10. Component Interface Tcl Reference

QII51023-13.1.0

This chapter contains descriptions of Tcl commands and indicates the Qsys phases during which each command is available: in the main static body of the program (global), or during the elaboration, composition, and fileset callback phases, or any combination. Table 10–1 summarizes the commands and provides a reference to the full description.

Qsys supports standard Avalon®, AMBA® AXI3™ (version 1.0), AMBA AXI4™ (version 2.0), and AMBA APB™ 3 (version 1.0) interfaces. For more information about Avalon and AMBA interfaces, refer to the *Avalon Interface Specifications* and the *AMBA Protocol Specifications* on the ARM® website. AXI4-Lite is not supported.

For more information about procedures for creating component **_hw.tcl** files in the Qsys Component Editor, and supported interface standards, refer to the *Creating Qsys Components* and the *Qsys Interconnect* chapters in volume 1 of the *Quartus II Handbook*.

If you are developing a component to work with the Nios II processor, refer to the *Publishing Component Information to Embedded Software* chapter, which describes how to publish hardware component information for embedded software tools, such as a C compiler and a Board Support Package (BSP) generator.

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

- "Command Summary"
- "Module Definition" on page 10–4
- "Parameters" on page 10–9
- "Display Items" on page 10–20
- "Interfaces and Ports" on page 10–24
- "Composition" on page 10–32
- "Fileset Generation" on page 10–40
- "Miscellaneous" on page 10–45
- "Simulator Properties" on page 10–46
- "Instance Properties" on page 10–47
- "Port Roles (Interface Signal Types)" on page 10–47

Qsys does not support the following Tcl commands in the global context of IP authoring for Qsys version 13.1:

- "add_instance"
- "add_connection"'
- "set_instance_parameter_value"

You can instead use a composition callback to add module instances and connections to your composed designs. For versions older than 13.1, a debug-level message is issued to inform developers about this limitation. Using these commands in a composition callback instead of in the global context significantly improves the performance of the validation and composition phases in composed systems with deep hierarchies.

Command Summary

Table 10–1. Command Summary (1) (Part 1 of 3)

Command	Full Description
Module Definition	
add_documentation_link *<title> <fileOrUrl>*	page 10–4
get_module_assignment *<moduleName>*	page 10–7
get_module_assignments	page 10–5
get_module_ports	page 10–5
get_module_properties	page 10–6
get_module_property *<propertyName>*	page 10–6
package <require> -exact qsys *<version>*	page 10–6
send_message *<messageLevel> <messageText>*	page 10–7
set_module_assignment *<moduleName> [value]*	page 10–7
set_module_property *<propertyName> <propertyValue>*	page 10–8
Parameters	
add_parameter *<parameterName> <parameterType> [<defaultValue> <description>]*	page 10–9
decode_address_map *<address_map_XML_string>*	page 10–10
get_parameters	page 10–11
get_parameter_properties	page 10–12
get_parameter_property *<parameterName> <propertyName>*	page 10–12
get_parameter_value *<parameterName>*	page 10–12
get_string *<identifier>*	page 10–13
load_strings *<fileName>*	page 10–13
set_parameter_property *<parameterName> <propertyName> <value>*	page 10–12
set_parameter_value *<parameterName> <value>*	page 10–12
Display Items	
add_display_item *<groupName> <id> <type> [<additionalInfo>]*	page 10–20
get_display_items	page 10–22

Table 10–1. Command Summary (1) (Part 2 of 3)

Command	Full Description
get_display_item_properties	page 10–22
get_display_item_property <itemName> <propertyName>	page 10–22
set_display_item_property <itemName> <propertyName> <value>	page 10–23
Interfaces and Ports	
add_interface <interfaceName> <interfaceType> <direction> [<associatedClock>]	page 10–24
add_interface_port <interfaceName> <portName> <portRole> [<direction> <width_expr>]	page 10–25
get_interfaces <interfaceName>	page 10–25
get_interface_assignment <interfaceName> <name>	page 10–26
get_interface_assignments	page 10–26
get_interface_ports [<interfaceName>]	page 10–26
get_interface_property <interfaceName> <propertyName>	page 10–27
get_port_properties	page 10–28
get_port_property <portName> <propertyName>	page 10–28
set_interface_assignmet <interfaceName> <name> [<value>]	page 10–26
set_interface_property <interfaceName> <propertyName> <value>	page 10–29
set_port_property <portName> <propertyName> [<value>]	page 10–30
Composition	
add_connection <startInterface> <endInterface>[<type>]	page 10–32
get_connections	page 10–33
get_connection_parameters <instanceName>	page 10–33
get_connection_parameter <connectionName> <parameterName>	page 10–33
add_instance <instanceName> <instanceType> <version>	page 10–32
get_instance_interfaces <instanceName>	page 10–34
get_instance_interface_ports <instanceName> <portName>	page 10–34
get_instance_interface_properties <instanceName> <interfaceName>	page 10–34
get_instance_property <instance> <property>	page 10–35
set_instance_property <instance> <property> <value>	page 10–35
get_instance_properties	page 10–36
get_instance_interface_property <instanceName> <interfaceName> <propertyName>	page 10–35
get_instances	page 10–34
get_instance_parameters <instanceName>	page 10–36
get_instance_parameter_value <instanceName> <parameterName>	page 10–38
get_instance_parameter_properties <instanceName> <parameterName>	page 10–34
get_instance_parameter_property <instanceName> <parameterName> <propertyName>	page 10–37
get_instance_port_property <instanceName> <interfaceName> <propertyName>	page 10–39

Table 10–1. Command Summary (1) (Part 3 of 3)

Command	Full Description
set_connection_parameter_value <*connectionName*> <*parameterName*> <*parameterValue*>	page 10–39
set_instance_parameter_value <*instanceName*> <*parameterName*> <*parameterValue*>	page 10–40
Fileset Generation	
add_fileset <*filesetName*> <*filesetKind*> <*callbackProcName*> [<*displayName*>]	page 10–40
add_fileset_file <*fileDestination*> <*fileKind*> <*fileSource*> <*contentsOrPath*> [<*attributes*>]	page 10–42
set_fileset_property <*fileset*> <*property*> <*value*>	page 10–42
get_fileset_file_attribute <*output_file*> <*attribute*>	page 10–43
set_fileset_file_attribute <*output_file*> <*attribute*> <*value*>	page 10–44
get_fileset_sim_properties <*fileset*> <*platform*> <*property*>	page 10–44
set_fileset_sim_properties <*fileset*> <*platform*> <*property*> <*value*>	page 10–44
create_temp_file <*fileName*>	page 10–45
Miscellaneous	page 10–45
check_device_family_equivalence <*device_family*> <*device_family_list*>	page 10–45
get_device_family_displayname <*device_family*>	page 10–45
set_qip_strings <*qip_strings*>	page 10–46

Note to Table 10–1:

(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.

add_documentation_link

This command allows you to link to documentation for your component.

add_documentation_link		
Callback availability	global	
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	

get_module_assignment

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

get_module_assignment		
Callback availability	global, elaboration, composition	
Usage	`get_module_assignment <name>`	
Returns	`string`	
Arguments	`name`	The name of the assignment whose value is being retrieved
Example	`get_module_assignment embeddedsw.CMacro.colorSpace`	

get_module_assignments

This command returns names of the module assignments.

get_module_assignments	
Callback availability	global, elaboration, composition
Usage	`get_module_assignments`
Returns	`string`
Arguments	None
Example	`get_module_assignments`

get_module_ports

This command returns a list of the names of all the ports that are currently defined.

get_module_ports	
Callback availability	global, elaboration, generation
Usage	`get_module_ports`
Returns	`string`
Example	`get_module_ports`

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	global, elaboration, generation, composition
Usage	`get_module_properties`
Returns	`list of strings`
Arguments	None
Example	`get_module_properties`

get_module_property

This command returns the value of a single module property.

get_module_property		
Callback availability	global, elaboration, fileset, composition	
Usage	`get_module_property <propertyName>`	
Returns	`string`, `boolean`, `file`	
Arguments	`propertyName`	"Module Properties"
Example	`set my_name [get_module_property NAME]`	

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. The package determines which version of the **_hw.tcl** API to use for this component.

This document describes the behavior of components which start with `package require -exact qsys 13.1` For earlier versions of the **_hw.tcl** commands, refer to the documentation for that release. If you do not request a package, you get **_hw.tcl** commands compatible with Quartus II version 9.0.

package		
Callback availability	global (before any other commands in the file)	
Usage	`package require -exact qsys <version>`	
Returns	None	
Arguments	`version`	The version of Qsys that you require, such as 13.1.
Example	`package require -exact qsys 13.1`	

send_message

This command sends a message to the user of the component. The message text is normally interpreted as HTML. The <b> element can be used to provide emphasis. If you do not want the message text to be interpreted as HTML, then pass a list such as {info text} as the message level.

send_message		
Callback availability	global, elaboration, fileset, composition	
Usage	send_message <messageLevel> <messageText>	
Returns	None	
Arguments	messageLevel	The following message levels are supported: ■ Error—Provides an error message. The Qsys system cannot be generated with existing 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.

set_module_assignment

This command sets the value of the specified assignment.

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

set_module_property

This command allows you to set the values for module properties.

set_module_property		
Callback availability	global	
Usage	`set_module_property <propertyName> <propertyValue>`	
Returns	None	
Arguments	`propertyName`	"Module Properties"
	`propertyValue`	The new value of the property.
Example	`set_module_property VERSION 13.1`	

add_hdl_instance

This command adds an instance of a predefined module, referred to as a child or child instance, and returns the fixed entity name of the added instance. The HDL entity generated from this instance can be instantiated and connected within this component's HDL.

add_hdl_instance		
Callback availability	global, elaboration	
Usage	add_hdl_instance <*entity_name*> <*ip_core_type*> [<*version*>]	
Returns	`string` - The entity name of the added instance.	
Arguments	`entity_name`	Specifies a unique local name that you can use to manipulate the instance. This name is used in the generated HDL to identify the instance.
	`ip_core_type`	The type refers to a kind of instance available in a library, for example altera_avalon_uart.
	`version`	Optional. The required version of the specified instance type. If no version is specified, the latest version is used.
Examples	`add_hdl_instance my_uart altera_avalon_uart`	

Module Properties Table

Table 10–2 lists the available module properties, their use, and the phases in which they can be set.

Table 10–2. Module Properties (Part 1 of 2)

Property Name	Property Type	Can Be Set	Description
`AUTHOR`	`string`	global	The module's author.
`COMPOSITION_CALLBACK`	`string`	`string`	The name of the composition callback. If you define a composition callback, you cannot define the generation or elaboration callbacks.
`DESCRIPTION`	`string`	global	The description of the module, such as "Example Qsys Module."

Table 10–2. Module Properties (Part 2 of 2)

Property Name	Property Type	Can Be Set	Description
DISPLAY_NAME	string	global	The name to display when referencing the module, such as “My Component.”
EDITABLE	boolean	global	Indicates whether you can edit the component in the Component Editor.
ELABORATION_CALLBACK	string	global	The name of the elaboration callback.When set, the component's elaboration callback is called to validate and elaborate interfaces for instances of the component.
GROUP	string	global	The group in the Component Library that includes this component.
ICON_PATH	string	global	A path to an icon to display in the module’s parameter editor.
INTERNAL	boolean	global	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	global	The name of the module, such as my_component.
OPAQUE_ADDRESS_MAP	string	global	For composed components created using a **_hw.tcl** file that include children that are memory-mapped slaves; specifies whether the children’s addresses are visible to downstream software tools. When true, the children’s address are not visible. When false, the children’s addresses are visible.
VERSION	string	global	The module’s version, such as 13.1.

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 found 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 add a parameter of type string_list or integer_list, the parameter is displayed in a single-column table that can add or remove items in the list.

- If you add multiple parameters of type `string_list` or `integer_list` to the same group, and add the group with a `TABLE` hint, the entire group is displayed as a multi-column table. Example 10–1 illustrates the use of the `integer_list` parameter types to create a multi-column table.

Example 10–1. Creating Tables Using the string_list and integer_list Parameter Types

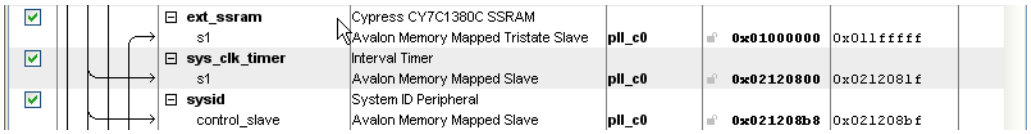
```
add_parameter names STRING_LIST
add_parameter counts INTEGER_LIST

add_display_item "" myTable GROUP TABLE
add_display_item myTable names PARAMETER
add_display_item myTable counts PARAMETER
```

add_parameter		
Callback availability	global	
Usage	`add_parameter <parameterName> <parameterType> [<defaultValue> <description>]`	
Returns	`string`	
Arguments	`parameterName`	A name that you, the component author, choose for your parameter.
	`parameterType`	The following types are supported: `integer`, `natural`, `positive`, `boolean`, `float`, `long`, `std_logic`, `std_logic_vector`, `string`, `string_list`, and `integer_list`.
	`defaultValue`	The value for this parameter if the parameter's value is never explicitly set.
	`description`	Explains the use of the parameter.
Example	`add_parameter seed integer 17 "The seed to use for data generation."`	

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 + l. Figure 10–1 shows a portion of a Qsys system with three slave devices.

Figure 10–1. Qsys System with Three Avalon-MM Slaves



Example 10–2 shows the XML code that describes the address map for the 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 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 10–2 to enumerate over the components within an address map, rather than writing your own parser.

Example 10–2. Address Map for 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, composition	
Usage	`decode_address_map <address_map_XML_string>`	
Returns	List of Tcl lists, each one suitable for passing to array set	
Arguments	`address_map_XML_string`	An XML string describing the address map of a master.

Example	`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)"`
	`}`	

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	global, elaboration, fileset, composition
Usage	`get_parameters`
Returns	`list of strings`
Arguments	None
Example	`set parameter_summary [get_parameters]`

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	global, elaboration, fileset, composition	
Usage	`get_parameter_properties`	
Returns	`list of strings`	
Arguments	None	
Example	`set property_summary [get_parameter_properties]`	

get_parameter_property

This command returns a single parameter property.

get_parameter_property		
Callback availability	global, elaboration, fileset, composition	
Usage	`get_parameter_property <parameterName> <propertyName>`	
Returns	`string`, `boolean`, or `units`, depending on property value.	
Arguments	`parameterName`	The name of the parameter whose property value is being retrieved.
	`propertyName`	"Parameter Properties"
Example	`get_parameter_property parameter1 GROUP`	

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, 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.

get_string

This command returns the value of an externalized string previously loaded by the `load_strings` command.

Example 10–3. get_string

```
package require -exact qsys <version>

load_strings test.properties

set_module_property NAME test
set_module_property VERSION [get_string VERSION]
set_module_property DISPLAY_NAME [get_string DISPLAY_NAME]

add_parameter firepower INTEGER 0 ""
set_parameter_property firepower DISPLAY_NAME [get_string PARAM_DISPLAY_NAME]
set_parameter_property firepower TYPE INTEGER
set_parameter_property firepower DESCRIPTION [get_string PARAM_DESCRIPTION]

DISPLAY_NAME = Trogdor!
VERSION = 1.0
PARAM_DISPLAY_NAME = Firepower
PARAM_DESCRIPTION = The amount of force to use when breathing fire.
```

get_string	
Availability	any
Usage	get_string <*identifier*>
Returns	string
Arguments	identifier
Example	get_string MY_STRING

load_strings

This command loads strings from an external **.properties** file. The format of the properties file is in the *Java Properties File* format.

Example 10–4. load_strings

```
package require -exact qsys 12.1

load_strings test.properties

set_module_property NAME test
set_module_property VERSION [get_string VERSION]
set_module_property DISPLAY_NAME [get_string DISPLAY_NAME]

add_parameter firepower INTEGER 0 ""
set_parameter_property firepower DISPLAY_NAME [get_string PARAM_DISPLAY_NAME]
set_parameter_property firepower TYPE INTEGER
set_parameter_property firepower DESCRIPTION [get_string PARAM_DESCRIPTION]

DISPLAY_NAME = Trogdor!
VERSION = 1.0
PARAM_DISPLAY_NAME = Firepower
PARAM_DESCRIPTION = The amount of force to use when breathing fire.
```

load_strings	
availability	any
Usage	`load_strings <path>`
Returns	none
Arguments	`path`
Example	`load_strings_my_externalized_strings.properties`

set_parameter_property

This command sets a single parameter property.

set_parameter_property		
Callback availability	global, elaboration, 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`	"Parameter Properties"
	`value`	The new value for the property.
Example	`set_parameter_property BAUD_RATE ALLOWED_RANGES {9600 19200 38400}`	

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, 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`	

Parameter Properties Table

Table 10–3 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 10–3. Parameter Properties (Part 1 of 3)

Property Name	Type/ Default	Can Be Set	Description
AFFECTS_ELABORATION	boolean, true	global	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	global	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.
AFFECTS_VALIDATION	boolean, refer to description	global	The AFFECTS_VALIDATION property marks whether a parameter's value is used to set derived parameters, and whether the value affects validation messages. When set to false, this may improve response time in the parameter editor UI when the value is changed. The default value is true.
ALLOWED_RANGES	string,""	global	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, such as {0:None 1:Monophonic 2:Stereo 4:Quadrophonic} meaning 0,1,2,4 are the legal value, and **None**, **Monophonic**, **Stereo**, and **Quadrophonic** are the values shown in the GUI.
DEFAULT_VALUE	string, boolean	global	The default value.
DERIVED	boolean, false	elaboration callback	When true, indicates that this parameter's value is computed by the component during elaboration or composition callback. The default value is false.

Table 10–3. Parameter Properties (Part 2 of 3)

Property Name	Type/ Default	Can Be Set	Description
DESCRIPTION LONG_DESCRIPTION	string, ""	global	DESCRIPTION—A tooltip description of the parameter that appears in the parameter editor. LONG_DESCRIPTION—A description of the parameter that appears in a popup dialog box when you click **Documentation** in the parameter editor. If LONG_DESCRIPTION is not set, then DESCRIPTION appears in the documentation window.
DISPLAY_NAME	string,""	global	The GUI label that identifies the parameter.
DISPLAY_UNITS	string, ""	global	The GUI label that describes the units that the parameter's value represents, such as "Megabytes".
ENABLED	boolean, true	global elaboration callback	When false, the parameter is disabled, meaning that it is displayed, but greyed out, indicating that it is not editable on the parameter editor.
HDL_PARAMETER	boolean,false	global	When true, the parameter must be passed to the HDL component description. The default value is false.
NEW_INSTANCE_VALUE	string, ""	global	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 defaultValue argument. The practical result is that older instances of the component will continue to use defaultValue for the parameter and newer instances can use the value assigned by NEW_INSTANCE_VALUE.
SYSTEM_INFO	string, ""	global	Allows you to assign information about the instantiating system to a parameter. A SYSTEM_INFO property requires an argument specifying the type of information requested, *<info-type>*. *<info-type>* may also take an argument. The syntax of the Tcl command is: `set_parameter_property my_parameter SYSTEM_INFO <info-type> [<arg>]` Refer to "SYSTEM_INFO Types" on page 10–18 for descriptions of the *<info_type>* argument. You can use the SYSTEM_INFO property to set the SYSTEM_INFO_TYPE and SYSTEM_INFO_ARG properties at the same time.
SYSTEM_INFO_ARG	string, ""	global	Defines an argument to be passed to a particular SYSTEM_INFO function.
SYSTEM_INFO_TYPE	various	global	Specifies one of the types of information listed in "SYSTEM_INFO Types" on page 10–18.
TYPE	string, ""	global	Specifies one of the following types: INTEGER, NATURAL, POSITIVE, BOOLEAN, STD_LOGIC, STD_LOGIC_VECTOR, STRING, STRING_LIST, INTEGER_LIST, LONG, or FLOAT.

Table 10–3. Parameter Properties (Part 3 of 3)

Property Name	Type/ Default	Can Be Set	Description
UNITS	string, ""	global	Sets the units of the parameter. The following values are possible: ■ NONE ■ ADDRESS ■ BITS ■ BITSPERSECOND ■ BYTES ■ CYCLES ■ GIGABYTES ■ GIGABITSPERSECOND ■ GIGAHERTZ ■ HERTZ ■ KILOBYTES ■ KILOHERTZ ■ KILOBITSPERSECOND ■ MEGABYTES ■ MEGABITSPERSECOND ■ MEGAHERTZ ■ MICROSECONDS ■ MILLISECONDS ■ NANOSECONDS ■ PERCENT ■ PICOSECONFDS ■ SECONDS For example, `set_parameter_property frequency UNITS GIGAHERTZ`. ☞ You can use `DISPLAY_UNITS` to define an alternate `UNIT` description.
VISIBLE	boolean, true	global elaboration callback	Indicates whether or not to display the parameter in the parameterization GUI.
WIDTH	string, ""	global	For a `STD_LOGIC_VECTOR` parameter, this indicates the width of the logic vector.

SYSTEM_INFO Properties Table

Table 10–4 describes the properties that you can use with the SYSTEM_INFO parameter property.

Table 10–4. SYSTEM_INFO Types (Part 1 of 2)

SYSTEM_INFO Type	Type of Parameter	Description
ADDRESS_MAP	string	Assigns an XML-formatted string describing the address map to the parameter you specify. set_parameter_property <my_parameter> SYSTEM_INFO {ADDRESS_MAP <my_avalon-mm_master>}
ADDRESS_WIDTH	integer	Assigns an integer to the parameter you specify that is the number of bits an master must drive to address all of its slaves, using byte addresses. set_parameter_property <my_parameter> SYSTEM_INFO {ADDRESS_WIDTH <my_avalon-mm_master>}
AVALON_SPEC	string	The version of the interconnect. Qsys interconnect uses Avalon Specification 2.0.
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 CLOCK_DOMAIN value is guaranteed to be the same and greater than zero. set_parameter_property <my_parameter> SYSTEM_INFO {CLOCK_DOMAIN <my_clk>}
CLOCK_RATE	integer, 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. set_parameter_property <my_parameter> SYSTEM_INFO {CLOCK_RATE <my_clk>}
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.
DESIGN_ENVIROMNMENT	string	A string that identifies the current design environment, NATIVE or QSYS.
DEVICE	string	The device part number of the currently selected device.
DEVICE_FAMILY	string	Assigns the family name (not the specific device part number) of the currently selected device to the parameter you specify. set_parameter_property <my_parameter> SYSTEM_INFO {DEVICE_FAMILY}
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 set command. This list is assigned to the parameter you specify. The following features are supported: M512_MEMORY, M4K_MEMORY, M9K_MEMORY, M144K_MEMORY, MRAM_MEMORY, MLAB_MEMORY, ESB, DSP, and EMUL. set_parameter_property <my_parameter> SYSTEM_INFO {DEVICE_FEATURES}
DEVICE_SPEEDGRADE	string	The speed grade of the currently selected device.

Table 10–4. SYSTEM_INFO Types (Part 2 of 2)

SYSTEM_INFO Type	Type of Parameter	Description
GENERATION_ID	integer	An integer that stores a hash of the generation time that Qsys uses as a unique ID for a generation run.
INTERRUPTS_USED	integer, 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. set_parameter_property <*my_parameter*> SYSTEM_INFO {INTERRUPTS_USED <*my_interrupt_receiver*>}
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 master. set_parameter_property <*my_parameter*> SYSTEM_INFO {MAX_SLAVE_DATA_WIDTH <*my_avalon_mm_master*>}
QUARTUS_INI	string boolean integer	The value of the **quartus.ini** setting specified in the system info argument.
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. set_parameter_property <*my_parameter*> SYSTEM_INFO {RESET_DOMAIN <*my_reset*>}
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: ■ The Avalon-TC slave interface name ■ The Avalon-TC master module and interface names ■ The Avalon-TC signal names, directions, and widths 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: 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" To retrieve information about all the slave interfaces on a module substitute "*" for the interface name, as the following example illustrates: set_parameter_property TC_slave_info SYSTEM_INFO_ARG "*"
UNIQUE_ID	string	A string guaranteed to be unique to this module.

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 "Creating Tables Using the string_list and integer_list Parameter Types" on page 10–10 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	global	
Usage	`add_display_item <groupName> <id> <type> [<additionalInfo>]`	
Returns	`string`	
Arguments	`groupName`	Specifies the group to which a display item belongs.
	`id`	Specifies the parameter or icon to be displayed in a group. Each display item associated with a component must have a different ID.
	type	Specifies the category of the display item. The following types are defined: ■ `icon`–a **.gif**, **.jpg**, or **.png** file ■ `parameter`–a parameter in the instance ■ `text`–a block of text ■ `group`–a group. If the *`groupName`* is also defined, the new group is a child of the *`groupName`* group. If *`groupName`* is an empty string, the group is top-level. ■ `action`–an action defined by a callback procedure when you click the button labeled by *`actionName`*.
	additionalInfo	Provides extra information required for display items. The following examples illustrate how you use the `additionalInfo` argument for the various types: ■ `add_display_item` *`groupName id`* `icon` *`path-to-image-file`* ■ `add_display_item` *`groupName parameterName`* `parameter` (`additionalInfo` not required) ■ `add_display_item` *`groupName id`* `text "your-text"` The `your-text` argument is a block of text that is displayed in the GUI. Some simple HTML formatting is allowed, such as `<b>` and `<i>`, if the text starts with `"<html>"`. ■ `add_display_item` *`parentGroupName childGroupName`* `group [tab]` The `tab` is an optional parameter. If present, the group appears in separate tab in the GUI for the instance. ■ `add_display_item` *`parentGroupName actionName`* `action buttonClickCallbackProc`
Examples	`add_display_item timing read_latency parameter` `add_display_item sound speaker icon speaker.jpg`	

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	global, elaboration, fileset, composition	
Usage	`get_display_items`	
Returns	`list of strings`	
Arguments	`None`	
Example	`get_display_items`	

get_display_item_properties

This command returns a list of properties that can be set on display items.

get_display_item_properties		
Callback availability	global	
Usage	`get_display_item_properties`	
Returns	`list of strings`	
Arguments	`None`	
Example	`get_display_item_properties`	

get_display_item_property

This command returns the value of a property that can be set on a display item.

get_display_item_property		
Callback availability	global	
Usage	`get_display_item_property` *<itemName> <propertyName>*	
Returns	`string`	
Arguments	`itemName`	The name of the display item whose property value is being retrieved.
	`propertyName`	The property whose value is being retrieved.
Example	`set my_label [get_display_item_property my_action DISPLAY_NAME]`	

set_display_item_property

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

set_display_item_property		
Callback availability	global	
Usage	`set_display_item_property <itemName> <propertyName> <value>`	
Returns	`string`	
Arguments	`itemName`	The name of the display item whose property value is being set.
	`propertyName`	The property whose value is being set.
	`value`	The value to set.
Example	`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 procedure in the hw.tcl file"`	

Display Item Properties Table

Table 10–5. Display Items Properties (Part 1 of 2)

Property Name	Type	Default	Can Be Set	Description
`DESCRIPTION`	`string`		global	For an ACTION display item, updates the description /tooltip for the action button.
`DISPLAY_HINT`	`string,""`		global	Provides a hint about how to display a parameter. The following values are possible: ■ `boolean`—for integer parameters whose value can be 0 or 1. The parameter displays as an option that you can turn on or off. ■ `radio`—displays a parameter with a list of values as radio buttons instead of a drop-down list. ■ `hexadecimal`—for integer parameters, display and interpret the value as a hexadecimal number, for example: 0x00000010 instead of 16. ■ `fixed_size`—if the parameter is displayed in a table, the `fixed_size` `DISPLAY_HINT` eliminates the **add** and **remove** buttons from tables.
`ENABLED`	`boolean`	true	global elaboration callback	You can be use this property to enable or disable PARAMETER, GROUP, and ACTION display items. For a GROUP display item, this enables or disables all items in the group.
`GROUP`	`string, ""`		global	Controls the grouping of parameters in GUI.
`PATH`	`string`		global	For an ICON display item, updates path to the file for the displayed graphics, and allows the image to be switched dynamically.

Table 10–5. Display Items Properties (Part 2 of 2)

Property Name	Type	Default	Can Be Set	Description
TEXT	string		global elaboration callback	For a TEXT display item, updates the displayed text to some new text. You can use a TEXT display item as a replacement for derived parameters to display text to users; provides a cleaner UI and reduces clutter when authoring parameters.
VISIBLE	boolean	true	global elaboration callback	Makes PARAMETER, ICON, GROUP, TEXT, and ACTION display items visible or hidden. For a GROUP display item, shows or hides all items in the group.

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. Active high signals are terminated to 0, and active low signals are terminated 1.

add_interface (Part 1 of 2)

Callback availability	global, elaboration callback, elaboration, composition	
Usage	add_interface *<interfaceName> <interfaceType> <direction>*	
Returns	string	
Arguments	interfaceName	A name that you choose to identify an interface.
	interfaceType and direction	There are seven interfaceTypes. The following directions are possible for these interfaceTypes

Interface Type	Direction
avalon	master, slave (1)
axi	master, slave
tristate_conduit	master, slave
avalon_streaming	source, sink
interrupt	sender, receiver
conduit	end
clock	source, sink
reset	source,sink
nios_custom_instruction	slave

add_interface (Part 2 of 2)	
Example	`add_interface mm_slave avalon slave`

Notes:

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

add_interface_port

This command adds a port to an interface on your module. The name must match the name of a signal on the top-level module in the HDL of your component. 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 variable-width expression can be set globally.
- A constant width can be set globally, and updated in the elaboration callback.

add_interface_port		
Callback availability	global, 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 matches a signal name on the top-level module in the component's HDL files.
	`portRole`	The role of this port within the interface. Port roles are referred to as `signal types` in the *Avalon Interface Specification*. Refer to the *Avalon Interface Specifications* 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_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	global, elaboration, generation, composition
Usage	`get_interfaces`
Returns	`list of strings`
Arguments	None
Example	`set all_interfaces [get_interfaces]`

get_interface_assignment

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

get_interface_assignment		
Callback availability	global, elaboration, composition	
Usage	`get_interface_assignments <interfaceName> <name>`	
Returns	`string`	
Arguments	`interfaceName`	The name of the interface whose assignment is being retrieved.
	`name`	The assignment whose value is being retrieved.
Example	`get_interface_assignment s1 embeddedsw.configuration.isFlash`	

get_interface_assignments

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

get_interface_assignments		
Callback availability	global, elaboration, composition	
Usage	`get_interface_assignments <interfaceName>`	
Returns	`string`	
Arguments	`interfaceName`	The name of the interface whose assignment is being retrieved.
Example	`get_interface_assignments s1`	

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	global, elaboration, fileset	
Usage	`get_interface_ports [<interfaceName>]`	
Returns	`string`	
Arguments	`interfaceName`	The name of the interface whose ports you want to list (optional).
Example	`get_interface_ports mm_slave`	

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	global, elaborations, 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 "Interface Properties" on page 10–30.

get_interface_property

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

get_interface_property		
Callback availability	global, composition, elaboration	
Usage	`get_interface_property <interfaceName> <propertyName>`	
Returns	`string`, `boolean`, or `units`, depending on property. Refer to "Interface Properties" on page 10–30 and the *Avalon Interface Specifications* for more information.	
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.
Example	`get_interface_property mm_slave readWaitTime`	

get_port_properties

This command returns a list of all available port properties.

get_port_properties	
Callback availability	global, elaboration, fileset, composition
Usage	`get_port_properties <portName>`
Returns	`string`, `boolean`, or `units`, depending on property.
Arguments	`portName` The name of the port whose properties are required. "Port Properties"
Example	`get_port_properties mm_slave`

get_port_property

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

get_port_property		
Callback availability	global, elaboration, fileset	
Usage	`get_port_property <portName> <propertyName>`	
Returns	Depends on the type of the property	
Arguments	`portName`	The name of the port.
	`propertyName`	"Port Properties"
Example	`get_port_property rdata WIDTH_VALUE`	

set_interface_assignment

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

set_interface_assignment		
Callback availability	global, elaboration, composition	
Usage	`set_interface_assignment <interfaceName> <name> [<value>]`	
Returns	None	
Arguments	`interfaceName`	The name of the 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*.

set_interface_property

This command sets a single interface property for an interface.

set_interface_property		
Callback availability	global, compose, 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 ("Interface Properties").
	`value`	The value to set for the specified property.
Example	`set_interface_property mm_slave linewrapBursts false`	

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.

set_port_property

This command sets a single port property.

set_port_property		
Callback availability	global, program, elaboration	
Usage	`set_port_property <portName> <propertyName> [<value>]`	
Returns	`string`, `boolean`, or `units`, depending on property.	
Arguments	`portName`	The name of the port.
	`propertyName`	"Port Properties"
	`value`	The value to set.
Example	`set_port_property rdata WIDTH_EXPR 32`	

Interface Properties Table

Table 10–6. Interface Properties

Property	Type	Description
`EXPORT_OF`	`string`	For composed **_hwl.tcl** files, the `EXPORT_OF` 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 `add_interface`. The interface to be exported is of the form *<instanceName.interfaceName>*. Example: `set_interface_property CSC_input EXPORT_OF my_colorSpaceConverter.input_port`.
`ENABLED`	`boolean`	Specifies whether or not interface is enabled.
`SVD_ADDRESS_GROUP <int>`	`integer`	<*int*> is the number of the address group. If two masters with the same `SVD_ADDRESS_GROUP` group number and one of the masters has a System View Description (**.svd**) file associated with it, then all the slaves connected to the second master are added to the merged **.svd** file of the first master. Refer to "set_interface_property" on page 10–29.
`SVD_ADDRESS_OFFSET`	`integer`	Slaves connected to the master of name <*interface_name*> are adjusted by the <*offset*> number.Refer to "set_interface_property" on page 10–29.
`CMSIS_SVD_FILE <path_to_svd_file>`	`string`	The path to the **.svd** file.CMSIS_SVD_VARIABLES <list of variables and values>. Type: list. Variable substitutions used when writing out the .svd contents.
`CMSIS_SVD_VARIABLES <list of variables and values>`	`list`	Qsys uses Variable substitutions when writing the **.svd** contents.

Port Properties Table

Table 10–7. Port Properties

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.
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 *Avalon Interface Specifications*.
FRAGMENT_LIST	string	Specifies a split or join from the component definition in the Qsys component library and the instantiation in a Qsys system. For example, if the library component has two output ports, data1[7:0] and data2[7:0], which the instantiation joins to create a single output port, data1_2[7:0], the fragment_list defines data1_2[7:0].fragment_list = {data1[7:0],data2[7:0]} using the following command: set_port_property data_1_2 FRAGMENT_LIST { data1[7:0], data2[7:0]}. If the library component has a vectored signal, bus[3:0], that is split into 4 separate ports in the instantiation, bus_0 .. bus_3, then the following commands specify the separate ports: set_port_property bus_0 FRAGMENT_LIST { bus@0 } set_port_property bus_1 FRAGMENT_LIST { bus@1 } set_port_property bus_2 FRAGMENT_LIST { bus@2 } set_port_property bus_3 FRAGMENT_LIST { bus@3 }

Composition

This section describes the commands that allow you to build a component by combining instances of other components. It also includes commands to query the child instances in the component.

add_instance

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

add_instance		
Callback availability	composition (refer to limitation on page 10–2)	
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 the component 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`	

add_connection

This command connects the named interfaces on child instances 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	composition (refer to limitation on page 10–2)	
Usage	`add_connection <start.Interface> [<end.Interface>] [kind] [name]`	
Returns	`string`	
Arguments	`start.interface`	The start interface to be connected, of the form, `<instance_name>.<interface_name>`
	`end.interface`	The end interface to be connected, `<instance_name>.<interface_name>`
	`kind`	Indicates the interface type. For a list of interface types refer to "add_interface" on page 10–24.
	`name`	Specifies the name of the connection. If omitted, the name is of the form `start-module.start-interface/end-module.end-interface`.
Example	`add_connection dma.read_master sdram.s1`	

get_connections

This command returns a list of connections. If no argument is specified, all connections in the component are returned. If a child instance is specified, all connections to interfaces on the instance are returned. If an interface on a child instance is specified, only connections to that interface are returned.

get_connections		
Callback availability	global, composition	
Usage	`get_connections [<interfaceName or instanceName>]`	
Returns	`list of strings`	
Arguments	None	
Example	`get_connections cpu.instruction_master`	

get_connection_parameters

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

get_connection_parameters		
Callback availability	global, composition	
Usage	`get_connection_parameters <connectionName>`	
Returns	`list of strings`	
Arguments	`connectionName`	Specifies the connection whose connection parameters are required.
Example	`get_connection_parameters cpu0.data_master/dma0.csr`	

get_connection_parameter_value

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

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

get_instances

This command lists the instance names of all child modules in the component.

get_instances	
Callback availability	global, elaboration, composition
Usage	`get_instances`
Returns	`list of strings`
Arguments	None
Example	`get_instances`

get_instance_interfaces

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

get_Instance_interfaces		
Callback availability	global, composition	
Usage	`get_instance_interfaces <instanceName>`	
Returns	`string`	
Arguments	`instanceName`	Specifies the instance name of the module.
Example	`get_instance_interfaces my_ColorSpaceConverter`	

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	global, composition	
Usage	`get_instance_interface_ports <instanceName> <interfaceName>`	
Returns	`list of strings`	
Arguments	`instanceName`	Specifies the instance name of the module.
	`interfaceName`	Specifies an interface of instance.
Example	`get_instance_interface_ports my_ColorSpaceConverter outputInterface`	

get_instance_interface_properties

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

get_Instance_interface_properties	
Callback availability	global, composition
Usage	`get_instance_interface_properties <instanceName> <interfaceName>`

get_Instance_interface_properties		
Returns	`string`	
Arguments	`instanceName`	Specifies the instance name of the module.
	`interfaceName`	Specifies an interface of instance.
Example	`get_instance_interface_properties my_ColorSpaceConverter inputInterface`	

get_instance_property

This command returns the value of a single instance property.

get_Instance_property		
Callback availability	global, elaboration, validation, composition, generation 2	
Usage	*get_instance_property <instance> <property>*	
Returns	`various`	
Arguments	`instance`	The name of the instance.
	`property`	The name of the property (“Instance Properties”).
Example	`set my_name [ get_instance_property myinstance NAME ]`	
See Also	`“add_instance”, “get_instance_properties”, “set_instance_property”`	

set_instance_property

This command sets the value of a parameter for a child instance. Derived parameters and `SYSTEM_INFO` parameters for the child instance can not be set with this command.

set_Instance_property		
Callback availability	global, elaboration, validation, composition, generation 2	
Usage	*set_instance_property <instance> <property> <value>*	
Returns	`boolean`	
Arguments	`instance`	The name of the instance.
	`property`	The name of the property to set (“Instance Properties”).
	`value`	The new property value.
Example	`set_instance_property myinstance SUPRESS_ALL_WARNINGS true`	
See Also	`“add_instance”, “get_instance_properties”, “get_instance_property”`	

get_instance_properties

This command returns the names of all the instance properties as a list of strings. You can use the `get_instance_property` and `set_instance_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_Instance_properties		
Callback availability	discovery, global, edit, elaboration, validation, generation, composition, generation 2,upgrade	
Usage	*get_instance_properties*	
Returns	`EInstanceProperty[]`	List of strings ("Instance Properties").
Arguments	`None`	
Example	`get_instance_properties`	
See Also	`"add_instance","get_instance_property","set_instance_property"`	

get_instance_interface_property

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

get_Instance_interface_property		
Callback availability	global, composition	
Usage	`get_instance_interface_property <instanceName> <interfaceName> <propertyName>`	
Returns	`string`	
Arguments	`instanceName`	Specifies the instance name of the module.
	`interfaceName`	Specifies an interface of instance.
	`propertyName`	Specifies the property whose value is being retrieved.
Example	`get_instance_interface_property my_component s1 setupTime`	

get_instance_parameters

This command gets the parameters for an existing instance where the return value is an array of key/value pairs. It omits parameters that are derived and those that have the `SYSTEM_INFO` parameter property set.

get_Instance_parameters		
Callback availability	global, elaboration, validation, composition	
Usage	`get_instance_parameters <instanceName>`	
Returns	`list of strings`	
Arguments	`instanceName`	Specifies the name of the instance whose parameters are being retrieved.
Examples	`get_instance_parameters pixel_converter`	

get_Instance_parameters	
Notes	`You can use this command with instances created by add_instance or add_hdl_instance commands.`
See Also	`"get_instance_parameter_value", "get_instances", "set_instance_parameter_value" "add_hdl_instance"`

get_instance_parameter_property

This command returns the names of the specified instance parameter property.

get_Instance_parameter_property		
Callback availability	global, composition	
Usage	`get_instance_parameter_property <instanceName> <parameterName> <propertyName>`	
Returns	`string`, `boolean`, or `units`, depending on property.	
Arguments	`instanceName`	Specifies the instance name of the module.
	`parameterName`	Specifies the parameter for which a property is being retrieved.
	`propertyName`	Specifies the property whose value is being retrieved ("Parameter Properties").
Example	`get_instance_parameter_property my_stereo separate_control DISPLAY_NAME`	

get_instance_parameter_value

This command returns the value of a parameter in a child instance. 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	elaboration, validation, composition (refer to limitation on page 10–2)	
Usage	`get_instance_parameter_value <instanceName> <parameterName>`	
Returns	`string`, `boolean`, or `units`, depending on parameter.	
Arguments	`instanceName`	Specifies the name of the instance whose parameter is being retrieved.
	`parameterName`	Specifies the parameter whose value is being retrieved "Parameter Properties").
Examples	`get_instance_parameter_value pixel_converter input_DPI`	
Notes	`You can use this command with instances created by the add_instance or add_hdl_instance commands.`	
See Also	`"get_instance_parameters", "get_instances", "set_instance_parameter_value" "add_hdl_instance"`	

get_instance_port_property

This command returns a information about the port property specified.

get_instance_port_property		
Callback availability	global, composition	
Usage	`get_instance_port_property <instanceName> <portName> <propertyName>`	
Returns	`string`	
Arguments	`instanceName`	Specifies the instance name of the module.
	`portName`	Specifies a port.
	`property`	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	`get_instance_port_property my_uart width`	

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 memory-mapped they include base addresses and arbitration priorities.

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

set_instance_parameter_value

This command sets the value of a parameter for a child instance. Derived parameters and SYSTEM_INFO parameters for the child instance can not be set using this command.

set_instance_parameter_value		
Callback availability	composition	
Usage	set_instance_parameter_value *<instance> <parameter> <value>*	
Returns	None	
Arguments	instanceName	Specifies the name of the child module.
	parameterName	Specifies the parameter that is being set.
	parameterValue	Specifies the value of the parameter that is being set.
Examples	set_instance_parameter_value pixel_converter input_DPI 1200	
Notes	You can use this command with instances created by the add_instance or add_hdl_instance commands.	
See Also	"get_instance_parameter_value", "get_instances" "add_hdl_instance"	

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 possible if all parameterizations of a core yield identical HDL

add_fileset		
Callback availability	global	
Usage	`add_fileset <filesetName> <filesetKind> <callbackProcName> [<displayName>]`	
Returns	`string`	
Arguments	`filesetName`	The name of the fileset.
	`filesetKind`	Files support the following kinds: ■ `SIM_VHDL` ■ `SIM_VERILOG` ■ `QUARTUS_SYNTH` ■ `EXAMPLE_DESIGN`
	`calbackProcName`	A string identifying the name of the callback procedure. If you add files in the global section, you can then specify a blank callback procedure.
	`displayName`	A display string to identify the fileset.

Example	`add_fileset PCIE_SYNTHESIS QUARTUS_SYNTH mySynthProc`
	`proc mySynthProc { generated_name } {`
	`  ...`
	`}`	

add_fileset_file

This command adds an output file for 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	`add_fileset_file <fileDestination> <fileKind> <fileSource> <contentsOrPath>`	
Returns	`string`	
Arguments	`fileDestination`	Specifies the output file for the file after Qsys generation.
	`fileKind`	Files support the following kinds: ■ `VERILOG` ■ `SYSTEM_VERILOG` ■ `SYSTEM_VERILOG_INCLUDE` ■ `VHDL` ■ `SDC` ■ `MIF` ■ `HEX` ■ `DAT` ■ `OTHER`
	`fileSource`	The following sources are defined: ■ `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	`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"`	

set_fileset_property

Allows a user to set the properties of a fileset.

set_fileset_property			
Callback availability	fileset, generation		
Usage	`set_fileset_property <filesetName> <filesetProperty> <value>`		
Returns	`string`		
Arguments	`filename`	The name of the fileset.	
	`filesetProperty`	TOP_LEVEL	The of the top-level HDL module produced by this fileset
Example	`set_fileset_property mySynthFileset TOP_LEVEL simple_uart`		

get_fileset_file_attribute

This command gets the attribute of a fileset file.

get_fileset_file_attribute		
Callback availability	global, generation	
Usage	`get_fileset_file_attribute <output_file> <attribute>`	
Returns	`string`	Value of the Fileset file attribute.
Arguments	`output_file`	Location of the output file.
	`attribute`	Specifies the name of the attribute: ■ ALDEC_SPECIFIC (simulation) ■ CADENCE_SPECIFIC (simulation) ■ COMMON_SYSTEMVERILOG_PACKAGE (simulation) ■ MENTOR_SPECIFIC (simulation) ■ SYNOPSYS_SPECIFIC (simulation) ■ TOP_LEVEL_FILE (synthesis)
Example	`get_fileset_file_attribute my_file.sv ALDEC_SPECIFIC`	
See Also	`"add_fileset"`, `"add_fileset_file"`, `"set_fileset_file_attribute"`	

set_fileset_file_attribute

This command sets the attribute of a fileset file.

set_fileset_file_attribute		
Callback availability	global, generation	
Usage	`set_fileset_file_attribute <output_file> <attribute> <value>`	
Returns	`string`	Value of the attribute, if set.
Arguments	`output_file`	Location of the output file.
	`attribute`	Specifies the name of the attribute: ■ ALDEC_SPECIFIC (simulation) ■ CADENCE_SPECIFIC (simulation) ■ COMMON_SYSTEMVERILOG_PACKAGE (simulation) ■ MENTOR_SPECIFIC (simulation) ■ SYNOPSYS_SPECIFIC (simulation) ■ TOP_LEVEL_FILE (synthesis)
	`value`	Value to set the attribute to.
Example	`set_fileset_file_attribute my_file_pkg.sv COMMON_SYSTEMVERILOG_PACKAGE my_file_package`	
See Also	`"add_fileset"`, `"add_fileset_file"`, `"get_fileset_file_attribute"`	

get_fileset_sim_properties

This command gets simulator properties for a given fileset.

get_fileset_sim_properties		
Callback availability	global, generation 2	
Usage	`get_fileset_sim_properties <fileset> <platform> <property>`	
Returns	`string`	The fileset simulator properties.
Arguments	`fileset`	The name of the fileset.
	`platform`	The operating system for which this property applies: ■ ALL ■ LINUX32 ■ LINUX64 ■ WINDOWS32 ■ WINDOWS64
	`property`	Specifies the name of the property to set ("Simulator Properties").
Example	`get_fileset_sim_properties my_fileset LINUX64 OPT_CADENCE_64BIT`	
See Also	`"add_fileset", "set_fileset_sim_properties"`	

set_fileset_sim_properties

This command sets simulator properties for a given fileset.

set_fileset_sim_properties		
Callback availability	global, generation 2	
Usage	`set_fileset_sim_properties <fileset> <platform> <property> <value>`	
Returns	`string`	The fileset simulator properties.
Arguments	`fileset`	The name of the fileset.
	`platform`	The operating system for which this property applies: ■ ALL ■ LINUX32 ■ LINUX64 ■ WINDOWS32 ■ WINDOWS64
	`property`	Specifies the name of the property to set ("Simulator Properties").
	`value`	Specifies the value of the property.
Example	`set_fileset_sim_properties my_fileset LINUX64 OPT_MENTOR_PLI "{libA} {libB}"`	
See Also	`"add_fileset", "get_fileset_sim_properties"`	

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	`create_temp_file <fileName>`	
Returns	`string`	
Arguments	`fileName`	The name of the created file.
Example	`set filelocation [create_temp_file "./hdl/compute_frequency.v"` `add_fileset_file compute_frequency.v VERILOG PATH ${filelocation}`	

Miscellaneous

check_device_family_equivalence

This command returns 1 if the device family is equivalent to one of the families in the list, and returns 0 if the device family is not equivalent to any families.

check_device_equivalence		
Callback availability	global, elaboration, fileset	
Usage	`check_device_family_equivalence <deviceName> <deviceList>`	
Returns	`1 or 0`	Based on equivalence.
Arguments	`deviceName`	The device name of device being checked.
	`deviceList`	The device string list to check against.
Example	`check_device_equivalence "CYLCONE III LS" { "stratixv" "Cyclone IV"` `"cycloneiiils" }`	

get_device_family_displayname

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

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

set_qip_strings

This command places strings in the Quartus II IP File (**.qip)** file. The **.qip** file contains paths to the files for an IP core. You add the **.qip** file to your Quartus II project in the under **Files** in the **Settings** dialog box. Successive calls to `set_qip_strings` are not additive, they replace the previously declared value.

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

Simulator Properties

Table 10–8.

Name	Description
ENV_ALDEC_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running riviera-pro
ENV_CADENCE_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running ncsim
ENV_MENTOR_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running modelsim
ENV_SYNOPSYS_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running vcs
OPT_ALDEC_PLI	-pli option for riviera-pro
OPT_CADENCE_64BIT	-64bit option for ncsim
OPT_CADENCE_PLI	-loadpli1 option for ncsim
OPT_CADENCE_SVLIB	-sv_lib option for ncsim
OPT_CADENCE_SVROOT	-sv_root option for ncsim
OPT_MENTOR_64	-64 option for modelsim
OPT_MENTOR_CPPPATH	-cpppath option for modelsim
OPT_MENTOR_LDFLAGS	-ldflags option for modelsim
OPT_MENTOR_PLI	-pli option for modelsim
OPT_SYNOPSYS_ACC	+acc option for vcs
OPT_SYNOPSYS_CPP	-cpp option for vcs
OPT_SYNOPSYS_FULL64	-full64 option for vcs
OPT_SYNOPSYS_LDFLAGS	-LDFLAGS option for vcs
OPT_SYNOPSYS_LLIB	-l option for vcs
OPT_SYNOPSYS_VPI	+vpi option for vcs

Instance Properties

Table 10–9.

Name	Description
HDLINSTANCE_GET_GENERATED_NAME	You use this property to get the auto-generated fixed name when the instance property HDLINSTANCE_USE_GENERATED_NAME is set to TRUE. You can use this property only in FileSet callbacks.
HDLINSTANCE_USE_GENERATED_NAME	If TRUE, instances added with the `add_hdl_instance` command are instructed to use unique auto-generated fixed name based on the parameterization.
SUPPRESS_ALL_INFO_MESSAGES	If True, allows a component author to suppress ALL Info messages that originate from a child instance. `set_instance_property <instance_name> SUPPRESS_ALL_INFO_MESSAGES <true, false>`
SUPPRESS_ALL_WARNINGS	If TRUE, allows a component author to suppress ALL warnings that originate from a child instance. `set_instance_property <instance_name> SUPPRESS_ALL_WARNINGS <true, false>`

Port Roles (Interface Signal Types)

Each interfaces defines a number of signal roles and their behavior. Many signal roles are optional, allowing component designers the flexibility to select only the signal roles necessary to implement the required functionality.

AXI Interface Signal Types

For complete AXI interface specifications, refer to the *AMBA Protocol Specifications* on the ARM® website

AXI Master Interface Signal Types

Table 10–10. altera_axi_master (Part 1 of 2)

Name	Direction	Width
araddr	Output	1 to 64
arburst	Output	2
arcache	Output	4
arid	Output	1 to 18
arlen	Output	4
arlock	Output	2
arprot	Output	3

Table 10–10. altera_axi_master (Part 2 of 2)

Name	Direction	Width
arready	Input	1
arsize	Output	3
aruser	Output	1 to 64
arvalid	Output	1
awaddr	Output	1 to 64
awburst	Output	2
awcache	Output	4
awid	Output	1 to 18
awlen	Output	4
awlock	Output	2
awprot	Output	3
awready	Input	1
awsize	Output	3
awuser	Output	1 to 64
awvalid	Output	1
bid	Input	1 to 18
bready	Output	1
bresp	Input	2
bvalid	Input	1
rdata	Input	8, 16, 32, 64, 128, 256, 512, 1024
rid	Input	1 to 18
rlast	Input	1
rready	Output	1
rresp	Input	2
rvalid	Input	1
wdata	Output	8, 16, 32, 64, 128, 256, 512, 1024
wid	Output	1 to 18
wlast	Output	1
wready	Input	1
wstrb	Output	1, 2, 4, 8, 16, 32, 64, 128
wvalid	Output	1

AXI Slave Interface Signals

Table 10–11. altera_axi_slave (Part 1 of 2)

Name	Direction	Width
araddr	Input	1 to 64
arburst	Input	2
arcache	Input	4
arid	Input	1 to 18

Table 10–11. altera_axi_slave (Part 2 of 2)

Name	Direction	Width
arlen	Input	4
arlock	Input	2
arprot	Input	3
arready	Output	1
arsize	Input	3
aruser	Input	1 to 64
arvalid	Input	1
awaddr	Input	1 to 64
awburst	Input	2
awcache	Input	4
awid	Input	1 to 18
awlen	Input	4
awlock	Input	2
awprot	Input	3
awready	Output	1
awsize	Input	3
awuser	Input	1 to 64
awvalid	Input	1
bid	Output	1 to 18
bready	Input	1
bresp	Output	2
bvalid	Output	1
rdata	Output	8, 16, 32, 64, 128, 256, 512, 1024
rid	Output	1 to 18
rlast	Output	1
rready	Input	1
rresp	Output	2
rvalid	Output	1
wdata	Input	8, 16, 32, 64, 128, 256, 512, 1024
wid	Input	1 to 18
wlast	Input	1
wready	Output	1
wstrb	Input	1, 2, 4, 8, 16, 32, 64, 128
wvalid	Input	1

APB Interface Signal Types

Table 10–12. APB Interface Signal Types

Name	Width	Direction (APB master)	Direction (APB Slave)	Required
paddr	[1:32]	Output	Input	Yes
psel	[1:16]	Output	Input	Yes
penable	1	Output	Input	Yes
pwrite	1	Output	Input	Yes
pwdata	[1:32]	Output	Input	Yes
prdata	[1:32]	Input	Output	Yes
pslverr	1	Input	Output	No
pready	1	Input	Output	Yes
paddr31	1	Output	Input	No

Avalon Interface Signal Types

Avalon Memory-Mapped Signals

Table 10–13. Avalon-MM Signals *(1)* (Part 1 of 4)

Signal Role	Width	Direction	Description
Fundamental Signals			
address	1-32	Master → Slave	For masters, the address signal represents a byte address. The value of the address must be aligned to the data width. To write to specific bytes within a data word, the master must use the byteenable signal. For slaves, the interconnect translates the byte address into a word address in the slave's address space so that each slave access is for a word of data from the perspective of the slave. For example, address= 0 selects the first word of the slave and address 1 selects the second word of the slave.
begintransfer	1	Master → Slave	Asserted by the interconnect for the first cycle of each transfer regardless of waitrequest and other signals. If you do not include this signal in your Avalon-MM master interface, Qsys automatically generates this signal for you.

Table 10–13. Avalon-MM Signals *(1)* (Part 2 of 4)

Signal Role	Width	Direction	Description
byteenable byteenable_n	1, 2, 4, 8, 16, 32, 64, 128	Master → Slave	Enables specific byte lane(s) during transfers on ports of width greater than 8 bits. Each bit in byteenable corresponds to a byte in writedata and readdata. The master bit <*n*> of byteenable indicates whether byte <*n*> is being written to. During writes, byteenables specify which bytes are being written to; other bytes should be ignored by the slave. During reads, byteenables indicates which bytes the master is reading. Slaves that simply return readdata with no side effects are free to ignore byteenables during reads. If an interface does not have a byteenable signal, the transfer proceeds as if all byteenables are asserted. When more than one bit of the byteenable signal is asserted, all asserted lanes are adjacent. The number of adjacent lines must be a power of 2, and the specified bytes must be aligned on an address boundary for the size of the data. For example, the following values are legal for a 32-bit slave: 1111 writes full 32 bits 0011 writes lower 2 bytes 1100 writes upper 2 bytes 0001 writes byte 0 only 0010 writes byte 1 only 0100 writes byte 2 only 1000 writes byte 3 only Altera strongly recommends that you use the byteenable signal in components that are used in systems with different word sizes. Using the byteenable signal prevents unintended side effects in systems that include width adapters.
chipselect chipselect_n	1	Master → Slave	When present, a slave port ignores all Avalon-MM signals unless chipselect is asserted. chipselect must be used in combination with read or write. The chipselect signal is not necessary; Altera does not recommend using it.
debugaccess	1	Master → Slave	When asserted, allows internal memories that are normally write-protected to be written. For example, on-chip ROM memories can only be written when debugaccess is asserted.
read read_n	1	Master → Slave	Asserted to indicate a read transfer. If present, readdata is required.
readdata	8,16, 32, 64, 128, 256, 512, 1024	Slave → Master	The readdata driven from the slave to the master in response to a read transfer.
write write_n	1	Master → Slave	Asserted to indicate a write transfer. If present, writedata is required.
writedata	8,16, 32, 64, 128, 256, 512, 1024	Master → Slave	Data for write transfers. The width must be the same as the width of readdata if both are present.

Table 10–13. Avalon-MM Signals *(1)* (Part 3 of 4)

Signal Role	Width	Direction	Description
Wait-State Signals			
lock	1	Master → Slave	lock ensures that once a master wins arbitration, it maintains access to the slave for multiple transactions. It is asserted coincident with the first read or write of a locked sequence of transactions, and is deasserted on the final transaction of a locked sequence of transactions. lock assertion does not guarantee that arbitration is won, but after the lock-asserting master has been granted, it retains grant until it is deasserted. A master equipped with lock cannot be a burst master. Arbitration priority values for lock-equipped masters are ignored. lock is particularly useful for read-modify-write operations, where master A reads 32-bit data that has multiple bit fields, changes one field, and writes the 32-bit data back. If lock is not used, a master B could perform a write between Master A's read and write and master A's write would overwrite master B's changes.
waitrequest waitrequest_n	1	Slave → Master	Asserted by the slave when it is unable to respond to a read or write request. Forces the master to wait until the interconnect is ready to proceed with the transfer. At the start of all transfers, a master initiates the transfer and waits until waitrequest is deasserted. A master must make no assumption about the assertion state of waitrequest when the master is idle: waitrequest may be high or low, depending on system properties. When waitrequest is asserted, master control signals to the slave remain constant with the exception of begintransfer and beginbursttransfer,. An Avalon-MM slave may assert waitrequest during idle cycles. An Avalon-MM master may initiate a transaction when waitrequest is asserted and wait for that signal to be deasserted. To avoid system lockup, a slave device should assert waitrequest when in reset.
Pipeline Signals			
readdatavalid readdatavalid_n	1	Slave → Master	Used for variable-latency, pipelined read transfers. Asserted by the slave to indicate that the readdata signal contains valid data in response to a previous read request. A slave with readdatavalid must assert this signal for one cycle for each read access it has received. There must be at least one cycle of latency between acceptance of the read and assertion of readdatavalid. Required if the master supports pipelined reads. Bursting masters with read functionality must include the readdatavalid signal.

Table 10–13. Avalon-MM Signals *(1)* (Part 4 of 4)

Signal Role	Width	Direction	Description
Burst Signals			
burstcount	1–11	Master → Slave	Used by bursting masters to indicate the number of transfers in each burst. The value of the maximum burstcount parameter must be a power of 2, so a burstcount port of width <n> can encode a max burst of size $2^{(<n>-1)}$. For example, a 4-bit burstcount signal can support a maximum burst count of 8. The minimum burstcount is 1. The timing of the burstcount signal is controlled by the constantBurst property. Bursting masters with read functionality must include the readdatavalid signal. For bursting masters and slaves, the following restriction applies to the width of the address: <address_w> >= <burstcount_w> + floor($\log_2$ (<symbols_per_word_on_this_interface>))
beginbursttransfer	1	Master → Slave	Asserted for the first cycle of a burst to indicate when a burst transfer is starting. This signal is deasserted after one cycle regardless of the value of waitrequest.

Notes to Table 10–13:

(1) All Avalon signals are active high. Avalon signals that can also be asserted low list both versions of the signal in the **Signal role** column.

Avalon-ST Interface Signals

Table 10–14. Avalon-ST Interface Signals (Part 1 of 2)

Signal Role	Width	Direction	Description
Fundamental Signals			
channel	1–128	Source → Sink	The channel number for data being transferred on the current cycle. If an interface supports the channel signal, it must also define the maxChannel parameter.
data	1–4096	Source → Sink	The data signal from the source to the sink, typically carries the bulk of the information being transferred. The contents and format of the data signal is further defined by parameters.
error	1–256	Source → Sink	A bit mask used to mark errors affecting the data being transferred in the current cycle. A single bit in error is used for each of the errors recognized by the component, as defined by the errorDescriptor property.
ready	1	Sink → Source	Asserted high to indicate that the sink can accept data. ready is asserted by the sink on cycle <n> to mark cycle <n + readyLatency> as a ready cycle, during which the source may assert valid and transfer data. Sources without a ready input cannot be backpressured, and sinks without a ready output never need to backpressure.

Table 10–14. Avalon-ST Interface Signals (Part 2 of 2)

Signal Role	Width	Direction	Description
valid	1	Source → Sink	Asserted by the source to qualify all other source to sink signals. On ready cycles where valid is asserted, the data bus and other source to sink signals are sampled by the sink, and on other cycles are ignored. Sources without a valid output implicitly provide valid data on every cycle that they are not being backpressured, and sinks without a valid input expect valid data on every cycle that they are not backpressuring.
Packet Transfer Signals			
empty	1–8	Source → Sink	Indicates the number of symbols that are empty during cycles that contain the end of a packet. The empty signal is not used on interfaces where there is one symbol per beat. If endofpacket is not asserted, this signal is not interpreted.
endofpacket	1	Source → Sink	Asserted by the source to mark the end of a packet.
startofpacket	1	Source → Sink	Asserted by the source to mark the beginning of a packet.

Tri-state Slave Interface Signals

Table 10–15. Avalon-MM tri-state Slave Signals *(1)* (Part 1 of 2)

Signal Role	Width	Direction	Req'd	Description
address	1-32	In	No	Address lines to the slave port. Specifies a byte offset into the slave's address space.
read read_n	1	In	No	Read-request signal. Not required if the slave port never outputs data. If present, data must also be used.
write write_n	1	In	No	Write-request signal. Not required if the slave port never receives data from a master. If present, data must also be present, and writebyteenable cannot be present.
chipselect chipselect_n	1	In	No	When present, the slave port ignores all Avalon-MM signals unless chipselect is asserted. chipselect is always present in combination with read or write.
outputenable outputenable_n	1	In	Yes	Output-enable signal. When deasserted, a tri-state slave port must not drive its data lines otherwise data contention may occur.
data	8,16, 32, 64, 128, 256, 512, 1024	Bidir	No	Bidirectional data. During write transfers, the FPGA drives the data lines. During read transfers the slave device drives the data lines, and the FPGA captures the data signals and provides them to the master.

Table 10–15. Avalon-MM tri-state Slave Signals *(1)* (Part 2 of 2)

Signal Role	Width	Direction	Req'd	Description
byteenable byteenable_n	2, 4, 8,16, 32, 64, 128	In	No	Enables specific byte lane(s) during transfers. Each bit in byteenable corresponds to a byte lane in data. During writes, byteenables specify which bytes the master is writing to the slave. During reads, byteenables indicates which bytes the master is reading. Slaves that simply return data with no side effects are free to ignore byteenables during reads. When more than one byte lane is asserted, all asserted lanes are guaranteed to be adjacent. The number of adjacent lines must be a power of 2, and the specified bytes must be aligned on an address boundary for the size of the data. The are legal values for a 32-bit slave: 1111writes full 32 bits 0011writes lower 2 bytes 1100writes upper 2 bytes 0001writes byte 0 only 0010writes byte 1 only 0100writes byte 2 only 1000writes byte 3 only
writebyteenable writebyteenable_n	2,4,8,16, 32, 64,128	In	No	Equivalent to the logical AND of the byteenable and write signals. When used, the write signal is not used.
begintransfer	1	In	No	Asserted for the first cycle of each transfer.

Note to Table 10–13:

(1) All Avalon signals are active high. Avalon signals that can also be asserted low list both versions in the **Signal Role** column.

Tri-state Conduit Interface Signals

Table 10–16. Tristate Conduit Interface Signal Roles

Signal Role	Width	Direction	Required	Description
request	1	Master → Slave	Yes	The meaning of request depends on the state of the grant signal, as the following rules dictate. 1. When request is asserted and grant is deasserted, request is requesting access for the current cycle. 2. When request is asserted and grant is asserted, request is requesting access for the next cycle; consequently, request should be deasserted on the final cycle of an access. Because request is deasserted in the last cycle of a bus access, it can be reasserted immediately following the final cycle of a transfer, making both rearbitration and continuous bus access possible if no other masters are requesting access. Once asserted, request must remain asserted until granted; consequently, the shortest bus access is 2 cycles.
grant	1	Slave → Master	Yes	When asserted, indicates that a tristate conduit master has been granted access to perform transactions. grant is asserted in response to the request signal and remains asserted until 1 cycle following the deassertion of request. The design of the Avalon-TC Interface does not allow a default Avalon-TC master to be granted when no masters are requesting.
<name>_in	1–1024	Slave → Master	No	The input signal of a logical tristate signal.
<name>_out	1–1024	Master → Slave	No	The output signal of a logical tristate signal.
<name>_outen	1	Master → Slave	No	The output enable for a logical tristate signal.

Avalon Clock Sink Interface Signals

Table 10–17. Clock Input Signal Roles

Signal Role	Width	Direction	Required	Description
clk	1	Input	Yes	A clock signal. Provides synchronization for internal logic and for other interfaces.

Avalon Clock Source Interface Signals

Table 10–18. Clock Source Signal Roles

Signal Role	Width	Direction	Required	Description
clk	1	Output	Yes	An output clock signal.

Avalon Conduit Interface Signals

Table 10–19. Conduit Signal Roles

Signal Role	Width	Direction	Description
<any>	<n>	In, out or bidirectional	A conduit interface consists of one or more signals of arbitrary width of direction input or output. Compatible conduit interfaces can be connected inside the Qsys system, exported to the next level of the hierarchical design, or to the top-level of the Qsys system.

Interrupt Sender Interface Signals

Table 10–20. Interrupt Sender Signal Roles

Signal Role	Width	Direction	Required	Description
irq irq_n	1	Output	Yes	Interrupt Request. A slave asserts irq when it needs to be serviced.

Interrupt Receiver Interface Signals

Table 10–21. Interrupt Receiver Signal Roles

Signal Role	Width	Direction	Required	Description
irq	1–32	Input	Yes	irq is an <n>-bit vector, where each bit corresponds directly to one IRQ sender, with no inherent assumption of priority.

Document Revision History

Table 10–22 shows the revision history for this document.

Table 10–22. Document Revision History (Part 1 of 2)

Date	Version	Changes
November 2013	13.1.0	■ add_hdl_instance
May 2013	13.0.0	■ Consolidated content from other Qsys chapters. ■ Added AMBA APB support.
November 2012	12.1.0	■ Added the **demo_axi_memory** example with screen shots and example **_hw.tcl** code.
June 2012	12.0.0	■ Added AMBA AXI3 support. ■ Added the set_display_item_property, set_parameter_property parameter LONG_DESCRIPTION, and static filesets.
November 2011	11.1.0	■ Template update. ■ Added commands set_qip_strings, get_qip_strings, get_device_family_displayname, check_device_family_equivalence ■ Updated text to reflect changes in 11.1.

Table 10–22. Document Revision History (Part 2 of 2)

Date	Version	Changes
May 2011	11.0.0	■ Removed Beta status. ■ Revised section describing HDL and composed component implementations. ■ Separated reset and clock interfaces in examples. ■ Added the `TRISTATECONDUIT_INFO`, `GENERATION_ID`, and `UNIQUE_ID SYSTEM_INFO` properties. ■ Added the `WIDTH` and `SYSTEM_INFO_ARG` parameter properties. ■ Removed the `doc_type` argument from the `add_documentation_link` command. ■ Removed `get_instance_parameter_properties` command. (`get_instance_parameter_property` is available.) ■ Added the `add_fileset`, `add_fileset_file` and `create_temp_file` commands. ■ Updated Tcl examples to show separate clock and reset interfaces.
December 2010	10.1.0	Initial release.

For more information about Tcl syntax, refer to the Tcl Developer Xchange website.

For previous versions of the *Quartus II Handbook*, refer to the Quartus II Handbook Archive.