

You define Qsys components in the component editor by declaring their properties and behaviors or directly in a Hardware Component Description File (`_hw.tcl`). Each `_hw.tcl` file represents one component which you can add to a Qsys system. You can then share these components with other designers. For your component to have maximum flexibility, you should consider that aspects of its behavior that can be parameterized so that other users can change the default parameterization to address different design requirements.

A Qsys component is usually composed of the following four types of files:

- `_hw.tcl` file—describes the Qsys related characteristics, such as interface behaviors. This file is required.
- HDL files—define the component's functionality as hardware, simulation, and constraint files. These files are optional.
- `_sw.tcl`—used by the software build tools to compile the component driver code. This file is optional.
- Component driver files—define the component register map and driver software to allow software to control the component. These files are optional.

This chapter discusses the following topics:

- “Information in a Hardware Component Description File”
- “Defining Components” on page 9-2
- “Writing a Hardware Component Description File” on page 9-3
- “Overriding Default Behaviors for Components Implemented in HDL” on page 9-10
- “Hardware Tcl Command Reference” on page 9-14



An excellent source of information about Tcl syntax is the [Tcl Developer Xchange](#) website.

Information in a Hardware Component Description File

A typical `_hw.tcl` file contains the following information:

- Basic component information—includes the component's name, version, and description, a link to its documentation, and pointers to HDL implementation files for synthesis and simulation.

- **Parameter Declarations**—Parameters are values that the user of your component can set which affect how the component is implemented, such as the size of a memory. Properties of each parameter include the parameter's name, whether or not it is visible, and, if visible, the text to display when describing it. When a Qsys system is generated, values of parameters declared with the HDL_PARAMETER property are applied to the HDL instantiation of the component as Verilog parameters or VHDL generics.
- **Interface Signals and Properties**—The interfaces of a component define how to connect it to the rest of the system and determine how other components in the system interact with it. When you add interfaces to a component, you declare which signals make up each interface. You also define interface properties, such as wait states for an Avalon® Memory-Mapped (Avalon-MM) interface.

Defining Components

You can use the Qsys system integration tool to implement two different kinds of components:

- **HDL components**—Components defined with HDL files to describe functionality and an `_hw.tcl` file to identify components to Qsys and downstream tools.
- **Composed components**—Components constructed by combining and connecting other components, taking advantage of hierarchical design facility available in Qsys. Composed components are defined by Tcl commands included in a `_hw.tcl` file. Composed components do not have their own HDL files; functionality is defined by the `_hw.tcl` and HDL files of the components that are instantiated.

The following sections describe Qsys component development for the both kinds of components.

HDL Component Implementation

Defining a Qsys component which is implemented in HDL includes the following four distinct phases:

- **Main Program**—Qsys first discovers a component and adds it to the component library. The `_hw.tcl` file is executed and the Tcl statements provide static (non-parameterized) information to Qsys. During this phase, component interfaces may be only partially defined.
- **Editor**—After an instance of your component has been added to a Qsys system, this phase allows the user of your component to use GUI to edit its parameters.
- **Elaboration**—Elaboration occurs when Qsys queries a component for its interface information. Interfaces defined in the main program can be enabled or disabled during elaboration. Elaboration is triggered when an instance of a component is created, when its parameters are changed, or when a property of the system is changed. As part of elaboration, a component can generate error, warning, or informational messages. Elaboration always occurs before generation. Once elaboration is complete, the component must be completely defined. For example, all port widths must have accurate values.

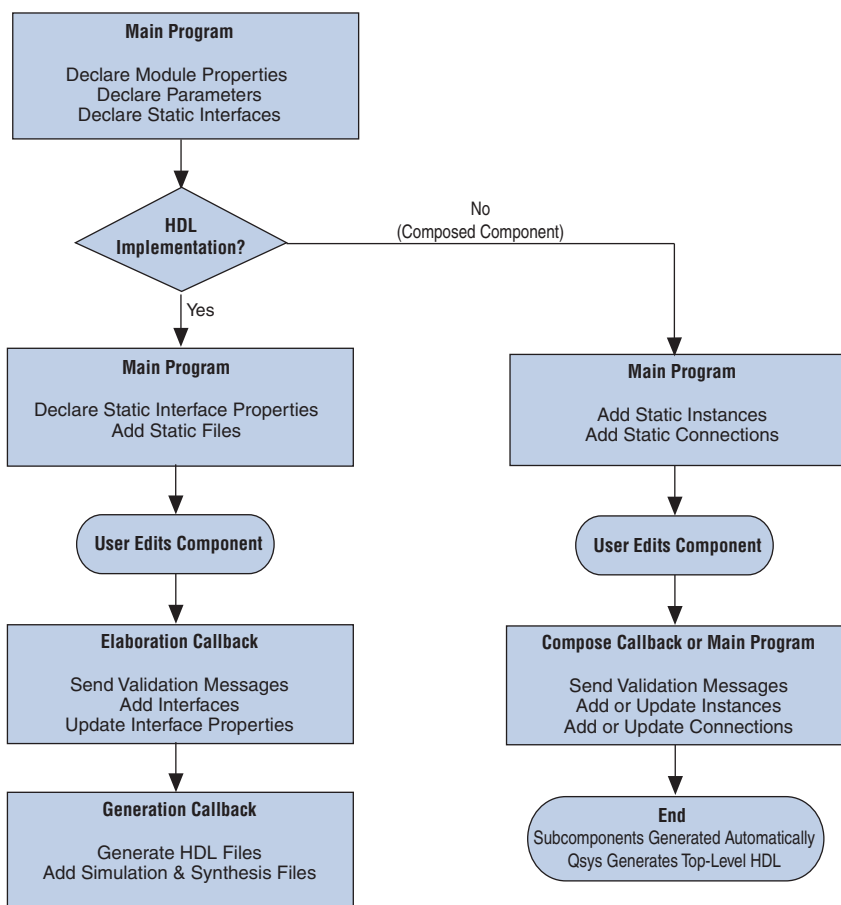
- **Generation**—Generation creates all the information that the Quartus® II software and HDL simulators require. The required files typically include VHDL or Verilog HDL files, simulation models, and timing constraints.

Composed Component Implementation

Because composed components are implemented by combining other components, composed components do not require the elaboration or generation phases that the HDL design flow requires. You can use Tcl commands to define composed components in the main program or in a separate composition callback.

Figure 9-1 illustrates the steps to create HDL and composed components.

Figure 9-1. Steps to Implement HDL and Composed Components



Writing a Hardware Component Description File

This section provides detailed information about `_hw.tcl` files and describes the default behavior of a component in all phases. The example provided uses a simple UART with some simple parameterization.

Providing Basic Information

A typical `_hw.tcl` file first declares basic information such as the name, location, and the files it includes. The first command in a `_hw.tcl` file should specify the version of the `_hw.tcl` API to use, with the following Tcl command:

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

The version number is a Quartus II release version, such as 11.1. Qsys guarantees that a valid `_hw.tcl` file that requests a particular qsys package behaves identically in future versions of the tool. Because of differences between versions of the Quartus II software, you cannot assume that a component's HDL files will work seamlessly in future versions of Quartus II.



This chapter describes the behavior of components that request the Qsys 11.1 package.



An excellent source of information about Tcl syntax is the [Tcl Developer Xchange](#) website.

Example 9-1. Basic Information for `_hw.tcl` File

```
# The package command must be the first command in the file
package require -exact qsys 11.1

# The name and VERSION of the component
set_module_property NAME example_uart
set_module_property VERSION 1.0

# The name of the component to display in the library
set_module_property DISPLAY_NAME "Example Component"

# The component's description.
set_module_property DESCRIPTION "An Example Component"

# The component library group that component belongs to
set_module_property GROUP Examples
```

Declaring Parameters

By including configuration parameters in your `_hw.tcl` file, you allow users of your component to parameterize it in different ways. Each parameter has a number of properties such as its name, type, display name, and default value that can be used to control how the parameter is displayed and used. [Example 9-2](#) illustrates the use of parameters that can be configured by users of your component.

Example 9-2. Declaring Parameters

```
# Declare Baud Rate parameter as an integer with a default value of 9600.
add_parameter BAUD_RATE int 9600

# Display this parameter as "Baud Rate" in the Parameter Editor.
set_parameter_property BAUD_RATE DISPLAY_NAME "Baud Rate (bps)"

# We only support three baud rates
set_parameter_property BAUD_RATE ALLOWED_RANGES {9600 19200 38400}
```

Parameters can be divided into three types: user parameters, system information parameters, and derived parameters. The following sections describe these parameter types.

User Parameters

User parameters are parameters that users have control over and that are displayed in the component parameter editor.

Derived Parameters

Derived parameters are parameters that are inferred by the component itself from user parameters or other derived parameters. For example, a clock period parameter can be derived from a data rate parameter. You can use derived parameters to perform operations that cannot be performed in HDL. For example, determining the number of address bits that a component requires using logarithmic functions is simple in Tcl and impossible in HDL. Derived parameters are setable in the `_hw.tcl` Elaboration callback and can have their values passed to the HDL component.

SYSTEM_INFO Parameters

You can use `SYSTEM_INFO` parameter to request that certain parameter values are populated with information about the system. For example, you might want to know the frequency of the clock that is connected to your clock input. When you define a `SYSTEM_INFO` parameter, you provide an `<info-type>` and further arguments. The `<info-type>` is the type of information you want, such as `clock_rate`, and you use the additional arguments to specify things, such as which clock input interface you require. [Example 9-3](#) illustrates the use of the `SYSTEM_INFO` parameter. For more information about the `SYSTEM_INFO` parameter properties refer to [Table 9-5 on page 9-27](#).

Example 9-3. Syntax of Tcl Command using the SYSTEM_INFO Parameter

```
set_parameter_property my_parameter SYSTEM_INFO_TYPE <info-type>  
set_parameter_property my_parameter SYSTEM_INFO_ARG <arg>
```

Declaring Interfaces

To declare an interface, use the `add_interface` command. Then use the `set_interface_property` and `add_interface_port` commands to set its properties and indicate which signals belong to it. The interface declaration statement includes the name of the interface, and the interface direction, and the clock and reset interfaces with which it is associated. [Example 9-4](#) illustrates interface declaration.

Example 9-4. Declare Interfaces

```
# Declare the clock sink interface, "clock_sink", type=clock, direction=sink
# Components using both the HDL and composition design flows add clocks and resets
add_interface clock_sink clock sink

# Declare the reset sink interface "reset_sink", type=reset direction=sink,
associatedClock=clock_sink
add_interface reset_set reset sink
set_interface_property reset_set associatedClock clock_sink

# Declare the Avalon slave interface, name=avalon_slave_0, type=avalon, directon=end
# Both HDL and composed components declare their top-level interfaces
add_interface avalon_slave_0 avalon end

#The commands below are for only for components created with HDL design flow, not the
composed component design flow

# The clock interface has one signal, named "clk" type "clk"
add_interface_port clock_sink clk clk input 1
set_interface_property reset_sync associatedClock clock_sink

# The reset interface has one signal, named "reset_n" type "reset_n"
add_interface_port reset_sink reset_n reset_n input 1

# Set a number of properties about the Avalon Slave interface
set_interface_property avalon_slave_0 writeWaitTime 0
set_interface_property avalon_slave_0 addressAlignment DYNAMIC
set_interface_property avalon_slave_0 readWaitTime 1
set_interface_property avalon_slave_0 readLatency 0

# Associate clock and reset interfaces with the Avalon slave
set_interface_property avalon_slave_0 associatedClock clock_sink
set_interface_property avalon_slave_0 associatedReset reset_sink

# Declare all the signals that belong to my Avalon Slave interface
add_interface_port avalon_slave_0 my_readdata readdata output 8
add_interface_port avalon_slave_0 my_read read input 1
add_interface_port avalon_slave_0 my_write write input 1
add_interface_port avalon_slave_0 my_waitrequest waitrequest output 1
add_interface_port avalon_slave_0 my_address address input 24
add_interface_port avalon_slave_0 my_writedata writedata input 8
```

Adding Files and Guiding Generation

Qsys allows a component author to describe exactly what files are needed in order to simulate a component in VHDL or Verilog, along with providing a list of files which are needed for Quartus II Analysis & Synthesis. Qsys defines the notion of filesets, where each fileset is targeted to a specific task. A component can have more than one callback which defines a given fileset. Currently, the type of filesets that can be defined are: `SIM_VERILOG`, `SIM_VHDL`, `QUARTUS_SYNTH`, and

EXAMPLE_DESIGN. Filesets can be added to a component with the command `add_fileset`. Refer to “[add_fileset](#)” on page 9-48 for the syntax of the `add_fileset` command. Adding files within a fileset is accomplished through the command `add_fileset_file`. A file type must be declared for each file, a list of available types can be found at “[add_fileset_file](#)” on page 9-49.

Every fileset is implemented through a user defined procedure specified by the `add_fileset` command. This procedure requires a single argument which will be populated with the expected module/entity name of the top level component. In the case where the HDL for a component is static and not generated based off the current parameterization, use the command `set_fileset_property <name of fileset> TOP_LEVEL <name of static top level module/entity>` in the main section of the component to force the fileset’s module name argument to represent the static module/entity name.



The `add_fileset_file` command can be used to rename files and create directories by specifying the output path as the 1st command argument.

[Example 9-5](#) shows an example of a Quartus synthesis fileset.

Default Behaviors

Example 9-5. Create Quartus Synthesis Fileset

```
# add quartus_synth fileset
add_fileset my_synthesis_fileset QUARTUS_SYNTH_synth_callback_procedure

# set top level name for static entity
set_fileset_property my_synthesis_fileset TOP_LEVEL simple_uart

# implement fileset callback
proc synth_callback_procedure { entityname } {
    # This check should never fail and is not required. It is shown for clarity of entityname
    contents.
    if { [string compare ${entityname} "simple_uart" ] != 0 } {
        send_message error " unexpected entity name; required simple_uart, received
        $entityname"
    }
}

#add the component hdl definition
add_fileset_file "./simple_uart.v" VERILOG PATH" ./simple_uart.v"

# add the timequest file with quartus timing constraints
add_fileset_file "./quartus_files/simple_uart.sdc" SDC PATH "./simple_uart.sdc"
}
```

The `_hw.tcl` file described in the previous section has default behaviors during the elaboration and generation phases. These default behaviors apply to instances of a component. This section describes the default Qsys behaviors for each of these phases. To override these default behaviors, refer to “[Overriding Default Behaviors for Components Implemented in HDL](#)” on page 9-10.

Elaboration Phase Behavior

The default Qsys elaboration phase checks each parameter value against its `ALLOWED_RANGES` property. If the values specified are outside the allowed ranges, an error message is displayed.

The `ALLOWED_RANGES` property of each parameter contains a list of acceptable values that a parameter can take on. `ALLOWED_RANGES` can be set to a single value, or a range of values defined by a start and end value separated by a colon. [Table 9-1](#) shows some examples of values the `ALLOWED_RANGES` property can take.

Table 9-1. ALLOWED_RANGES Property

ALLOWED_RANGES	Meaning
{a b c}	a or b or c
{1 2 4 8 16}	1, 2, 4, 8, or 16.
1:3	1 through 3, inclusive
{1 2 3 7:10}	1, 2, 3, or 7 through 10 inclusive
{"1:64-bit" "2:32-bit" "3: 16-bit"}	Display the choices Tom, Choice 2 and Bob in a drop-down menu to the user. Store the value 1, 2, or 3 as an integer value based off the user's selection.

Port Widths

You can set port widths to values or simple HDL expressions using the `width_expr` property. `width_expr` is a string that holds an expression or value describing the port width. By using the `width_expr` property, you can define port widths as an expression that is evaluated without needing to set them in the elaboration callback. The syntax for width expressions is the same as the HDL language that you use; however, only the addition, subtraction, multiplication, and division operators are allowed. For more complex port widths, the width of the port can be set as an arbitrary function of the component's parameters. The width expression is the last argument to the `add_interface_port` command. [Example 9-6](#) illustrates two uses of `width_expr`: one using mathematical operators and parameters, the other directly setting the value.

Example 9-6. Defining Port Widths Using Simple Mathematical Operators

```
add_interface_port din din_data data input {WIDTH * SYMBOLS}
set_port_property dout_data width_expr 32
```

Parameterized Parameter Widths

For VHDL users, Qsys allows a `std_logic_vector` parameter to have a width that is defined by another parameter. When adding a parameter of type `std_logic_vector` you can also specify its width as a parameter property. The width can be a constant or the name of another parameter. The commands [Example 9-7](#) add a `std_logic_vector` parameter called `myParameter` whose width is set by another parameter, called `dataWidth`.

Example 9-7. Adding Parameters

```
add_parameter myParameter STD_LOGIC_VECTOR
set_parameter_property myParameter WIDTH_EXPR dataWidth
```

Generation Phase Behavior

During the generation phase, Qsys creates a Verilog HDL or VHDL wrapper module to instantiate the top-level module and applies the parameters as selected by the user of your component. Qsys does not apply parameters in the wrapper if they are not declared in the underlying HDL file.

Edit Phase Behavior

The default Qsys editor phase behavior is to use all of the parameter definitions to display the parameter editor. The properties of the parameters guide Qsys when it builds the default parameter editor. [Table 9-4 on page 9-24](#) lists the properties of parameters.

You can place parameters in logical groups and provide images and text to create a custom parameter editor for your component. [Example 9-8](#) defines four parameters and illustrates the use of the `add_display_item` command and the `DISPLAY_HINT` and `ALLOWED_RANGES` parameters.

Example 9-8. Defining and Customizing the parameter editor

```
# provide an icon for the sound group
add_display_item icon Speaker speaker-image speaker.png
add_parameter sound string 0 0
add_parameter volume_control boolean 0 0
add_parameter separate_control string 0 0

# Setup DISPLAY_NAMES for the parameters
set_parameter_property sound DISPLAY_NAME Audio
set_parameter_property volume_control DISPLAY_NAME "Include Volume Control Interface"
set_parameter_property separate_control DISPLAY_NAME "Treble/Bass Controls"

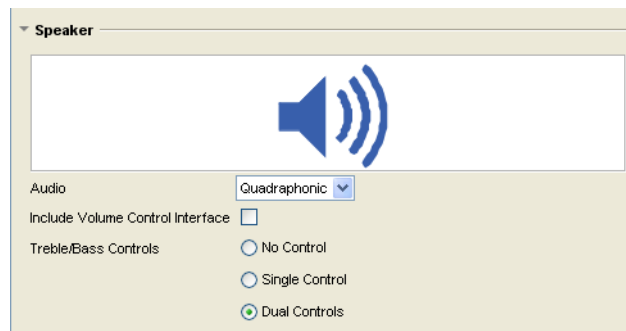
# Display all parameters in the Speaker group
add_display_item Speaker sound parameter
add_display_item Speaker volume_control parameter
add_display_item Speaker separate_control parameter

# There are 4 choices for the sound parameter.
# Strings with internal spaces require double quotes
set_parameter_property sound ALLOWED_RANGES {"0:No Audio" 1:Monophonic 2:Stereo
4:Quadraphonic}
set_parameter_property separate_control ALLOWED_RANGES {"No Control" "Single Control"
"Dual Controls"}
```

```
#Specify how parameters should be displayed
set_parameter_property volume_control DISPLAY_HINT boolean
set_parameter_property separate_control DISPLAY_HINT radio
```

Figure 9-2 shows the parameter editor that the Tcl commands in Example 9-8 produce.

Figure 9-2. Parameter Editor for Audio Component



Overriding Default Behaviors for Components Implemented in HDL

You can override each of the default behaviors by using callbacks. This section explains how to write callback procedures for each phase of component development.

Elaboration Callback

You can use the elaboration callback to provide elaboration and validation that extends beyond the default range checking. An elaboration callback is defined by setting the `ELABORATION_CALLBACK` module property to be the name of the elaboration callback procedure, as shown in Example 9-9. This elaboration procedure displays an error if you select a baud rate of 38400 and odd parity.

You can also use the elaboration callback to set the value of derived parameters. Derived parameters are parameters that are derived from other parameters; their values are not editable by the user and thus are not saved in the Qsys System File (`.qsys`). You indicate that a parameter is derived by setting the parameter's `DERIVED` property to `true`. In Example 9-9 `BAUDRATE_PRESCALE` is a derived parameter whose value is 1/16 of the value of the `BAUDRATE` parameter.

You can also use the elaboration callback to change interface properties or add new interfaces as a function of parameter values. You can enable and disable interfaces from the elaboration callback if they are only needed for some parameterizations of the component. Example 9-9 shows how an Avalon-MM slave interface can be included in an instance of the component, based on the `USE_STATUS_INTERFACE` parameter.

Example 9-9. Elaboration Callback

```
# Declare the callback.
set_module_property ELABORATION_CALLBACK my_elaboration_callback

# Add the BAUDRATE_PRESCALE parameter, and indicate that it's derived
add_parameter BAUDRATE_PRESCALE int 600
set_parameter_property BAUDRATE_PRESCALE DERIVED true

# Add the PARITY parameter
add_parameter PARITY string ODD
set_parameter_property PARITY ALLOWED_RANGES {EVEN ODD}

# Add the USE_STATUS_INTERFACE parameter
add_parameter USE_STATUS_INTERFACE boolean

# Declare the status slave interface
add_interface status_slave avalon end
set_interface_property status_slave associatedClock clock_sink
set_interface_property status_slave associatedReset clock_sink
set_interface_property status_slave enabled false

# Declare signals
add_interface_port status_slave st_readdata readdata output 16
add_interface_port status_slave st_read read input 1
add_interface_port status_slave st_write write input 1
add_interface_port status_slave st_waitrequest waitrequest output 1
add_interface_port status_slave st_address address input 24
add_interface_port status_slave st_writedata writedata input 16
```

Elaboration Callback (continued)

```
The elaboration callback
proc my_elaboration_callback {} {
# Get the current value of parameters we care about
  set br [get_parameter_value BAUD_RATE]
  set p [get_parameter_value PARITY]
  set use_status [get_parameter_value USE_STATUS_INTERFACE]

  # Display a warning for invalid combinations.
  if {($br==38400) && ([string compare ${p} "ODD"]==0)} {
    send_message warning "Odd parity at 38400 bps is not supported."
  }
  # Set the value of our derived parameter
  set bp [expr $br / 16]
  set_parameter_value BAUDRATE_PRESCALE $bp

  # Optionally add the status interface
  if { $use_status } {
    set_interface_property status_slave ENABLED true
  } else {
    set_interface_property status_slave ENABLED false
  }
}
```



The elaboration callback is not called when parameters with `AFFECTS_ELABORATION=false` are changed by the user of the component.

Fileset Callbacks

Example 9-10. Fileset Callback Example

```
# Add Simulation and Quartus Synth filesets
add_fileset my_synthesis_fileset QUARTUS_SYNTH fileset_callback_procedure
add_fileset my_verilog_simulation_fileset SIM_VERILOG fileset_callback_procedure

# Set top level name for static entity
set_fileset_property my_synthesis_fileset TOP_LEVEL simple_uart
set_fileset_property my_verilog_simulation_fileset TOP_LEVEL simple_uart

# Implement fileset callback
proc fileset_callback_procedure { entityName } {
    # This check should never fail and is not required. It is shown for clarity of entityName
    contents.
    if { [string compare entityName simple_uart ] != 0 } {
        send_message error " Unexpected entity name; required simple_uart received $entityName"
    }
}

# get parameter values
set p1 [get_parameter_value PARAMETER_ONE]
set car [get_parameter_value CSR_ENABLED]

#Your callback needs to add any generated files to the fileset. Perhaps through
generation
set fileLocation [ create_temp_file ./HDL/simple_uart.v ]

exec perl uart_generation.pl lang-VERILOG dir-`${fileLocation} name-`${entityName} p1-
`${p1} car-`${car}

add_fileset_file ./simple_uart.v VERILOG PATH `${fileLocation}
}
```

Implementing Composed Components

You can use a composition callback to define components that are constructed from combinations of other components. Compose commands can be used in either the main program or in a composition callback.


- **Main program**—In the main program, you can use compose commands such as `add_instance`, `set_instance_parameter_value`, and `add_connection` to create and parameterize subcomponent instances. However, any parameter queries return the default value because the main program is run before parameterization.
- **Composition callback**—After you have set up the basic component template in the main program, you can then use a composition callback to instantiate and parameterize subcomponents as a function of the composed component's parameter values. You define a composition callback by setting the `COMPOSITION_CALLBACK` module property to the name of the composition callback function.

When used, a composition callback replaces elaboration and generation. Your component's interface information is collected by analyzing the interfaces on exported subcomponents. HDL is generated by generating all of your subcomponents and a top-level that stitches them all together. [Figure 9-1 on page 9-3](#) illustrates the steps to define a composed component.

Exporting an interface means that you are making the interface visible from the outside of your component, instead of connecting it internally. Set the `EXPORT_OF` property of the externally visible interface to indicate that it is an exported view of the submodule's interface. Refer to [“get_interface_properties” on page 9-36](#) for the format of the `EXPORT_OF` property. You can set this from the main program or the composition callback.

Exporting an interface is different than defining an interface. The exported interface is an exact copy of the subcomponent's interface; you are not allowed to change any properties on the exported interface. For example, if the internal interface is a 32-bit Avalon-MM master without bursting then the exported interface will be as well.

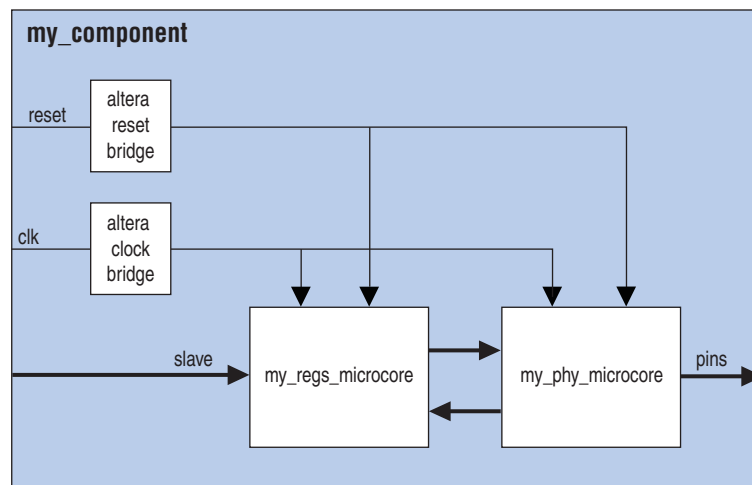
 You cannot export an interface and connect it internally.

 Because the exported interface is a copy of the inner interface, no alteration is possible between the exported and internal interfaces.

When you create an exported interface, the properties of the exported interface are copied from the subcomponent's interface without modification. Ports are copied from the subcomponent's interface with only one modification—the names of the exported ports on the composed component are chosen to ensure they are unique.

[Figure 9-3](#) is a block diagram for the composed component in [Example 9-11](#).

Figure 9-3. Top-Level of a Composed Component



Example 9-11 provides an example of a composed `_hw.tcl` file that instantiates two subcomponents. It connects them together, also connecting the clocks and resets. Note that a clock bridge and reset bridge components are required to allow both subcomponents to see a common clock and reset inputs.

Example 9-11. Composed `_hw.tcl` File with Two Subcomponents

```
package require -exact qsys 11.1
set_module_property name my_component
...
add_instance clk altera_clock_bridge
add_instance phy my_phy_microcore
add_interface clk clock end
add_instance reset altera_reset_bridge
set_interface_property clk EXPORT_OF clk.in_clk
add_instance regs my_regs_microcore
set_instance_property_value reset synchronous_edges deassert

add_interface reset reset end
set_interface_property reset EXPORT_OF reset.in_reset

add_interface pins conduit end
set_interface_property pins EXPORT_OF phy.pins

add_interface slave avalon slave
set_interface_property slave EXPORT_OF regs.slave

add_connection clk.out_clk reset.clk

add_connection clk.out_clk phy.clk
add_connection reset.out_reset phy.clk_reset

add_connection clk.out_clk regs.clk
add_connection reset.out_reset regs.reset
add_connection phy.output regs.input
add_connection regs.output phy.input
```

Hardware Tcl Command Reference

This section provides a reference for all hardware Tcl commands, as follows:

- “Module Definition” on page 9-17
- “Parameters” on page 9-22
- “Display Items” on page 9-31
- “Interfaces and Ports” on page 9-34
- “Composition” on page 9-41
- “Fileset Generation” on page 9-48

The description of each command indicates the phases during which it is available: in the main body of the program (main), or during the elaboration, composition, and generation callback phases, or any combination. Table 9-2 summarizes the commands and provides a reference to the full description.

Table 9-2. Command Summary ⁽¹⁾ (Part 1 of 2)

Command	Full Description
Module Definition	
package <require> -exact qsys <version>	page 9-17
get_module_properties	page 9-17
get_module_property <propertyName>	page 9-19
set_module_property <propertyName> <propertyValue>	page 9-19
get_module_ports	page 9-19
get_module_assignments	page 9-20
get_module_assignment <moduleName>	page 9-20
set_module_assignment <moduleName> [value]	page 9-20
add_documentation_link <title> <fileOrUrl>	page 9-21
send_message <messageLevel> <messageText>	page 9-21
Parameters	
add_parameter <parameterName> <parameterType> [<defaultValue> <description>]	page 9-22
get_parameters	page 9-23
get_parameter_properties	page 9-23
get_parameter_property <parameterName> <propertyName>	page 9-29
set_parameter_property <parameterName> <propertyName> <value>	page 9-29
get_parameter_value <parameterName>	page 9-29
set_parameter_value <parameterName> <value>	page 9-30
decode_address_map <address_map_XML_string>	page 9-30
Display Items	
add_display_item <groupName> <id> <type> [<additionalInfo>]	page 9-31
get_display_items	page 9-33
get_display_item_properties	page 9-33
get_display_item_property <itemName> <propertyName>	page 9-33
set_display_item_property <itemName> <propertyName> <value>	page 9-33
Interfaces and Ports	
add_interface <interfaceName> <interfaceType> <direction> [<associatedClock>]	page 9-35
get_interfaces <interfaceName>	page 9-35
get_interface_property <interfaceName> <propertyName>	page 9-36
set_interface_property <interfaceName> <propertyName> <value>	page 9-37
add_interface_port <interfaceName> <portName> <portRole> [<direction> <width_expr>]	page 9-37
get_interface_ports [<interfaceName>]	page 9-38

Table 9-2. Command Summary ⁽¹⁾ (Part 2 of 2)

Command	Full Description
<code>get_port_properties</code>	page 9-38
<code>get_port_property <portName> <propertyName></code>	page 9-39
<code>set_port_property <portName> <propertyName> [<value>]</code>	page 9-40
<code>get_interface_assignments</code>	page 9-40
<code>get_interface_assignment <interfaceName> <name></code>	page 9-40
<code>set_interface_assignmet <interfaceName> <name> [<value>]</code>	page 9-41
Composition	
<code>add_instance <instanceName> <instanceType> <version></code>	page 9-41
<code>get_instances</code>	page 9-42
<code>get_instance_parameters <instanceName></code>	page 9-42
<code>set_instance_parameter <instanceName> <parameterName> <parameterValue></code>	page 9-42
<code>get_instance_parameter_value <instanceName> <parameterName></code>	page 9-42
<code>get_instance_parameter_properties <instanceName> <parameterName></code>	page 9-43
<code>get_instance_parameter_property <instanceName> <parameterName> <propertyName></code>	page 9-43
<code>get_instance_interfaces <instanceName></code>	page 9-44
<code>get_instance_interface_properties <instanceName> <interfaceName></code>	page 9-44
<code>get_instance_interface_property <instanceName> <interfaceName> <propertyName></code>	page 9-44
<code>get_instance_interface_ports <instanceName> <portName></code>	page 9-45
<code>get_instance_port_property <instanceName> <interfaceName> <propertyName></code>	page 9-45
<code>add_connection [<instanceName>] <startInterface> <endInterface></code>	page 9-46
<code>get_connections</code>	page 9-46
<code>get_connection_parameters <instanceName></code>	page 9-47
<code>get_connection_parameter <connectionName> <parameterName></code>	page 9-47
<code>set_connection_parameter_value <connectionName> <parameterName> <parameterValue></code>	page 9-47
Fileset Generation	
<code>add_fileset <filesetName> <filesetKind> <callbackProcName> [<displayName>]</code>	page 9-48
<code>add_fileset_file <fileDestination> <fileKind> <fileSource> <contentsOrPath> [<attributes>]</code>	page 9-49
<code>set_fileset_property</code>	page 9-50
<code>create_temp_file <fileName></code>	page 9-50
Miscellaneous	
<code>check_device_family_equivalence</code>	page 9-50
<code>get_device_family_displayname</code>	page 9-51
<code>get_qip_strings</code>	page 9-51
<code>set_qip_strings</code>	page 9-51

Note to Table 9-2:

(1) Arguments enclosed in []'s are optional

Module Definition

This section provides information about the commands that you use to define and query a module.

package

The `package` command allows you to specify a particular version of the Qsys software to avoid software compatibility issues. You must use the `package` command at the beginning of your `_hw.tcl` file. When used, the component files behave as if they are interpreted by the version of the Qsys software that you specify. When the `package` command is not used, installed version of the Qsys software is assumed. This document describes the behavior of component which start with `package require -exact qsys 11.1`. For earlier releases, refer to the documentation for that release.

package	
Callback availability	Main (before any other commands in the file)
Usage	<code>package require -exact qsys <version></code>
Returns	None
Arguments	<code>version</code> The version of Qsys that you require, specified as decimal number
Example	<code>package require -exact qsys 11.1</code>

get_module_properties

This command returns the names of all the available module properties as a list of strings. You can use the `get_module_property` and `set_module_property` commands to get and set values of individual properties. The value returned by this command is always the same for a particular version of Qsys.

get_module_properties	
Callback availability	Main, elaboration, generation, and composition
Usage	<code>get_module_properties</code>
Returns	List of strings
Arguments	None
Example	<code>get_module_properties</code>

Table 9-3 lists the available module properties, their use, and the phases in which they can be set.

Table 9-3. Module Properties

Property Name	Property Type	Can Be Set	Description
ANALYZE_HDL	Boolean	Main program	When set to <i>false</i> , prevents a call to the Quartus II mapper to verify port widths and directions, speeding up generation time at the expense of fewer validation checks. If this property is set to <i>false</i> , invalid port widths and directions are discovered during Quartus II compilation.
AUTHOR	String	Main program	The module's author.
DESCRIPTION	String	Main program	The description of the module, such as "Example Qsys Module."
DISPLAY_NAME	String	Main program	The name to display when referencing the module, such as "My SOPC Component."
EDITABLE	Boolean	Main program	Indicates whether the component is editable in the component editor.
ELABORATION_CALLBACK	String	Main program	The name of the elaboration callback. For static and generated components, the default elaborations used if this property is not set.
GROUP	String	Main program	The component group that the module belongs to, such as "Example Components."
ICON_PATH	String	Main program	A path to an icon to display in the module's parameter editor.
INTERNAL	Boolean	Main program	A component which is marked as internal does not appear in the Qsys component library. This feature allows you to hide the submodules of a larger composed component.
NAME	String	Main program	The name of the module, such as <i>my_sopc_component</i> .
OPAQUE_ADDRESS_MAP	String	Main program	For composed components created using a <i>_hw.tcl</i> file that include children that are Avalon-MM slaves, specifies whether the children's addresses are visible to downstream software tools. When <i>true</i> , the children's address are not visible. When <i>false</i> , the children's addresses are visible.
VERSION	String	Main program	The module's version, such as <i>10.0</i>
COMPOSE_CALLBACK	String	Main Program	The name of the compose callback. If you define a compose callback, you must not define the generation or elaboration callbacks.

get_module_assignments


This command returns names of the module assignment variables.

get_module_assignments	
Callback availability	Main, elaboration, and composition
Usage	get_module_assignments
Returns	String
Arguments	None
Example	get_module_assignments

get_module_assignment

This command returns the value of the specified argument. You can use the `get_module_assignment` and `set_module_assignment` and the `get_interface_assignment` and `set_interface_assignment` commands to transfer information about hardware components to embedded software tools and applications.

get_module_assignment	
Callback availability	Main, elaboration, and composition
Usage	get_module_assignment <name>
Returns	String
Arguments	name The name whose value is being retrieved
Example	get_module_assignment embeddedsw.CMacro.colorSpace

 For more information about specifying information for software tools, refer to *Publishing Component Information to Embedded Software* in the *Nios II Software Developer's Handbook*.

set_module_assignment

This command sets the value of the specified argument.

set_module_assignment	
Callback availability	Main, elaboration, and composition
Usage	set_module_assignment <name> [<value>]
Returns	None
Arguments	name The name whose value is being set value The value of the <name> argument
Example	set_module_assignment embeddedsw.CMacro.colorSpace CMYK

add_documentation_link

This command allows you to add multiple documentation links for a single component.

add_documentation_link		
Callback availability	Main	
Usage	add_documentation_link filename <title> <fileOrUrl>	
Returns	None	
Arguments	title	The title of the document for use on menus and buttons.
	fileOrUrl	A path to the component documentation, using a syntax that provides the entire URL, not a relative path. For example: http://www.mydomain.com/my_memory_controller.html or file:///datasheet.txt .
Example	add_documentation_link "Avalon Verification IP Suite User Guide" http://www.altera.com/literature/ug/ug_avalon_verification_ip.pdf	

send_message

This command 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.

send_message		
Callback availability	Main, elaboration, fileset, and composition	
Usage	send_message <messageLevel> <messageText>	
Returns	None	
Arguments	messageLevel	The following 6 message levels are supported: <ul style="list-style-type: none"> ■ Error—Provides an error message. The Qsys system cannot be generated while there are error messages. ■ Warning—Provides a warning message. ■ Info—Provides an informational message. ■ Progress—Reports progress during generation. ■ Debug—Provides messages when debug mode is enabled.
	messageText	The text of the message
Example	send_message Error "param1 must be greater than param2."	

Parameters

Parameters allow users of your component to affect its operation in the same manner as Verilog HDL parameters or VHDL generics.

add_parameter

This command adds a parameter to your component. Most of the parameter types are self-explanatory because they are used in the C programming language or HDL. However, the `string_list` and `integer_list` parameters that are used to create tables in GUIs require some explanation.

- When you use the `add_parameter` command with a `string_list` or `integer_list` parameter type, the parameter you define is displayed in a variable-sized table that includes **add** and **remove** buttons.
- If you define multiple parameters of type `string_list` or `integer_list`, you can also use the `add_display_item` command to specify that parameters should each be displayed as a column in a table, each parameter of type `string_list` or `integer_list` becomes a column in the table. [Example 9-12](#) illustrates the use of the `integer_list` parameter types to create a multi-column table.

Example 9-12. Creating Tables Using the `string_list` and `integer_list` Parameter Types

```
add_parameter bitWide INTEGER
add_parameter divider INTEGER
add_parameter coefficients INTEGER_LIST
add_parameter positions INTEGER_LIST
add_display_item myTable coefficients TABLE
add_display_item myTable positions TABLE
```

add_parameter		
Callback availability	Main	
Usage	<code>add_parameter <parameterName> <parameterType> [<defaultValue> <description>]</code>	
Returns	String	
Arguments	<code>parameterName</code>	A name that you, the component author, choose for your parameter
	<code>parameterType</code>	The following types are supported: Integer, Natural, Positive, Boolean, Std_logic, Std_logic_vector, String, String_list, and Integer_list.
	<code>defaultValue</code>	The default length of the parameter is derived from its range
	<code>description</code>	Explains the use of the parameter
Example	<code>add_parameter seed integer 17 "The seed to use for data generation."</code>	

get_parameters

This command returns the names of all parameters that have been previously defined by `add_parameter` as a space-separated list.

get_parameters	
Callback availability	Main, elaboration, fileset, and composition
Usage	<code>get_parameters</code>
Returns	List of strings
Arguments	None
Example	<code>set parameter_summary [get_parameters]</code>

get_parameter_properties

This command returns a list of all the available parameter properties as a list of strings. The `get_parameter_property` and `set_parameter_property` commands are used to get and set the values of these properties, respectively.

get_parameter_properties	
Callback availability	Main, elaboration, fileset, and composition
Usage	<code>get_parameter_properties</code>
Returns	List of strings
Arguments	None
Example	<code>set property_summary [get_parameter_properties]</code>

Table 9-4 describes the properties available to describe the behaviors of each of the parameters you can specify, their use, and when they can be set.

Table 9-4. Parameter Properties (Part 1 of 3)

Property Name	Type/Default	Can Be Set	Description
AFFECTS_ELABORATION	Boolean, true	Main program	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 isNonVolatileStorage. An example of a parameter that does affect the external interface is width. When the value of a parameter changes, if that parameter has set AFFECTS_ELABORATION=false, the elaboration phase (calling the callback or hardware analysis) is not repeated, improving performance. Because the default value of AFFECTS_ELABORATION is true, the provided HDL file is normally re-analyzed to determine the new port widths and configuration every time a parameter changes.
AFFECTS_GENERATION	Boolean, refer to description	Main program	The default value of AFFECTS_GENERATION is false if you provide a top-level HDL module; it 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.
ALLOWED_RANGES	String, ""	Main program	Indicates the range or ranges that the parameter value can have. For integers, The ALLOWED_RANGES property is a list of ranges that the parameter can take on, where each range is a single value, or a range of values defined by a start and end value separated by a colon, such as 11:15. This property can also specify legal values and display strings for integers, such as {0:None 1:Monophonic 2:Stereo 4:Quadrophonic} meaning 0,1,2,4 are the legal values. You can also assign longer strings to be displayed in the parameter editor to string variables. For example, ALLOWED_RANGES {"dev1:Cyclone IV GX" "dev2:Stratix V GT"} Refer to Example 9-8 on page 9-9 and Figure 9-2 on page 9-10 for additional examples illustrating the use of this property.
DEFAULT_VALUE	String or Boolean	Main program	The default value.
DERIVED	Boolean, false	Elaboration callback	When true, indicates that the parameter value does not need to be stored, typically because it is set from the elaboration callback. The default value is false.
DESCRIPTION	String, ""	Main program	A user-visible description of the parameter.

Table 9-4. Parameter Properties (Part 2 of 3)

Property Name	Type/Default	Can Be Set	Description
DISPLAY_HINT	String, ""	Main program	<p>Provides a hint about how to display a property. The following values are possible:</p> <ul style="list-style-type: none"> ■ <code>boolean</code>—for integer parameters whose value can be 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 instead of a drop-down list. ■ <code>hexadecimal</code>—for integer parameters, display and interpret the value as a hexadecimal number, for example: <code>0x00000010</code> instead of 16. ■ <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. <p>Refer to Example 9-8 on page 9-9 and Figure 9-2 on page 9-10 for examples illustrating the use of this property.</p>
DISPLAY_NAME	String, ""	Main program	This is the GUI label that appears to the left of the parameter.
DISPLAY_UNITS	String, ""	Main program	This is the GUI label that appears to the right of the parameter.
ENABLED	Boolean, true	Main program and elaboration callbacks	When <code>false</code> , the parameter is disabled, meaning that it is displayed, but greyed out, indicating that it is not editable on the parameter editor.
GROUP	String, ""	Main	Controls the layout of parameters in GUI. Refer to Example 9-8 for an illustration of its use.
HDL_PARAMETER	Boolean, false	Main program	When <code>true</code> , the parameter must be passed to the HDL component description. The default value is <code>false</code> .
NEW_INSTANCE_VALUE	String, ""	Main program	This property allows you to change the default value of a parameter without affecting older components that have assigned a default value to this parameter using the <code>defaultValue</code> argument. The practical result is that older components will continue to use <code>defaultValue</code> for the parameter and newer components can use the value assigned by <code>NEW_INSTANCE_VALUE</code> .

Table 9-4. Parameter Properties (Part 3 of 3)

Property Name	Type/Default	Can Be Set	Description
SYSTEM_INFO	String, ""	Main program	<p>Allows you to assign information about the instantiating system to a parameter that you define. SYSTEM_INFO requires a keyword argument specifying the type of information requested, <info-type>. <info-type> may also take an argument. The syntax of the Tcl command is:</p> <pre>set_parameter_property my_parameter SYSTEM_INFO <info-type> [<arg>]</pre> <p>The following values for <info-type> are predefined: ADDRESS_MAP, ADDRESS_WIDTH, CLOCK_DOMAIN, CLOCK_RATE, CLOCK_RESET_INFO, CUSTOM_INSTRUCTION_SLAVES, DEVICE, DEVICE_FAMILY, DEVICE_FEATURES, INTERRUPTS_USED, MAX_SLAVE_DATA_WIDTH, RESET_DOMAIN, and TRISTATE_ONDUIT_MASTERS</p> <p>Refer to Table 9-5 for descriptions of the <info_type> argument.</p>
SYSTEM_INFO_TYPE	Various	Main program	Specifies one of the types of information listed in Table 9-5 on page 9-27 .
SYSTEM_INFO_ARG	String, ""	Main program	Defines an argument to be passed to a particular SYSTEM_INFO function.
TYPE	String, ""	Main program	Specifies one of the following types: INTEGER, NATURAL, POSITIVE, BOOLEAN, STD_LOGIC, STD_LOGIC_VECTOR, STRING, STRING_LIST, INTEGER_LIST, LONG, or FLOAT.
UNITS	String, ""	Main program	<p>Sets the units of the parameter. The following values are possible: None, Picoseconds, Nanoseconds, Microseconds, Milliseconds, Seconds, Hertz, Kilohertz, Megahertz, Gigahertz, Address, Bits, Bytes, Kilobytes, Megabytes, Gigabytes, BitsPerSecond, KiloBitsPerSecond, MegaBitsPerSecond, GigaBitsPerSecond, Percent, and Cycles. For example,</p> <pre>set_parameter_property frequency UNITS GIGAHERTZ</pre>
VISIBLE	Boolean, true	Main program elaboration, callbacks	Indicates whether or not to display the parameter in the parameterization GUI.
WIDTH	String, ""	Main program	

- Table 9-5 describes the properties that you can use with the `system_info` parameter property. For more information about how to use the `system_info` parameter property, refer to “[SYSTEM_INFO Parameters](#)” on page 9-5.

Table 9-5. SYSTEM_INFO Properties (Part 1 of 2)

Property	Type	Description
ADDRESS_MAP	String	Assigns an XML-formatted string describing the address map to the parameter you specify. <code>set_parameter_property <my_parameter> SYSTEM_INFO {ADDRESS_MAP <my_avalon-mm_master>}</code>
ADDRESS_WIDTH	Integer	Assigns an integer to the parameter you specify that is the number of bits an Avalon-MM master must drive to address all of its slaves, using byte addresses. <code>set_parameter_property <my_parameter> SYSTEM_INFO {ADDRESS_WIDTH <my_avalon-mm_master>}</code>
CLOCK_DOMAIN	Integer	Assigns an integer representing the clock domain to the parameter you specify. You can use this command to determine whether multiple interfaces in your module are on the same clock domain. The absolute value of the integer value is arbitrary, but if two interfaces are on the same clock domain, the <code>CLOCK_DOMAIN</code> value is guaranteed to be the same and greater than zero. <code>set_parameter_property <my_parameter> SYSTEM_INFO {CLOCK_DOMAIN <my_clk>}</code>
CLOCK_RATE	Integer or String	Assigns a positive number, which is the clock frequency in Hz to the clock input interface you specify. Assigns 0 if the clock rate is not known. <code>set_parameter_property <my_parameter> SYSTEM_INFO {CLOCK_RATE <my_clk>}</code>
CLOCK_RESET_INFO	String	Specifies the name of the module's clock or reset sink interface. (Specifies the clock sink interface for designs that use a global reset.)
CUSTOM_INSTRUCTION_SLAVES	String	Provides custom instruction slave information, including the name, base address, address span, and clock cycle type.
DEVICE_FAMILY	String	Assigns the family name (not the specific device part number) of the currently selected device to the parameter you specify. <code>set_parameter_property <my_parameter> SYSTEM_INFO {DEVICE_FAMILY}</code>
DEVICE_FEATURES	String	Creates a list of key/value pairs delineated by spaces indicating whether a particular device feature is available in the currently selected device family. The format of the list is suitable for passing to the Tcl array <code>set</code> command. This list is assigned to the parameter you specify. The following features are supported: <code>M512_MEMORY</code> , <code>M4K_MEMORY</code> , <code>M9K_MEMORY</code> , <code>M144K_MEMORY</code> , <code>MRAM_MEMORY</code> , <code>MLAB_MEMORY</code> , <code>ESB</code> , <code>DSP</code> , and <code>EMUL</code> . <code>set_parameter_property <my_parameter> SYSTEM_INFO {DEVICE_FEATURES}</code>
INTERRUPTS_USED	Integer or string	Creates a mask indicating which bits of the interrupt receiver vector are connected to an interrupt sender. This mask is assigned to the parameter you specify. You can use this interrupt mask to optimize logic that handles interrupts. <code>set_parameter_property <my_parameter> SYSTEM_INFO {INTERRUPTS_USED <my_interrupt_receiver>}</code>

Table 9-5. SYSTEM_INFO Properties (Part 2 of 2)

Property	Type	Description
MAX_SLAVE_DATA_WIDTH	Integer	Assigns an integer to the parameter you specify that is the data width of the widest slave connected to the specified Avalon-MM master. <pre>set_parameter_property <my_parameter> SYSTEM_INFO {MAX_SLAVE_DATA_WIDTH <my_avalon_mm_master>}</pre>
RESET_DOMAIN	Integer	Assigns an integer representing the reset domain to the parameter you specify. You can use this command to determine whether multiple interfaces in your module are on the same reset domain. The absolute value of the integer value is arbitrary, but if two interfaces are on the same reset domain, the RESET_DOMAIN value is guaranteed to be the same and greater than zero. <pre>set_parameter_property <my_parameter> SYSTEM_INFO {RESET_DOMAIN <my_reset>}</pre>
UNIQUE_ID	String	A string guaranteed to be unique to this module.
TRISTATECONDUIT_MASTERS	String	Specifies the name or names of the module's interfaces that are tri-state conduit slaves.
TRISTATECONDUIT_INFO	String	Returns an XML string containing information about the Avalon-TC masters connected to the specified Avalon-TC slave interface on a given component. The returned string may include all of the following information: <ul style="list-style-type: none"> ■ The Avalon-TC slave interface name ■ The Avalon-TC master module and interface names ■ The Avalon-TC signal names, directions, and widths <p>The argument to SYSTEM_INFO_ARG is a regular expression that specifies the interface or interfaces of interest. The following example returns an XML string named TC_slave_info for TC slave interface named CFI_FLASH.uas:</p> <pre>add_parameter TC_slave_info string "" set_parameter_property TC_slave_info SYSTEM_INFO_TYP TRISTATECONDUIT_INFO set_parameter_property TC_slave_info SYSTEM_INFO_ARG "uas"</pre> <p>To retrieve information about all the slave interfaces on a module substitute "*" for the interface name, as the following example illustrates:</p> <pre>set_parameter_property TC_slave_info SYSTEM_INFO_ARG "*"</pre>

get_parameter_property

This command returns a single parameter property.

get_parameter_property		
Callback availability	Main, elaboration, fileset, and composition	
Usage	get_parameter_property <parameterName> <propertyName>	
Returns	string, boolean, or units, depending on property. Refer to Table 9-4 on page 9-24 .	
Arguments	parameterName	The name of the parameter whose property value is being retrieved
	propertyName	One of the properties listed in Table 9-4 on page 9-24
Example	get_parameter_property parameter1 GROUP	

set_parameter_property

This command sets a single parameter property.

set_parameter_property		
Callback availability	Main, elaboration, and composition	
Usage	set_parameter_property <parameterName> <propertyName> <value>	
Returns	string, boolean, or units depending on property	
Arguments	parameterName	Specifies the parameter that is being set
	propertyName	Specifies the property of parameterName that is being set, refer to Table 9-4 on page 9-24 for a list of properties
	value	Provides the values
Example	set_parameter_property BAUD_RATE ALLOWED_RANGES {9600 19200 38400}	

get_parameter_value

This command returns the current value of a parameter defined previously with the add_parameter command.

get_parameter_value		
Callback availability	Elaboration ⁽¹⁾ , fileset, and composition	
Usage	get_parameter_value <parameterName>	
Returns	String	
Arguments	parameterName	Specifies the parameter that is being retrieved
Example	set fifo_width [get_parameter_value fifo_width]	

Note:

- (1) If AFFECTS_ELABORATION=false for a given parameter, get_parameter_value is not available for that parameter from the elaboration callback. If AFFECTS_GENERATION=false then it is not available from the generation callback.

set_parameter_value

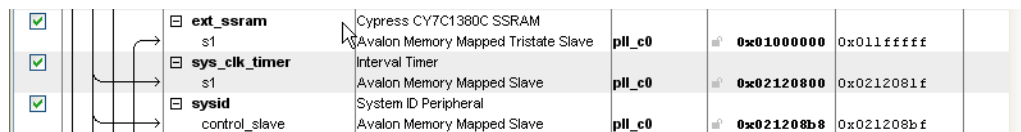
This command sets a parameter value. The values of derived parameters can be set from the elaboration callback.

set_parameter_value		
Callback availability	Elaboration and composition	
Usage	set_parameter_value <parameterName> <value>	
Returns	None	
Arguments	parameterName	Specifies the parameter that is being set
	value	Specifies the value of parameterName
Example	set_parameter_value BAUD_RATE 19200	

decode_address_map

This utility function converts an XML-formatted address map into a list of Tcl lists. Each inner list is in the correct format for conversion to an array. The XML code describing each slave includes; its name, start address, and end address + 1. [Figure 9-4](#) shows a portion of a Qsys system with three Avalon-MM slave devices.

Figure 9-4. Qsys System with Three Avalon-MM Slaves



[Example 9-13](#) shows the XML that describes the address map for the Avalon-MM master that accesses these slaves. The format of the XML string provided may differ from that described here; it may have different white space between the elements and could include additional attributes or elements. Using the decode_address_map command to decode the XML representing an Avalon-MM master's address map is easier and ensures that your code will work with future versions of the XML address map.



Altera recommends that you use the code provided in the description of [Example 9-13](#) to enumerate over the components within an address map, rather than writing your own parser.

Example 9-13. Address Map for an Avalon-MM Master

```
<address-map>
  <slave name='ext_ssram' start='0x01000000' end='0x01200000' />
  <slave name='sys_clk_timer' start='0x02120800' end='0x02120820' />
  <slave name='sysid' start='0x021208B8' end='0x021208C0' />
</address-map>
```

decode_address_map			
Callback availability	Elaboration, generation, and composition		
Usage	<code>decode_address_map <address_map_XML_string></code>		
Returns	List of Tcl lists, each one suitable for passing to array set		
Arguments	<table border="1"> <tr> <td><code>address_map_XML_string</code></td> <td>An XML string describing the address map of an Avalon-MM master.</td> </tr> </table>	<code>address_map_XML_string</code>	An XML string describing the address map of an Avalon-MM master.
<code>address_map_XML_string</code>	An XML string describing the address map of an Avalon-MM master.		
Example	<pre>set address_map_xml [get_parameter_value my_map_param] set address_map_dec [decode_address_map \$address_map_xml] foreach i \$address_map_dec { array set info \$i send_message info "Connected to slave \$info(name)" }</pre>		

Display Items

You specify your component GUI using the display commands.

add_display_item

You can use this command to specify the following aspects of component display:

- You can create logical groups for a component's parameters. For example, you might want to create separate groups for the component's timing, size, and simulation parameters. A component displays the groups and parameters in the order that you specify the display items for them in the `_hw.tcl` file.
- You can create multicolumn tables to present a component's parameters. Refer to [Example 9-12 on page 9-22](#) for an example that illustrates multicolumn tables.
- You can specify an image to provide a pictorial representation of a parameter or parameter group.
- You can create a button by adding a display item of type `action`. The display item includes the name of the callback to run when the action is performed.

You create a display group by adding display items to it.

add_display_item		
Callback availability	Main	
Usage	<code>add_display_item <groupName> <id> <type> [<additionalInfo>]</code>	
Returns	String	
Arguments	<code>groupName</code>	Specifies the group to which a display item belongs.
	<code>id</code>	Specifies the parameter or icon to be displayed in a group. Each display item associated with a component must have a different ID.
	<code>type</code>	Specifies the category of the display item. The following types are defined: <ul style="list-style-type: none"> ■ <code>icon</code>—a <code>.gif</code>, <code>.jpg</code>, or <code>.png</code> file ■ <code>parameter</code>—a parameter in the instance ■ <code>text</code>—a block of text ■ <code>group</code>—a group. If the <code>groupName</code> is also defined, the new group is a child of the <code>groupName</code> group. If <code>groupName</code> is an empty string, the group is top-level. ■ <code>action</code>—an action defined by a callback procedure when you click the button labeled by <code>actionName</code>.
	<code>additionalInfo</code>	Provides extra information required for display items. The following examples illustrate how you use the <code>additionalInfo</code> argument for the various types: <ul style="list-style-type: none"> ■ <code>add_display_item groupName id icon path-to-image-file</code> ■ <code>add_display_item groupName parameterName parameter</code> (<code>additionalInfo</code> not required) ■ <code>add_display_item groupName id text "your-text"</code> The <code>your-text</code> argument is a block of text that is displayed in the GUI. Some simple HTML formatting is allowed, such as <code></code> and <code><i></code>, if the text starts with <code>"html"</code>. ■ <code>add_display_item parentGroupName childGroupName group [tab]</code> The <code>tab</code> is an optional parameter. If present, the group appears in separate tab in the GUI for the instance. ■ <code>add_display_item parentGroupName actionName action</code> <code>buttonClickCallbackProc</code>
Examples	<pre>add_display_item timing read_latency parameter add_display_item sound speaker icon speaker.jpg</pre>	

get_display_items

This command returns a list of all items to be displayed as part of the parameterization GUI.

get_display_items	
Callback availability	Main, elaboration, fileset, and composition
Usage	<code>get_display_items</code>
Returns	List of strings
Arguments	None
Example	<code>get_display_items</code>

get_display_item_properties

This command returns a list of names of the properties of display items that are part of the parameterization GUI.

get_display_item_properties	
Callback availability	Main
Usage	<code>get_display_item_properties</code>
Returns	List of strings
Arguments	None
Example	<code>get_display_item_properties</code>

get_display_item_property

This command returns the value of specific property of a display item that is part of the parameterization GUI.

get_display_item_property		
Callback availability	Main	
Usage	<code>get_display_item_property <itemName> <propertyName></code>	
Returns	String	
Arguments	<code>itemName</code>	The item whose property value is being retrieved
	<code>propertyName</code>	The property whose value is being retrieved
Example	<code>set my_label [get_display_item_property my_action DISPLAY_NAME]</code>	

set_display_item_property

This command sets the value of specific property of a display item that is part of the parameterization GUI.


set_display_item_property		
Callback availability	Main	
Usage	<code>set_display_item_property <itemName> <propertyName> <value></code>	
Returns	String	
Arguments	<code>itemName</code>	The item whose property value is being set
	<code>propertyName</code>	The property whose value is being set
	<code>value</code>	The value to set
Example	<pre>set_display_item_property my_action DISPLAY_NAME "Click Me" set_display_item_property my_action DESCRIPTION "clicking this button runs the click_me_callback proc in the hw.tcl file"</pre>	

Interfaces and Ports

You can use the interface and port commands to define interfaces and ports and retrieve their properties.

add_interface

This command adds an interface to your module. As the component author, you choose the name of the interface. By default, interfaces are enabled. You can set the interface property `ENABLED` to `false`, to disable a component interface. If an interface is disabled, it is hidden and its ports are automatically terminated to their default values. Signals that you designate as active low by appending an `_n` are terminated to 1. All other signals are terminated to 0.

 The properties available for each interface type are different. The `ENABLED` property applies to all interface types. Refer to the *Avalon Interface Specifications* for a description of other properties.

add_interface																			
Callback availability	Main, elaboration, and compose																		
Usage	<code>add_interface <interfaceName> <interfaceType> <direction></code>																		
Returns	String																		
Arguments	<code>interfaceName</code>	A name that you choose to identify an interface.																	
	<code>interfaceType</code> and <code>direction</code>	<p>There are seven <code>interfaceTypes</code>. The following directions are possible for these <code>interfaceTypes</code></p> <table border="1"> <thead> <tr> <th>Interface Type</th> <th>Direction</th> </tr> </thead> <tbody> <tr> <td><code>avalon</code></td> <td><code>master, slave</code> ⁽¹⁾</td> </tr> <tr> <td><code>tristate_conduit</code></td> <td><code>master, slave</code></td> </tr> <tr> <td><code>avalon_streaming</code></td> <td><code>source, sink</code></td> </tr> <tr> <td><code>interrupt</code></td> <td><code>sender, receiver</code></td> </tr> <tr> <td><code>conduit</code></td> <td><code>end</code></td> </tr> <tr> <td><code>clock</code></td> <td><code>source, sink</code></td> </tr> <tr> <td><code>reset</code></td> <td><code>source, sink</code></td> </tr> <tr> <td><code>nios_custom_instruction</code></td> <td><code>slave</code></td> </tr> </tbody> </table>	Interface Type	Direction	<code>avalon</code>	<code>master, slave</code> ⁽¹⁾	<code>tristate_conduit</code>	<code>master, slave</code>	<code>avalon_streaming</code>	<code>source, sink</code>	<code>interrupt</code>	<code>sender, receiver</code>	<code>conduit</code>	<code>end</code>	<code>clock</code>	<code>source, sink</code>	<code>reset</code>	<code>source, sink</code>	<code>nios_custom_instruction</code>
Interface Type	Direction																		
<code>avalon</code>	<code>master, slave</code> ⁽¹⁾																		
<code>tristate_conduit</code>	<code>master, slave</code>																		
<code>avalon_streaming</code>	<code>source, sink</code>																		
<code>interrupt</code>	<code>sender, receiver</code>																		
<code>conduit</code>	<code>end</code>																		
<code>clock</code>	<code>source, sink</code>																		
<code>reset</code>	<code>source, sink</code>																		
<code>nios_custom_instruction</code>	<code>slave</code>																		
Example	<code>add_interface mm_slave avalon slave</code>																		

Notes:

(1) The terms *master*, *source*, and *start* are interchangeable. The terms *slave*, *sink*, and *end* are interchangeable.

get_interfaces


This command returns the names of all interfaces that have been previously defined by `add_interface` as a space-separated list.

get_interfaces	
Callback availability	Main, elaboration, generation, and composition
Usage	<code>get_interfaces</code>
Returns	List of strings
Arguments	None
Example	<code>set all_interfaces [get_interfaces]</code>

get_interface_properties

This command returns the names of all the available interface properties for the specified interface as a space separated list.

get_interface_properties	
Callback availability	Main, elaborations, and composition
Usage	get_interface_properties <interfaceName>
Returns	List of strings
Arguments	interfaceName The name of an interface that you defined
Example	get_interface_properties mm_slave

 The properties available for each interface type are different. Refer to the *Avalon Interface Specifications* for more information about interface properties.

The interface properties that are common to all interface types are listed below in [Table 9-6](#).

Table 9-6. Interface Properties Common to All Interface Types

Property	Type	Description
EXPORT_OF	String	For composed <code>_hwl.tcl</code> files, the <code>EXPORT_OF</code> property indicates which interface of a child instance is to be exported through this interface. Before using this command, you must have created the border interface using <code>add_interface</code> . The interface to be exported is of the form <code><instanceName.interfaceName></code> . Example: <code>set_interface_property CSC_input EXPORT_OF my_colorSpaceConverter.input_port</code>
ENABLED	Boolean	Specifies whether or not interface is enabled.

get_interface_property

This command returns the value of a single interface property from the specified interface.

get_interface_property	
Callback availability	Main, compose, and elaboration
Usage	get_interface_property <interfaceName> <propertyName>
Returns	string, boolean, or units, depending on property. Refer to the <i>Avalon Interface Specifications</i> for more information about interface properties
Arguments	interfaceName The name of an interface from which you want to retrieve information
	propertyName The name of the property whose value you want to retrieve. This property is either <code>ENABLED</code> or <code>ASSOCIATED_CLOCK</code> or a property name defined by the interface.
Example	get_interface_property mm_slave readWaitTime

set_interface_property

This command sets a single interface property for an interface.

set_interface_property		
Callback availability	Main, compose, and elaboration	
Usage	set_interface_property <interfaceName> <propertyName> <value>	
Returns	String	
Arguments	interfaceName	The name of an interface that includes this property
	propertyName	The name of the property whose value you want to set, which is ENABLED or ASSOCIATED_CLK or a name from the <i>Avalon Interface Specifications</i> .
	value	The value to set for the specified property
Example	set_interface_property mm_slave linewidthBursts false	

add_interface_port

This command adds a port to an interface on your module. As the component author, you determine the name of the port. The port width and direction must be set by the end of the elaboration phase. The port width can be set with one of the following mechanisms:

- A constant width or a width expression can be set in the main program.
- A constant width can be set in the elaboration callback.



Without an elaboration callback, for static components, quartus_map determines the port width from the HDL.

add_interface_port		
Callback availability	Main and elaboration	
Usage	add_interface_port <interfaceName> <portName> <portRole> [<direction> <width_expr>]	
Returns	String	
Arguments	interfaceName	The name of the interface to which the port belongs.
	portName	The name of the port that you, the component author, have chosen.
	portRole	The role of this port within the interfaces. Port roles are referred to as signal types in the <i>Avalon Interface Specification</i> . Refer to the <i>Avalon Interface Specifications</i> for the signal types available for each interface type.
	direction	The direction can be input, output, or bidir
	width_expr	The port's width expression. In simple cases, this is just the width of the port in bits.
Example	add_interface_port mm_slave s0_rdata readdata output 32	

get_interface_ports

This command returns the names of all of the ports that have been added to a given interface. If the interface name is omitted, all ports for all interfaces are returned.

get_interface_ports	
Callback availability	Main, elaboration, and filesset
Usage	<code>get_interface_ports [<interfaceName>]</code>
Returns	String
Arguments	interfaceName The name of the interface whose ports you want to list (optional)
Example	<code>get_interface_ports mm_slave</code>

get_port_properties

This command returns a list of all available port properties.

get_port_properties	
Callback availability	Main, elaboration, filesset, and composition
Usage	<code>get_port_properties <portName></code>
Returns	String, boolean, or units, depending on property. Refer to Table 9-4 on page 9-24
Arguments	<p>portName The name of the port whose properties are required. The following port properties are supported:</p> <ul style="list-style-type: none"> ■ DIRECTION ■ TERMINATION ■ TERMINATION_VALUE ■ VHDL_TYPE ■ WIDTH_VALUE ■ WIDTH_EXPR ■ DRIVEN_BY ■ ROLE <p>Refer to Table 9-7 for a description of these properties.</p>
Example	<code>get_port_properties mm_slave</code>

[Table 9-7](#) describes the available port properties.

Table 9-7. Port Properties (Part 1 of 2)

Name	Type	Description
DIRECTION	input, output, bidir	The direction of the port from the component's perspective.
TERMINATION	boolean	When true, instead of connecting the port to the Qsys system, it is left unconnected for output and bidir or set to a fixed value for input. Has no effect for components that implement a generation callback instead of using the default wrapper generation.

Table 9–7. Port Properties (Part 2 of 2)

Name	Type	Description
TERMINATION_VALUE	integer	The constant value to drive an input port.
VHDL_TYPE	std_logic std_logic_vector auto	indicates the type of a VHDL port. The default value, auto, selects std_logic if the width is fixed at 1, and std_logic_vector otherwise.
WIDTH_VALUE	integer	The width of the port in bits. Cannot be set directly. Any changes must be set through the WIDTH_EXPR property.
WIDTH_EXPR	string	The width expression of a port. The width_value_expr property can be set directly to an integer if desired. When get_port_property is used width always returns the current integer width of the port while width_expr always returns the unevaluated width expression.
DRIVEN_BY	integer, input	Indicates that this output port is always driven to a constant value or by an input port. If all outputs on a component have their driven_by property set to a valid value then the component's HDL is generated automatically.
ROLE	string	Specifies an Avalon signal type such as waitrequest, readdata, or read. For a complete list of signal types, refer to the <i>Avalon Interface Specifications</i> .

get_port_property

This command returns the value of single port property for the specified port.

get_port_property		
Callback availability	Main, elaboration, and fileset	
Usage	get_port_property <portName> <propertyName>	
Returns	Depends on the type of the property	
Arguments	portName	The name of the port
	propertyName	One of the supported properties described in Table 9–7 .
Example	get_port_property rdata WIDTH_VALUE	

set_port_property

This command sets a single port property.

set_port_property		
Callback availability	Main program and elaboration	
Usage	<code>set_port_property <portName> <propertyName> [<value>]</code>	
Returns	String, boolean, or units, depending on property. Refer to Table 9-4 on page 9-24 .	
Arguments	portName	The name of the port
	propertyName	One of the supported properties described in Table 9-7 .
	value	The value to set
Example	<code>set_port_property rdata WIDTH_EXPR 32</code>	

get_interface_assignments

This command returns the value of all interface assignments for the specified interface.

get_interface_assignments		
Callback availability	Main, elaboration, and composition	
Usage	<code>get_interface_assignments <interfaceName></code>	
Returns	String	
Arguments	interfaceName	The name of the Avalon interface whose assignment is being retrieved
Example	<code>get_interface_assignments s1</code>	

get_interface_assignment


This command returns the value of the specified name for the specified interface.

get_interface_assignment		
Callback availability	Main, elaboration, and composition	
Usage	<code>get_interface_assignments <interfaceName> <name></code>	
Returns	String	
Arguments	interfaceName	The name of the Avalon interface whose assignment is being retrieved
	name	The assignment whose value is being retrieved
Example	<code>get_interface_assignment s1 embeddedsw.configuration.isFlash</code>	

set_interface_assignment

This command sets the value of the specified assignment for the specified interface.

set_interface_assignment		
Callback availability	Main, elaboration, and composition	
Usage	set_interface_assignment <interfaceName> <name> [<value>]	
Returns	None	
Arguments	interfaceName	The name of the Avalon interface whose assignment is being set
	name	The assignment whose value is being set
	value	The value to assign
Example	set_interface_assignment s1 embeddedsw.configuration.isFlash 1	

 For more information about the use of the set_interface_assignment command, refer to the *Publishing Component Information to Embedded Software* chapter in the *Nios II Software Developer's Handbook*.

Composition

This section covers the commands that allow you to build new components by combining other components. It also includes commands to query the module instances in the system.

add_instance

The add_instance command adds an instance of a predefined module, referred to as a *child* or *child module*, to a new component. You can use this command to create components that are composed of other components.

add_instance		
Callback availability	Main and composition	
Usage	add_instance <instanceName> <type> [<version>]	
Returns	String	
Arguments	instanceName	Specifies a unique local name that you can use to manipulate the module. This name is used in the generated HDL to identify the module.
	type	The type refers to a module available in a library, for example altera_avalon_uart.
	version	The required version of the specified module. If no version is specified, the latest version is used.
Example	add_instance my_uart altera_avalon_uart	

get_instances

This command lists the instance names of all modules in the system.

get_instances	
Callback availability	Main, elaboration, and composition
Usage	<code>get_instances</code>
Returns	List of strings
Arguments	None
Example	<code>get_instances</code>

get_instance_parameters

This command 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.

get_instance_parameters	
Callback availability	Main, elaboration, and composition
Usage	<code>get_instance_parameters <instanceName></code>
Returns	List of strings
Arguments	<code>instanceName</code> Specifies the name of the instance whose parameters are being retrieved.
Example	<code>get_instance_parameters pixel_converter</code>

set_instance_parameter_value

This command sets a parameter on a child module. Derived parameters and `SYSTEM_INFO` parameters for the child module may not be set using this command.

set_instance_parameter_value	
Callback availability	Main and composition
Usage	<code>set_instance_parameter_value <instanceName> <parameterName> <parameterValue></code>
Returns	None
Arguments	<code>instanceName</code> Specifies the name of the child module
	<code>parameterName</code> Specifies the parameter that is being set
	<code>parameterValue</code> Specifies the value of the parameter that is being set
Example	<code>set_instance_parameter_value pixel_converter input_DPI 1200</code>

get_instance_parameter_value

This command returns the value of the named parameter. You cannot use this command to get the value of parameters whose values are derived or those that are defined using the `SYSTEM_INFO` parameter property.

get_instance_parameter_value	
Callback availability	Main and composition
Usage	<code>get_instance_parameter_value <instanceName> <parameterName></code>
Returns	String, boolean, or units, depending on property. Refer to Table 9-4 on page 9-24
Arguments	<code>instanceName</code> Specifies the name of the instance whose parameter is being retrieved
	<code>parameterName</code> Specifies the parameter whose value is being retrieved
Example	<code>get_instance_parameter_value pixel_converter input_DPI</code>

get_instance_parameter_property

This command returns the names of the specified instance parameter property. The following parameter properties on a child instance that are visible from the parent: `TYPE`, `WIDTH`, `DERIVED`, `VISIBLE`, `ENABLED`, `UNITS`, `DISPLAY_NAME`, `ALLOWED_RANGES`, and `SYSTEM_INFO`.

get_instance_parameter_property	
Callback availability	Main and composition
Usage	<code>get_instance_parameter_property <instanceName> <parameterName> <propertyName></code>
Returns	String, boolean, or units, depending on property. Refer to Table 9-4 on page 9-24 .
Arguments	<code>instanceName</code> Specifies the instance name of the module
	<code>parameterName</code> Specifies the parameter for which a property is being retrieved
	<code>propertyName</code> Specifies the property whose value is being retrieved
Example	<code>get_instance_parameter_property my_stereo separate_control DISPLAY_NAME</code>

get_instance_interfaces

This command returns the names of all of the interfaces of a child module as a list. The interfaces can change if the parameterization of the module changes.

get_Instance_interfaces	
Callback availability	Main and composition
Usage	<code>get_instance_interfaces <instanceName></code>
Returns	String
Arguments	<code>instanceName</code> Specifies the instance name of the module
Example	<code>get_instance_interfaces my_ColorSpaceConverter</code>

get_instance_interface_properties

This command returns the names of all of the properties of the specified interface.

get_Instance_interface_properties	
Callback availability	Main and composition
Usage	<code>get_instance_interface_properties <instanceName> <interfaceName></code>
Returns	String
Arguments	<code>instanceName</code> Specifies the instance name of the module
	<code>interfaceName</code> Specifies an interface of instance
Example	<code>get_instance_interface_properties my_ColorSpaceConverter inputInterface</code>

get_instance_interface_property

This command returns the value of a property associated with the specified module interface.

get_Instance_interface_property	
Callback availability	Main and composition
Usage	<code>get_instance_interface_property <instanceName> <interfaceName> <propertyName></code>
Returns	String
Arguments	<code>instanceName</code> Specifies the instance name of the module
	<code>interfaceName</code> Specifies an interface of instance
	<code>propertyName</code> Specifies the property whose value is being retrieved.
Example	<code>get_instance_interface_property my_component s1 setupTime</code>

get_instance_interface_ports

This command returns a list of the names of the ports on the specified interface.

get_instance_interface_ports	
Callback availability	Main and composition
Usage	<code>get_instance_interface_ports <instanceName> <interfaceName></code>
Returns	List of Strings
Arguments	<code>instanceName</code> Specifies the instance name of the module
	<code>interfaceName</code> Specifies an interface of instance
Example	<code>get_instance_interface_ports my_ColorSpaceConverter outputInterface</code>

get_instance_port_property

This command returns a information about the port property specified.

get_instance_port_property	
Callback availability	Main and composition
Usage	<code>get_instance_port_property <instanceName> <portName> <propertyName></code>
Returns	String
Arguments	<code>instanceName</code> Specifies the instance name of the module
	<code>portName</code> Specifies a port
	<code>property</code> Specifies the property for which information is being retrieved. Not all port properties are visible from the parent. Those which are visible are ROLE, DIRECTION, WIDTH, WIDTH_EXPR and VHDL_TYPE.
Example	<code>get_instance_port_property my_uart width</code>

add_connection

This command connects the named interfaces together 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`. The command returns the name of the newly added connection in `start.point/end.point` format. Be careful to connect the start to the end, and not the other way around.

add_connection	
Callback availability	Main and composition
Usage	<code>add_connection <start.Interface> [<end.Interface>] [kind] [name]</code>
Returns	String
Arguments	<code>start.interface</code> The start interface to be connected, of the form, <code><instance_name>.<interface_name></code>
	<code>end.interface</code> The end interface to be connected, <code><instance_name>.<interface_name></code>
	<code>kind</code> Indicates the interface type. For a list of interface types refer to “ add_interface ” on page 9–35.
	<code>name</code> Specifies the name of the connection. If omitted, the name is of the form <code>start-module.start-interface/end-module.end-interface</code> .
Example	<code>add_connection dma.read_master sdram.s1</code>

get_connections

This command returns a list of all connections in the system if no element is specified. If an interface is specified, for example `cpu.instruction_master`, all connections to that interface are returned. If a component instance is specified, all connections to any interface on the instance is returned.

get_connections	
Callback availability	Main and composition
Usage	<code>get_connections [<interfaceName or instanceName>]</code>
Returns	List of strings
Arguments	None
Example	<code>get_connections cpu.instruction_master</code>

get_connection_parameters

This command gets the names of all parameters for the connection specified.

get_connection_parameters	
Callback availability	Main and composition
Usage	<code>get_connection_parameters <connectionName></code>
Returns	List of strings
Arguments	<code>connectionName</code> Specifies the connection whose connection parameters are required.
Example	<code>get_connection_parameters cpu0.data_master/dma0.csr</code>

get_connection_parameter_value

This command gets the value of a parameter on the connection.

get_connection_parameter_value	
Callback availability	Main and composition
Usage	<code>get_connection_parameters <connectionName> <parameterName></code>
Returns	String
Arguments	<code>connectionName</code> Specifies the connection whose connection parameters are required.
	<code>parameterName</code> The name of the parameter whose property value is being retrieved
Example	<code>get_connection_parameters cpu0.data_master/dma0.csr</code>

set_connection_parameter_value

This command sets a property of the connection. The start and end are each interface names of the format `<instance>.<interface>`. Connection parameters depend on the type of connection, for Avalon-MM they include base addresses and arbitration priorities.

set_connection_parameter_value	
Callback availability	Main program and composition
Usage	<code>set_connection_parameter_value <connName> <parameterName> <parameterValue></code>
Returns	None
Arguments	<code>connName</code> Specifies the name of the connection as returned by the <code>add_connection</code> command. It is of the form <code>start.point/end.point</code>
	<code>parameterName</code> Specifies the parameter that is being set
	<code>parameterValue</code> Specifies the value of the parameter
Example	<code>set_connection_parameter_value cpu0.data_master/dma0.csr baseAddress 0x1000</code>

Fileset Generation

This section covers the commands that create files to define the component and provide information to downstream tools.

add_fileset

This command adds a generation fileset for a particular target as specified by `<filesetKind>`. This target (`SIM_VHDL`, `SIM_VERILOG`, `QUARTUS_SYNTH`, or `EXAMPLE_DESIGN`) is called by Qsys when the specified generation target is requested. You may define multiple filesets for each kind of fileset. The specified callback procedure must have a single argument. The value of this argument is a generated name which must be used in the top-level module or entity declaration of your component. To override this generated name, you may set the fileset property `TOP_LEVEL`.



Overriding the generated name is only safe if all parameterizations of a core yield identical HDL.

add_fileset									
Callback availability	Main								
Usage	<code>add_fileset <filesetName> <filesetKind> <callbackProcName> [<displayName>]</code>								
Returns	String								
Arguments	<table border="1"> <tr> <td><code>filesetName</code></td> <td>The name of the fileset.</td> </tr> <tr> <td><code>filesetKind</code></td> <td>Files support the following kinds: <ul style="list-style-type: none"> ■ <code>SIM_VHDL</code> ■ <code>SIM_VERILOG</code> ■ <code>QUARTUS_SYNTH</code> ■ <code>EXAMPLE_DESIGN</code> </td> </tr> <tr> <td><code>callbackProcName</code></td> <td>A string identifying the name of the callback procedure.</td> </tr> <tr> <td><code>displayName</code></td> <td>A display string to identify the fileset.</td> </tr> </table>	<code>filesetName</code>	The name of the fileset.	<code>filesetKind</code>	Files support the following kinds: <ul style="list-style-type: none"> ■ <code>SIM_VHDL</code> ■ <code>SIM_VERILOG</code> ■ <code>QUARTUS_SYNTH</code> ■ <code>EXAMPLE_DESIGN</code> 	<code>callbackProcName</code>	A string identifying the name of the callback procedure.	<code>displayName</code>	A display string to identify the fileset.
	<code>filesetName</code>	The name of the fileset.							
	<code>filesetKind</code>	Files support the following kinds: <ul style="list-style-type: none"> ■ <code>SIM_VHDL</code> ■ <code>SIM_VERILOG</code> ■ <code>QUARTUS_SYNTH</code> ■ <code>EXAMPLE_DESIGN</code> 							
	<code>callbackProcName</code>	A string identifying the name of the callback procedure.							
<code>displayName</code>	A display string to identify the fileset.								
Example	<code>add_fileset PCIE_SYNTHESIS QUARTUS_SYNTH mySynthProc</code>								

add_fileset_file

This command adds a file in a particular location in the generation directory. You can specify source file locations using either an absolute path or a path that is relative to the component's `_hw.tcl` file.

add_fileset_file		
Callback availability	Fileset	
Usage	<code>add_fileset_file <fileDestination> <fileKind> <fileSource> <contentsOrPath></code>	
Returns	String	
Arguments	fileDestination	Specifies the location to store the file after Qsys generation.
	fileKind	Files support the following kinds: <ul style="list-style-type: none"> ■ VERILOG ■ SYSTEM_VERILOG ■ SYSTEM_VERILOG_INCLUDE ■ VHDL ■ SDC ■ MIF ■ HEX ■ DAT ■ OTHER
	fileSource	The following sources are defined: <ul style="list-style-type: none"> ■ PATH—specifies the original source file to be copied to filePath ■ TEXT—specifies an arbitrary text string for the contents of the file
	contentsOrPath	When the fileSource is PATH, specifies the file to be copied to filePath. When the fileSource is TEXT, specifies the text string to be stored in the file.
Example	<pre>add_fileset_file "./implementation/rx_pma.sv" SYSTEM_VERILOG PATH synth_rx_pma.sv add_fileset_file gui.sv SYSTEM_VERILOG TEXT "Customize your IP core"</pre>	

set_fileset_property

Allows a user to set the properties of a fileset.

set_fileset_property			
Callback availability	Fileset Generation		
Usage	<code>set_fileset_property <filesetName> <filesetProperty> <value></code>		
Returns	String		
Arguments	filename	The name of the fileset.	
	filesetProperty	TOP_LEVEL	The of the top-level HDL module produced by this fileset
Example	<code>set_fileset_property mySynthFileset TOP_LEVEL simple_uart</code>		

create_temp_file

This command creates a temporary file which can be manipulated inside the fileset callbacks of a `_hw.tcl` file. This temporary file can serve as a scratch pad or can be included in the generation output if it is included using the `add_fileset_file` command.

create_temp_file			
Callback availability	Fileset		
Usage	<code>create_temp_file <fileName></code>		
Returns	String		
Arguments	<table border="1"> <tr> <td>fileName</td> <td>The name of the created file.</td> </tr> </table>	fileName	The name of the created file.
fileName	The name of the created file.		
Example	<pre>set filelocation [create_temp_file "./hdl/compute_frequency.v" add_fileset_file compute_frequency.v VERILOG PATH \${filelocation}]</pre>		

Miscellaneous**check_device_family_equivalence**

This command returns a 1 or 0 indicating whether the device name returned in the string list is legal.

check_device_equivalence					
Callback availability	Main, elaboration, and fileset				
Usage	<code>check_device_family_equivalence <deviceName> <deviceList></code>				
Returns	<table border="1"> <tr> <td>1 or 0</td> <td>Based on equivalence.</td> </tr> </table>	1 or 0	Based on equivalence.		
1 or 0	Based on equivalence.				
Arguments	<table border="1"> <tr> <td>deviceName</td> <td>The device name of device being checked.</td> </tr> <tr> <td>deviceList</td> <td>The device string list to check against.</td> </tr> </table>	deviceName	The device name of device being checked.	deviceList	The device string list to check against.
deviceName	The device name of device being checked.				
deviceList	The device string list to check against.				
Example	<pre>check_device_equivalence CYLCONE III LS { "stratixv" "Cyclone IV" "cycloneiiiis" }</pre>				

get_device_family_displayname

This command returns the display name of a given device family.

get_device_family_displayname	
Callback availability	Main, elaboration, fileset
Usage	<code>get_device_family_displayname <deviceName></code>
Returns	A user-suitable display name for a device family
Arguments	A device family name (example stratixv, arriaii)
Example	<code>get_device_family_displayname cycloneiiiis (Returns: Cyclone III LS)</code>

set_qip_strings

This command places strings in the Quartus II IP File (**.qip**) file which are passed to the command as a TCL list. Successive calls to `set_qip_strings` are not additive, they replace the previously declared value.

set_qip_strings	
Callback availability	Main, elaboration
Usage	<code>set_qip_strings <qip Entries></code>
Returns	The TCL list which was set
Arguments	qip Entries A space-delimited TCL list.
Macros	<code>%entityName%</code> The generated name of the entity replaces this macro when the string is written to the .qip file
	<code>%libraryName%</code> The compilation library this component was compiled into is inserted in place of this marco inside the .qip file.
Example	<code>set_qip_strings {"QIP Entry 1" "QIP Entry 2"}</code>

get_qip_strings

This command returns a TCL list of QIP strings defined by the component's `_hw.tcl` file.

get_qip_strings	
Callback availability	Main, elaboration
Usage	<code>get_qip_strings</code>
Returns	A TCL list of qip string entries set by this component``
Example	<code>get_qip_strings</code>


Document Revision History

Table 9-8 shows the revision history for this document.

Table 9-8. Document Revision History

Date	Version	Changes
November 2011	11.1.0	<ul style="list-style-type: none"> ■ Template update. ■ Added commands <code>set_qip_strings</code>, <code>get_qip_strings</code>, <code>get_device_family_displayname</code>, <code>check_device_family_equivalence</code> ■ Updated text to reflect changes in 11.1.
May 2011	11.0.0	<ul style="list-style-type: none"> ■ Removed Beta status. ■ Revised section describing HDL and composed component implementations. ■ Separated reset and clock interfaces in examples. ■ Added the <code>TRISTATECONDUIT_INFO</code>, <code>GENERATION_ID</code>, and <code>UNIQUE_ID</code> <code>SYSTEM_INFO</code> properties. ■ Added the <code>WIDTH</code> and <code>SYSTEM_INFO_ARG</code> parameter properties. ■ Removed the <code>doc_type</code> argument from the <code>add_documentation_link</code> command. ■ Removed <code>get_instance_parameter_properties</code> command. (<code>get_instance_parameter_property</code> is available.) ■ Added the <code>add_fileset</code>, <code>add_fileset_file</code> and <code>create_temp_file</code> commands. ■ Updated Tcl examples to show separate clock and reset interfaces.
December 2010	10.1.0	Initial release.

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

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