`get_interface_properties`

Description

Returns the names of all the available interface properties common to all interface types.

Usage

```
get_interface_properties
```

Returns

A list of interface properties.

Arguments

No arguments.

Example

```
get_interface_properties
```

Related Information

- [get_interface_property](#) on page 5-111
- [set_interface_property](#) on page 5-131

get_interface_property

Description

Returns the value of a single interface property from the specified interface.

Usage

```
get_interface_property <interface> <property>
```

Returns

The property value.

Arguments

interface

The name of a top-level interface on the system.

property

The name of the property. Refer to *Interface Properties*.

Example

```
get_interface_property export_clk_out EXPORT_OF
```

Related Information

- [get_interface_properties](#) on page 5-110
- [set_interface_property](#) on page 5-131
- [Interface Properties](#) on page 5-147

`get_interfaces`

Description

Returns a list of top-level interfaces in the system.

Usage

```
get_interfaces
```

Returns

A list of the top-level interfaces exported from the system.

Arguments

No arguments.

Example

```
get_interfaces
```

Related Information

- [add_interface](#) on page 5-64
- [get_interface_ports](#) on page 5-109
- [get_interface_property](#) on page 5-111
- [remove_interface](#) on page 5-124
- [set_interface_property](#) on page 5-131

get_module_properties

Description

Returns the properties that you can manage for top-level module of the Qsys system.

Usage

```
get_module_properties
```

Returns

A list of property names.

Arguments

No arguments.

Example

```
get_module_properties
```

Related Information

- [get_module_property](#) on page 5-114
- [set_module_property](#) on page 5-132

`get_module_property`

Description

Returns the value of a top-level system property.

Usage

```
get_module_property <property>
```

Returns

The value of the property.

Arguments

property

The name of the property to query. Refer to *Module Properties*.

Example

```
get_module_property NAME
```

Related Information

- [get_module_properties](#) on page 5-113
- [set_module_property](#) on page 5-132

get_parameter_properties

Description

Returns a list of properties that you can query for any parameters, for example parameters on instances, interfaces, instance interfaces, and connections.

Usage

```
get_parameter_properties
```

Returns

A list of parameter properties.

Arguments

No arguments.

Example

```
get_parameter_properties
```

Related Information

- [get_connection_parameter_property](#) on page 5-80
- [get_instance_interface_parameter_property](#) on page 5-91
- [get_instance_parameter_property](#) on page 5-99

`get_port_properties`

Description

Returns a list of properties that you can query for ports.

Usage

```
get_port_properties
```

Returns

A list of port properties.

Arguments

No arguments.

Example

```
get_port_properties
```

Related Information

- [get_instance_interface_port_property](#) on page 5-94
- [get_instance_interface_ports](#) on page 5-95
- [get_instance_port_property](#) on page 5-102
- [get_interface_port_property](#) on page 5-108
- [get_interface_ports](#) on page 5-109

get_project_properties

Description

Returns a list of properties that you can query for properties pertaining to the Quartus II project.

Usage

```
get_project_properties
```

Returns

A list of project properties.

Arguments

No arguments

Example

```
get_project_properties
```

Related Information

- [get_project_property](#) on page 5-118
- [set_project_property](#) on page 5-133

`get_project_property`

Description

Returns the value of a Quartus II project property. Not all Quartus II project properties are available.

Usage

```
get_project_property <property>
```

Returns

The value of the property.

Arguments

property

The name of the project property. Refer to *Project properties*.

Example

```
get_project_property DEVICE_FAMILY
```

Related Information

- [get_module_properties](#) on page 5-113
- [get_module_property](#) on page 5-114
- [set_module_property](#) on page 5-132

load_system

Description

Loads a Qsys system from a file, and uses the system as the current system for scripting commands.

Usage

```
load_system <file>
```

Returns

No return value.

Arguments

file

The path to a .qsys file.

Example

```
load_system example.qsys
```

Related Information

- [create_system](#) on page 5-70
- [save_system](#) on page 5-125

lock_avalon_base_address

Description

Prevents the memory-mapped base address from being changed for connections to the specified interface on an instance when Qsys runs the `auto_assign_base_addresses` or `auto_assign_system_base_addresses` commands.

Usage

```
lock_avalon_base_address <instance.interface>
```

Returns

No return value.

Arguments

instance.interface

The qualified name of the interface of an instance, in `<instance>.<interface>` format.

Example

```
lock_avalon_base_address sdram.s1
```

Related Information

- [auto_assign_base_addresses](#) on page 5-66
- [auto_assign_system_base_addresses](#) on page 5-67
- [unlock_avalon_base_address](#) on page 5-136

remove_connection

Description

This command removes a connection from the system.

Usage

```
remove_connection <connection>
```

Returns

no return value

Arguments

connection

The name of the connection to remove

Example

```
remove_connection cpu.data_master/sdram.s0
```

Related Information

- [add_connection](#) on page 5-62
- [get_connections](#) on page 5-85

remove_dangling_connections

Description

Removes connections where both end points of the connection no longer exist in the system.

Usage

```
remove_dangling_connections
```

Returns

No return value.

Arguments

No arguments.

Example

```
remove_dangling_connections
```

remove_instance

Description

Removes a child instance from the system.

Usage

```
remove_instance <instance>
```

Returns

No return value.

Arguments

instance

The name of the child instance to remove.

Example

```
remove_instance cpu
```

Related Information

- [add_instance](#) on page 5-63
- [get_instances](#) on page 5-105

`remove_interface`

Description

Removes an exported top-level interface from the system.

Usage

```
remove_interface <interface>
```

Returns

No return value.

Arguments

interface

The name of the exported top-level interface.

Example

```
remove_interface clk_out
```

Related Information

- [add_interface](#) on page 5-64
- [get_interfaces](#) on page 5-112

save_system

Description

Saves the current system to the named file. If you do not specify the file, Qsys saves the system to the same file that was opened with the `load_system` command. You can specify the file as an absolute or relative path. Relative paths are relative to directory of the most recently loaded system, or relative to the working directory if no systems are loaded.

Usage

```
save_system <file>
```

Returns

No return value.

Arguments

file

If available, the path of the **.qsys** file to save.

Example

```
save_system
```

```
save_system file.qsys
```

send_message

Description

Sends a message to the user of the script. The message text is normally interpreted as HTML. You can use the `` element to provide emphasis.

Usage

```
send_message <level> <message>
```

Returns

No return value.

Arguments

level

The following message levels are supported:

- `ERROR`--Provides an error message.
- `WARNING`--Provides a warning message.
- `INFO`--Provides an informational message.
- `PROGRESS`--Provides a progress message.
- `DEBUG`--Provides a debug message when debug mode is enabled. Refer to *Message Levels Properties*.

message

The text of the message.

Example

```
send_message ERROR "The system is down!"
```

Related Information

[Message Levels Properties](#) on page 5-148

set_connection_parameter_value

Description

Sets the value of a parameter for a connection.

Usage

```
set_connection_parameter_value <connection> <parameter> <value>
```

Returns

No return value.

Arguments

connection

The name of the connection.

parameter

The name of the parameter.

value

The new parameter value.

Example

```
set_connection_parameter_value cpu.data_master/dma0.csr baseAddress "0x000a0000"
```

Related Information

- [get_connection_parameter_value](#) on page 5-81
- [get_connection_parameters](#) on page 5-82

set_instance_parameter_value

Description

Set the value of a parameter for a child instance. You cannot set derived parameters and `SYSTEM_INFO` parameters for the child instance with this command.

Usage

```
set_instance_parameter_value <instance> <parameter> <value>
```

Returns

No return value.

Arguments

instance

The name of the child instance.

parameter

The name of the parameter.

value

The new parameter value.

Example

```
set_instance_parameter_value uart_0 baudRate 9600
```

Related Information

- [get_instance_parameter_value](#) on page 5-100
- [get_instance_parameter_property](#) on page 5-99

set_instance_property

Description

Sets the value of a property of a child instance. Most instance properties are read-only and can only be set by the instance itself. The primary use for this command is to update the `ENABLED` parameter, which includes or excludes a child instance when generating Qsys interconnect.

Usage

```
set_instance_property <instance> <property> <value>
```

Returns

No return value.

Arguments

instance

The name of the child instance.

property

The name of the property. Refer to *Instance Properties*.

value

The new property value.

Example

```
set_instance_property cpu ENABLED false
```

Related Information

- [get_instance_parameters](#) on page 5-101
- [get_instance_property](#) on page 5-104
- [Instance Properties](#) on page 5-146

set_interconnect_requirement

Description

Sets the value of an interconnect requirement for a system or an interface on a child instance.

Usage

```
set_interconnect_requirement <element_id> <requirement> <value>
```

Returns

No return value.

Arguments

element_id

{*\$system*} for the system, or qualified name of the interface of an instance, in <*instance*>.<*interface*> format. In Tcl, the system identifier is escaped, for example, {*\$system*}.

requirement

The name of the requirement.

value

The new requirement value.

Example

```
set_interconnect_requirement {$system} qsys_mm.clockCrossingAdapter HANDSHAKE
```

set_interface_property

Description

Sets the value of a property on an exported top-level interface. You use this command to set the `EXPORT_OF` property to specify which interface of a child instance is exported via this top-level interface.

Usage

```
set_interface_property <interface> <property> <value>
```

Returns

No return value.

Arguments

interface

The name of an exported top-level interface.

property

The name of the property. Refer to *Interface Properties*.

value

The new property value.

Example

```
set_interface_property clk_out EXPORT_OF clk.clk_out
```

Related Information

- [add_interface](#) on page 5-64
- [get_interface_properties](#) on page 5-110
- [get_interface_property](#) on page 5-111
- [Interface Properties](#) on page 5-147

set_module_property

Description

Sets the value of a system property, such as the name of the system using the `NAME` property.

Usage

```
set_module_property <property> <value>
```

Returns

No return value.

Arguments

property

The name of the property. Refer to *Module Properties*.

value

The new property value.

Example

```
set_module_property NAME "new_system_name"
```

Related Information

- [get_module_properties](#) on page 5-113
- [get_module_property](#) on page 5-114
- [Module Properties](#) on page 5-149

set_project_property

Description

Sets the value of a project property, such as the device family.

Usage

```
set_project_property <property> <value>
```

Returns

No return value.

Arguments

property

The name of the property. Refer to *Project Properties*.

value

The new property value.

Example

```
set_project_property DEVICE_FAMILY "Cyclone IV GX"
```

Related Information

- [get_project_properties](#) on page 5-117
- [set_project_property](#) on page 5-133
- [Project Properties](#) on page 5-156
- [get_project_properties](#) on page 5-117
- [get_project_property](#) on page 5-118
- [Project Properties](#) on page 5-156

set_use_testbench_naming_pattern

Description

Use this command to create testbench systems so that the generated file names for the test system match the system's original generated file names. Without setting this, the generated file names for the test system receive the top-level testbench system name.

Usage

```
set_use_testbench_naming_pattern <value>
```

Returns

No return value.

Arguments

value

True or false.

Example

```
set_use_testbench_naming_pattern true
```

Notes

Use this command only to create testbench systems.

set_validation_property

Description

Sets a property that affects how and when validation is run. To disable system validation after each scripting command, set `AUTOMATIC_VALIDATION` to `False`.

Usage

```
set_validation_property <property> <value>
```

Returns

No return value.

Arguments

property

The name of the property. Refer to *Validation Properties*.

value

The new property value.

Example

```
set_validation_property AUTOMATIC_VALIDATION false
```

Related Information

- [validate_system](#) on page 5-140
- [Validation Properties](#) on page 5-160

unlock_avalon_base_address

Description

Allows the memory-mapped base address to change for connections to the specified interface on an instance when Qsys runs the `auto_assign_base_addresses` or `auto_assign_system_base_addresses` commands.

Usage

```
unlock_avalon_base_address <instance.interface>
```

Returns

No return value.

Arguments

instance.interface

The qualified name of the interface of an instance, in `<instance>.<interface>` format.

Example

```
unlock_avalon_base_address sdram.sl
```

Related Information

- [auto_assign_base_addresses](#) on page 5-66
- [auto_assign_system_base_addresses](#) on page 5-67
- [lock_avalon_base_address](#) on page 5-120

validate_connection

Description

Validates the specified connection and returns validation messages.

Usage

```
validate_connection <connection>
```

Returns

A list of messages produced during validation.

Arguments

connection

The name of the connection to validate.

Example

```
validate_connection cpu.data_master/sdram.sl
```

Related Information

- [validate_instance](#) on page 5-138
- [validate_instance_interface](#) on page 5-139
- [validate_system](#) on page 5-140

validate_instance

Description

Validates the specified child instance and returns validation messages.

Usage

```
validate_instance <instance>
```

Returns

A list of messages produced during validation.

Arguments

instance

The name of the child instance to validate.

Example

```
validate_instance cpu
```

Related Information

- [validate_connection](#) on page 5-137
- [validate_instance_interface](#) on page 5-139
- [validate_system](#) on page 5-140

validate_instance_interface

Description

Validates an interface on a child instance and returns validation messages.

Usage

```
validate_instance_interface <instance> <interface>
```

Returns

A list of messages produced during validation.

Arguments

instance

The name of a child instance.

interface

The name of the interface on the child instance to validate.

Example

```
validate_instance_interface cpu data_master
```

Related Information

- [validate_connection](#) on page 5-137
- [validate_instance](#) on page 5-138
- [validate_system](#) on page 5-140

`validate_system`

Description

Validates the system and returns validation messages.

Usage

```
validate_system
```

Returns

A list of validation messages produced during validation.

Arguments

No arguments.

Example

```
validate_system
```

Related Information

- [validate_connection](#) on page 5-137
- [validate_instance](#) on page 5-138
- [validate_instance_interface](#) on page 5-139

Qsys Scripting Property Reference

Interface properties work differently for `_hw.tcl` scripting than with qsys scripting. In `_hw.tcl`, interfaces do not distinguish between properties and parameters. In qsys scripting, properties and parameters are unique.

Connection Properties on page 5-142

Design Environment Type Properties on page 5-143

Direction Properties on page 5-144

Element Properties on page 5-145

Instance Properties on page 5-146

Interface Properties on page 5-147

Message Levels Properties on page 5-148

Module Properties on page 5-149

Parameter Properties on page 5-150

Parameter Status Properties on page 5-153

Parameter Type Properties on page 5-154

Port Properties on page 5-155

Project Properties on page 5-156

System Info Type Properties on page 5-157

Units Properties on page 5-159

Validation Properties on page 5-160

Connection Properties

Type	Name	Description
string	END	The end interface of the connection.
string	NAME	The name of the connection.
string	START	The start interface of the connection.
String	TYPE	The type of the connection.

Design Environment Type Properties

Description

IP cores use the design environment to identify what sort of interfaces are most appropriate to connect in the parent system.

Name	Description
NATIVE	The design environment supports native IP interfaces.
QSYS	The design environment supports standard Qsys interfaces.

Direction Properties

Name	Description
BIDIR	The direction for a bidirectional signal.
INOUT	The direction for an input signal.
OUTPUT	The direction for an output signal.

Element Properties

Description

Element properties are, with the exception of `ENABLED` and `NAME`, read-only properties of the types of instances, interfaces, and connections. These read-only properties represent metadata that does not vary between copies of the same type. `ENABLED` and `NAME` properties are specific to particular instances, interfaces, or connections.

Type	Name	Description
String	<code>AUTHOR</code>	The author of the component or interface.
Boolean	<code>AUTO_EXPORT</code>	Indicates whether unconnected interfaces on the instance are automatically exported.
String	<code>CLASS_NAME</code>	The type of the instance, interface or connection, for example, <code>altera_nios2</code> or <code>avalon_slave</code> .
String	<code>DESCRIPTION</code>	The description of the instance, interface or connection type.
String	<code>DISPLAY_NAME</code>	The display name for referencing the type of instance, interface or connection.
Boolean	<code>EDITABLE</code>	Indicates whether you can edit the component in the Qsys Component Editor.
Boolean	<code>ENABLED</code>	Indicates whether the instance is turned on.
String	<code>GROUP</code>	The IP Catalog category.
Boolean	<code>INTERNAL</code>	Hides internal IP components or sub-components from the IP Catalog..
String	<code>NAME</code>	The name of the instance, interface or connection.
String	<code>VERSION</code>	The version number of the instance, interface or connection, for example, <code>14.0</code> .

Instance Properties

Type	Name	Description
String	AUTO_EXPORT	Indicates whether unconnected interfaces on the instance are automatically exported.
Boolean	ENABLED	If true, this instance is included in the generated system. if false, it is not included.
String	NAME	The name of the system, which is used as the name of the top-level module in the generated HDL.

Interface Properties

Type	Name	Description
String	EXPORT_OF	<p>Indicates which interface of a child instance to export through the top-level interface. Before using this command, you must create the top-level interface using the <code>add_interface</code> command. You must use the format: <code><instanceName.interfaceName></code>. For example:</p> <pre>set_interface_property CSC_input EXPORT_OF my_colorSpaceCon- verter.input_port</pre>

Message Levels Properties

Name	Description
COMPONENT_INFO	Reports an informational message only during component editing.
DEBUG	Provides messages when debug mode is turned on.
ERROR	Provides an error message.
INFO	Provides an informational message.
PROGRESS	Reports progress during generation.
TODOERROR	Provides an error message that indicates the system is incomplete.
WARNING	Provides a warning message.

Module Properties

Type	Name	Description
String	GENERATION_ID	The generation ID for the system.
String	NAME	The name of the instance.

Parameter Properties

Type	Name	Description
Boolean	AFFECTS_ELABORATION	Set AFFECTS_ELABORATION to false for parameters that do not affect the external interface of the module. An example of a parameter that does not affect the external interface is <code>isNonVolatileStorage</code> . An example of a parameter that does affect the external interface is <code>width</code> . When the value of a parameter changes and AFFECTS_ELABORATION is false, the elaboration phase does not repeat and improves performance. When AFFECTS_ELABORATION is set to true, the default value, Qsys re-analyzes the HDL file to determine the port widths and configuration each time a parameter changes.
Boolean	AFFECTS_GENERATION	The default value of AFFECTS_GENERATION is false if you provide a top-level HDL module. The default value is true if you provide a fileset callback. Set AFFECTS_GENERATION to false if the value of a parameter does not change the results of fileset generation.
Boolean	AFFECTS_VALIDATION	The AFFECTS_VALIDATION property determines whether a parameter's value sets derived parameters, and whether the value affects validation messages. When set to false, this may improve response time in the parameter editor when the value changes.
String[]	ALLOWED_RANGES	Indicates the range or ranges of the parameter. For integers, Each range is a single value, or a range of values defined by a start and end value, and delimited by a colon, for example, <code>11:15</code> . This property also specifies the legal values and description strings for integers, for example, <code>{0:None 1:Monophonic 2:Stereo 4:Quadrophonic}</code> , where 0, 1, 2, and 4 are the legal values. You can assign description strings in the parameter editor for string variables. For example, <pre>ALLOWED_RANGES { "dev1:Cyclone IV GX" "dev2:Stratix V GT" }</pre>
String	DEFAULT_VALUE	The default value.
Boolean	DERIVED	When <code>True</code> , indicates that the parameter value is set by the component and cannot be set by the user. Derived parameters are not saved as part of an instance's parameter values. The default value is <code>False</code> .
String	DESCRIPTION	A short user-visible description of the parameter, suitable for a tooltip description in the parameter editor.
String[]	DISPLAY_HINT	Provides a hint about how to display a property. <ul style="list-style-type: none"> <code>boolean</code>--For integer parameters whose value are 0 or 1. The parameter displays as an option that you can turn on or off. <code>radio</code>--Displays a parameter with a list of values as radio buttons. <code>hexadecimal</code>--For integer parameters, displays and interprets the value as a hexadecimal number, for example: <code>0x00000010</code> instead of 16.

Type	Name	Description
		<ul style="list-style-type: none"> <code>fixed_size</code>--For <code>string_list</code> and <code>integer_list</code> parameters, the <code>fixed_size</code> <code>DISPLAY_HINT</code> eliminates the Add and Remove buttons from tables.
String	<code>DISPLAY_NAME</code>	The GUI label that appears to the left of this parameter.
String	<code>DISPLAY_UNITS</code>	The GUI label that appears to the right of the parameter.
Boolean	<code>ENABLED</code>	When <code>False</code> , the parameter is turned off. It displays in the parameter editor but is greyed out, indicating that you cannot edit this parameter.
String	<code>GROUP</code>	Controls the layout of parameters in the GUI.
Boolean	<code>HDL_PARAMETER</code>	When <code>True</code> , Qsys passes the parameter to the HDL component description. The default value is <code>False</code> .
String	<code>LONG_DESCRIPTION</code>	A user-visible description of the parameter. Similar to <code>DESCRIPTION</code> , but allows a more detailed explanation.
String	<code>NEW_INSTANCE_VALUE</code>	Changes the default value of a parameter without affecting older components that do not explicitly set a parameter value, and use the <code>DEFAULT_VALUE</code> property. Older instances continue to use <code>DEFAULT_VALUE</code> for the parameter and new instances use the value assigned by <code>NEW_INSTANCE_VALUE</code> .
String[]	<code>SYSTEM_INFO</code>	<p>Allows you to assign information about the instantiating system to a parameter that you define. <code>SYSTEM_INFO</code> requires an argument specifying the type of information for example,</p> <pre>SYSTEM_INFO <info-type></pre>
String	<code>SYSTEM_INFO_ARG</code>	Defines an argument to pass to <code>SYSTEM_INFO</code> . For example, the name of a reset interface.
(various)	<code>SYSTEM_INFO_TYPE</code>	Specifies the types of system information that you can query. Refer to <i>System Info Type Properties</i> .
(various)	<code>TYPE</code>	Specifies the type of the parameter. Refer to <i>Parameter Type Properties</i> .
(various)	<code>UNITS</code>	Sets the units of the parameter. Refer to <i>Units Properties</i> .
Boolean	<code>VISIBLE</code>	Indicates whether or not to display the parameter in the parameter editor.
String	<code>WIDTH</code>	Indicates the width of the logic vector for the <code>STD_LOGIC_VECTOR</code> parameter.

Related Information

- [System Info Type Properties](#) on page 5-157
- [Parameter Type Properties](#) on page 5-154

- [Units Properties](#) on page 5-159

Parameter Status Properties

Type	Name	Description
Boolean	ACTIVE	Indicates that this parameter is an active parameter.
Boolean	DEPRECATED	Indicates that this parameter exists only for backwards compatibility, and may not have any effect.
Boolean	EXPERIMENTAL	Indicates that this parameter is experimental and not exposed in the design flow.

Parameter Type Properties

Name	Description
BOOLEAN	A boolean parameter set to <code>true</code> or <code>false</code> .
FLOAT	A signed 32-bit floating point parameter. (Not supported for HDL parameters.)
INTEGER	A signed 32-bit integer parameter.
INTEGER_LIST	A parameter that contains a list of 32-bit integers. (Not supported for HDL parameters.)
LONG	A signed 64-bit integer parameter. (Not supported for HDL parameters.)
NATURAL	A 32-bit number that contains values 0 to 2147483647 (0x7fffffff).
POSITIVE	A 32-bit number that contains values 1 to 2147483647 (0x7fffffff).
STD_LOGIC	A single bit parameter set to 0 or 1.
STD_LOGIC_VECTOR	An arbitrary-width number. The parameter property <code>WIDTH</code> determines the size of the logic vector.
STRING	A string parameter.
STRING_LIST	A parameter that contains a list of strings. (Not supported for HDL parameters.)

Port Properties

Type	Name	Description
(various)	DIRECTION	The direction of the signal. Refer to <i>Direction Properties</i> .
String	ROLE	The type of the signal. Each interface type defines a set of interface types for its ports.
Integer	WIDTH	The width of the signal in bits.

Project Properties

Type	Name	Description
String	DEVICE	The device part number in the Quartus II project that contains the Qsys system.
String	DEVICE_FAMILY	The device family name in the Quartus II project that contains the Qsys system.

System Info Type Properties

Type	Name	Description
String	ADDRESS_MAP	An XML-formatted string that describes the address map for the interface specified in the <code>SYSTEM_INFO</code> parameter property.
Integer	ADDRESS_WIDTH	The number of address bits that Qsys requires to address memory-mapped slaves connected to the specified memory-mapped master in this instance.
String	AVALON_SPEC	The version of the Qsys interconnect. Refer to <i>Avalon Interface Specifications</i> .
Integer	CLOCK_DOMAIN	An integer that represents the clock domain for the interface specified in the <code>SYSTEM_INFO</code> parameter property. If this instance has interfaces on multiple clock domains, you can use this property to determine which interfaces are on each clock domain. The absolute value of the integer is arbitrary.
Long, Integer	CLOCK_RATE	The rate of the clock connected to the clock input specified in the <code>SYSTEM_INFO</code> parameter property. If zero, the clock rate is currently unknown.
String	CLOCK_RESET_INFO	The name of this instance's primary clock or reset sink interface. You use this property to determine the reset sink for global reset when you use SOPC Builder interconnect that conforms to <i>Avalon Interface Specifications</i> .
String	CUSTOM_INSTRUCTION_SLAVES	Provides slave information, including the name, base address, address span, and clock cycle type.
String	DESIGN_ENVIRONMENT	A string that identifies the current design environment. Refer to <i>Design Environment Type Properties</i> .
String	DEVICE	The device part number of the selected device.
String	DEVICE_FAMILY	The family name of the selected device.
String	DEVICE_FEATURES	A list of key/value pairs delimited by spaces that indicate whether a device feature is available in the selected device family. The format of the list is suitable for passing to the <code>array</code> command. The keys are device features. The values are 1 if the feature is present, and 0 if the feature is absent.
String	DEVICE_SPEEDGRADE	The speed grade of the selected device.
Integer	GENERATION_ID	A integer that stores a hash of the generation time that Qsys uses as a unique ID for a generation run.
BigInteger, Long	INTERRUPTS_USED	A mask indicating which bits of an interrupt receiver are connected to interrupt senders. The interrupt receiver is specified in the system info argument.

Type	Name	Description
Integer	MAX_SLAVE_DATA_WIDTH	The data width of the widest slave connected to the specified memory-mapped master.
String, Boolean, Integer	QUARTUS_INI	The value of the quartus.ini setting specified in the system info argument.
Integer	RESET_DOMAIN	An integer representing the reset domain for the interface specified in the <code>SYSTEM_INFO</code> parameter property. If this instance has interfaces on multiple reset domains, you can use this property to determine which interfaces are on each reset domain. The absolute value of the integer is arbitrary.
String	TRISTATECONDUIT_INFO	An XML description of the tri-state conduit masters connected to a tri-state conduit slave. The slave is specified as the <code>SYSTEM_INFO</code> parameter property. The value contains information about the slave, connected master instance and interface names, and signal names, directions, and widths.
String	TRISTATECONDUIT_MASTERS	The names of the instance's interfaces that are tri-state conduit slaves.
String	UNIQUE_ID	A string guaranteed to be unique to this instance.

Related Information

- [Design Environment Type Properties](#) on page 5-143
- [Avalon Interface Specifications](#)
- [Qsys Interconnect](#)

Units Properties

Name	Description
ADDRESS	A memory-mapped address.
BITS	Memory size in bits.
BITSPERSECOND	Rate in bits per second.
BYTES	Memory size in bytes.
CYCLES	A latency or count in clock cycles.
GIGABITSPERSECOND	Rate in gigabits per second.
GIGABYTES	Memory size in gigabytes.
GIGAHERTZ	Frequency in GHz.
HERTZ	Frequency in Hz.
KILOBITSPERSECOND	Rate in kilobits per second.
KILOBYTES	Memory size in kilobytes.
KILOHERTZ	Frequency in kHz.
MEGABITSPERSECOND	Rate, in megabits per second.
MEGABYTES	Memory size in megabytes.
MEGAHERTZ	Frequency in MHz.
MICROSECONDS	Time in microseconds.
MILLISECONDS	Time in milliseconds.
NANOSECONDS	Time in nanoseconds.
NONE	Unspecified units.
PERCENT	A percentage.
PICOSECONDS	Time in picoseconds.
SECONDS	Time in seconds.

Validation Properties

Type	Name	Description
Boolean	<code>AUTOMATIC_VALIDATION</code>	When <code>true</code> , Qsys runs system validation and elaboration after each scripting command. When <code>false</code> , Qsys runs system validation with validation scripting commands. Some queries affected by system elaboration may be incorrect if automatic validation is turned off. You can disable validation to make a system script run faster.

Document Revision History

The table below indicates edits made to the *Creating a System With Qsys* content since its creation.

Table 5-17: Document Revision History

Date	Version	Changes
August 2014	14.0a10.0	<ul style="list-style-type: none">Added distinction between legacy and standard device generation.Updated: <i>Upgrading Outdated IP Components</i>.Updated: <i>Generating a Qsys System</i>.Updated: <i>Integrating a Qsys System with the Quartus II Software</i>.Added screen shot: <i>Displaying Your Qsys System</i>.
June 2014	14.0.0	<ul style="list-style-type: none">Added tab descriptions: Details, Connections.Added <i>Managing IP Settings in the Quartus II Software</i>.Added <i>Upgrading Outdated IP Components</i>.Added <i>Support for Avalon-MM Non-Power of Two Data Widths</i>.
November 2013	13.1.0	<ul style="list-style-type: none">Added <i>Integrating with the .qsys File</i>.Added <i>Using the Hierarchy Tab</i>.Added <i>Managing Interconnect Requirements</i>.Added <i>Viewing Qsys Interconnect</i>.
May 2013	13.0.0	<ul style="list-style-type: none">Added AMBA APB support.Added qsys-generate utility.Added VHDL BFM ID support.Added <i>Creating Secure Systems (TrustZones)</i>.Added <i>CMSIS Support for Qsys Systems With An HPS Component</i>.Added VHDL language support options.
November 2012	12.1.0	<ul style="list-style-type: none">Added AMBA AXI4 support.
June 2012	12.0.0	<ul style="list-style-type: none">Added AMBA AX3I support.Added Preset Editor updates.Added command-line utilities, and scripts.
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 <i>Using Instance Parameters and Example Hierarchical System Using Parameters</i>.

Date	Version	Changes
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.

Related Information[Quartus II Handbook Archive](#)