

A Qsys *component* is a hardware design block available within Qsys that can be instantiated in a Qsys system. You can use Altera-provided or define custom Qsys components as hierarchical building blocks in creating Qsys systems. This chapter describes the structure of Qsys components, with an emphasis on the using the component editor to create and edit the Hardware Component Description File (**_hw.tcl**) that describes a component to Qsys.

This chapter includes the following major sections:

- “Qsys Components” on page 6–1
- “Component Editor” on page 6–8


Qsys Components

A Qsys component includes the following elements:

- The HDL description of the component’s hardware.
- A description of the interface to the component hardware, such as the names and types of I/O signals.
- A description of the parameters that determine the operation of the component.
- A parameter editor for customizing an instance of the component in Qsys.
- Scripts and other information Qsys requires to generate the HDL files for the component and integrate the component into the Qsys system.
- Other component-related information, such as references to software drivers, necessary for development steps downstream of Qsys.

Component Providers

Qsys components are provided by multiple sources, including the following:

- Altera provides a great variety of components automatically installed with the Quartus® II software.
- You can use the Qsys component editor to define your own custom Qsys components.
- Third-party IP developers provide Qsys-compliant components. the **Intellectual Property & Reference Designs** web page, type Qsys Certified  in the **Search** box labeled **Search for IP and Reference Designs products by their descriptions**.
- Altera® development kits which are listed on the **All Development Kits** web page.

Component Interfaces

You can design Qsys components with any combination of the following Avalon interface types:

- Avalon Memory-Mapped (Avalon-MM)—for Avalon-MM master and slaves that communicate using read and write commands.
- Avalon Streaming (Avalon-ST)—for point-to-point connections between Avalon-ST sources and sinks that stream data.
- Avalon Tri-state Conduits (Avalon-TC)—for a tri-state conduit controller in your Qsys system to tri-state devices on your PCB.
- Avalon Interrupts—for point-to-point connections between interrupt senders that generate interrupts and interrupt receivers that service interrupts.
- Avalon Clocks—for point-to-point connections between clock sources and clock sinks.
- Avalon Resets—for point-to-point connections between reset sources and reset sinks.
- Avalon Conduits—for point-to-point connections between conduit interfaces. You can use the conduit interface type to define an arbitrary collection of signals that do not fit into any of the other Avalon interface categories.

A single component can use as many of these interface types as it requires. For example, a component might provide an Avalon-ST source port for high-throughput data, in addition to an Avalon-MM slave port for control. All components must include the Avalon Clock and Avalon Reset interface types.

Component Types

You can build more flexibility into your components by writing a generation callback routine which generates your HDL programmatically. The following sections describe the different component types.

Static HDL Components

A static `_hw.tcl` file defines the top-level HDL file and associated component files. The HDL that describes a static component is created by the component author and is not changed by users of the component. HDL parameters are available when instantiating the component.


Generated HDL Components

Alternatively, you can also define a component whose HDL is generated based on the value of its declared parameters. These components use a custom callback to generate the HDL for each instance of the component.

For example, you could write a custom callback to include a control and status interface based on the value of a status interface parameter. The callback overcomes a limitation of HDL languages which do not allow runtime parameters.

Composed HDL Components

Composed components are constructed from combinations of other components. You can use a compose callback to connect and parameterize a composed component. Composed components can be static or generated.

 For more information about defining your own generation or compose callback procedure, refer to the “Generation Callback” and “Compose Callback” sections in the *Component Interface Tcl Reference* chapter in volume 1 of the *Quartus II Handbook*.

Component Structure

Components are defined with a `_hw.tcl` file, a text file written in the Tcl scripting language that describes the component to Qsys. You can author an `_hw.tcl` file by creating a text file manually or using the component editor. This section describes the structure of `_hw.tcl` components and how they are stored.

Component Description File (`_hw.tcl`)

Component files include the following elements:

- A component description file, which is a Tcl file with file name of the form `<entity name>_hw.tcl`.
- SystemVerilog, Verilog HDL, or VHDL files that define the custom component.

The `_hw.tcl` file defines everything that Qsys requires about the name and location of component design files, including files for simulation and constraint files.

The component editor simplifies the process of creating components and automatically saves components in the `_hw.tcl` format. You can use these Tcl files as a template for editing components by hand. When you edit a previously saved `_hw.tcl` file, Qsys automatically backs up the earlier version as `_hw.tcl~`.

For more information about `_hw.tcl` file details, refer to the *Component Interface Tcl Reference* chapter in volume 1 of the *Quartus II Handbook*.

Component File Organization

A typical component uses the following directory structure. The precise names of the directories are not significant.

- `<component_directory>/`
 - `<hdl>/`—a directory that contains the component HDL design files and the `_hw.tcl` file.
 - `<component name>_hw.tcl`—the component description file.
 - `<component name>.v` or `.vhd`—the HDL file that contains the top-level module.
 - `<component_name>_sw.tcl`—the software driver configuration file. This file specifies the paths for the `.c` and `.h` files associated with the component.
 - `<software>/`—a directory that contains software drivers or libraries related to the component, if any. Altera recommends that the software directory be a subdirectory of the directory that contains the `_hw.tcl` file.

- For information about writing a device driver or software package suitable for use with the Nios II processor, refer to the *Hardware Abstraction Layer* section of the *Nios II Software Developer's Handbook*. The *Nios II Software Build Tool Reference* chapter of the *Nios II Software Developer's Handbook* describes the commands you can use in the Tcl script.

Component Versioning

You can create and maintain multiple versions of the same component using one of the following options:

- Define the module property `version` in your `_hw.tcl` file.
- If multiple versions of the component are defined in your component libraries, you can add a different version of a component by right-clicking on the component and selecting **Add version** `<version_number>`.
- You can create an IP Index (`.ipx`) file in the same directory as your Qsys project to control the search path for your project.

Component Search Path

Qsys searches for component files each time you open the tool. Qsys locates and displays the list of available components in the component library. Qsys searches the directories in the IP search path for the following component file types:

- Hardware Component Description Files (`_hw.tcl`) files. Each `_hw.tcl` file defines a single component.
- IP Index (`.ipx`) files. Each file indexes a collection of available components, or a reference to other directories to search. In general, `.ipx` files facilitate faster startup for Qsys and other tools because fewer files need to be read and analyzed.


Qsys searches some directories recursively and other directories only to a specific depth. In the following list of search locations, a recursive descent is annotated by `**`. The `*` signifies any file. When a directory is recursively searched, the search stops at any directory containing a `_hw.tcl` or `.ipx` file; subdirectories are not searched.

The following directories are searched:

- `$$PROJECT_DIR/*`
- `$$PROJECT_DIR/ip/**/*`
- `$QUARTUS_INSTALLDIR/./ip/**/*`

Complete the following steps to extend the default search path by specifying additional directories:

1. On the Tools menu click **Options**.
2. In the **Category** list, click **IP Search Path**.
3. Click **Add**.
4. Browse to locate additional directories and click **Open** to add them to your search path.

 These additional paths apply to all projects; that is, the paths are global to the current version of Qsys. The search path is ultimately defined by the file, `<$QUARTUS_INSTALLDIR>/sopc_builder/bin/root_components.ipx`.

Adding Components to the Library

Use one of the following methods to add components to the Component Library:

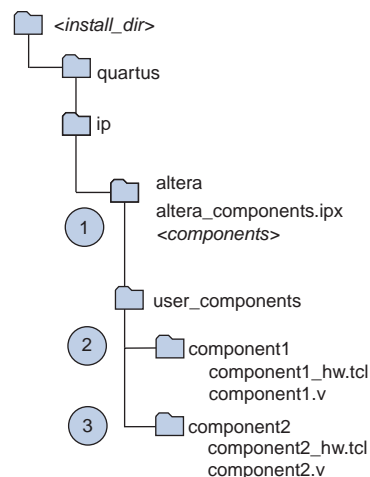
Copy to the IP Root Directory

The simplest method to add a new component is to copy your components into the standard IP directory provided by Altera. This approach is useful in the following situations:

- You want to associate your components with a specific release of the Quartus II software
- You want to and have the same components available across multiple projects


Figure 6-1 illustrates this approach.

Figure 6-1. User Library Included In Subdirectory of \$IP_ROOTDIR



In Figure 6-1, the circled numbers identify three steps of the component discovery algorithm that Qsys follows during initialization. These steps are explained in the following paragraphs.

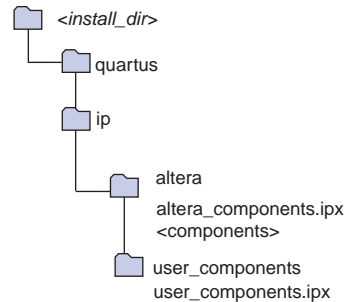
1. Qsys recursively searches the `<install_dir>/ip/` directory by default. It finds the file in the `altera` subdirectory, which tells it about all of the Altera components. **altera_library.ipx** includes listings for all components found in its subdirectories. The recursive search stops when Qsys finds this **.ipx** file.
2. As part of its recursive search, Qsys also looks in the adjacent **user_components** directory. Qsys finds the **component1** directory, which contains **component1_hw.tcl**. When Qsys finds that component, the recursive search stops so that no components in subdirectories of **component1** are found.
3. Qsys then searches in the adjacent **component2** directory, which includes **component2_hw.tcl**. If Qsys finds that component, the recursive search stops.

 If you save your `_hw.tcl` file in the `<install_dir>/ip/` directory, Qsys finds your `_hw.tcl` file and stops. Qsys does not conduct the search just described.

Reference Components in an .ipx File

Alternatively, you can specify the search path in a `user_components.ipx` file under `<install_dir>/ip` path. This method allows you to store components in a location that is not linked to your Quartus II installation and to add a location that is independent of the default search path. Figure 6-2 illustrates this approach.

Figure 6-2. Specifying A User .ipx directory




The `user_components.ipx` file includes a single line of code redirecting Qsys to the location of the user library. Example 6-1 shows the code for this redirection.

Example 6-1. Redirect to User Library

```

<library>
  <path path="<user_lib_dir>/user_ip/**/*" />

/<library>
  
```

 For both of these approaches, if you install a new version of the Quartus II software, you must also repeat the steps to include your components.

You can verify that components are available and also decrease the time it takes to launch Qsys by using the utilities, `ip-catalog` and `ip-make-ipx` commands. The following sections describe these commands.

ip-catalog

This command displays the a catalog of available components in either plain text or XML format.

Usage

```

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

Options

- `--project-dir[=<directory>]`. Optional. Components can be found in certain locations relative to the project, if any. By default, the current directory, `'.'` is used. To exclude any project directory, use `"`.
- `--name[=<value>]`. Optional. This argument provides a pattern to filter the names of the components found. To show all components, use a `*` or `'`. By default, all components are shown. The argument is not case sensitive.
- `--verbose[=<true/false>]`. Optional. When true, reports the progress of the command.
- `--xml[=<true/false>]`. Optional. When true, prints the output in XML format instead of a line- and colon-delimited format.
- `--help`. Shows help for the `ip-catalog` command.

ip-make-ipx

This command creates an index file for the directory specified. It returns a 0 for successful completion and a non-zero value for failure.

Usage

```
ip-make-ipx --source-directory[=<directory>] --output[=<file>]  
--relative-vars[=<value>] --thorough-descent  
--message-before[=<value>] --message-after[=<value>] --help
```

Options

- `--source-directory=<directory>`. Optional. The directory to index. The default directory is `"."`. You can also provide a comma separated list of directories.
- `--output[=<file>]`. Optional. The name of the file to generate. The default name is `./components.ipx`.
- `--relative-vars[=<value>]`. Optional. Causes the output file to include references relative to the specified variable or variables where possible. You can specify multiple variables as a comma-separated list.
- `--thorough-descent[=<true/false>]`. Optional. If set, a component or `.ipx` file in a directory does not prevent subdirectories from being searched.
- `--message-before[=<value>]`. Optional. A message to print to stdout when indexing begins
- `--message-after[=<value>]`. Optional. A message to print to stdout when indexing completes
- `--help`. Show help for this command

Understanding IPX File Syntax

An `.ipx` file is an XML file whose top-level element is a library with any number of subelements to define paths and components. [Example 6-2](#) illustrates this format.

Example 6-2. `.ipx` File Structure

```
<library>
  <path>... </path>
  <path>... </path>
  ...
  <component>... </component>
  ...
</library>
```

A `<path>` element contains a single attribute, also called `path` and may reference a directory with a wildcard, (`*`), or reference a single file. Two asterisks designate any number of subdirectories. A single asterisk designates a match to a single file or directory. In searching down the designated path, the following three types of files are identified:

- `.ipx`—additional index files
- `_hw.tcl`—Qsys component definitions
- `_sw.tcl`—Nios II board support package (BSP) software component definitions

A `<component>` element contains several attributes to define a component. If you provide all the required details for each component in an `.ipx` file, the start-up time for Qsys is less than if Qsys must discover the files in a directory. [Example 6-3](#) shows two `<component>` elements. Note that the paths for file names are specified relative to the `.ipx` file.

Example 6-3. Component Elements

```
<library>
  <component
    name="A Qsys Component"
    displayName="Qsys FIR Filter Component"
    version="2.1"
    file="./components/qsys_filters/fir_hw.tcl"
  />
  <component
    name="rgb2cmyk_component"
    displayName="RGB2CMYK Converter(Color Conversion Category!)"
    version="0.9"
    file="./components/qsys_converters/color/rgb2cmyk_hw.tcl"
  />
</library>
```

Component Editor

The Qsys component editor is a GUI that allows you to define a component and its parameter editor GUI. You use the component editor to do the following:

- Specify the SystemVerilog, Verilog HDL, or VHDL files that describe the modules in your component, simulation, and constraint files

- Conversely, create an HDL template for a component by first defining its interface using the **HDL Files** tab of the component editor.
- Specify the signals for each of the component's interfaces, and define the behavior of each interface signal.
- Specify relationships between interfaces, such as determining which clock interface is used by a slave interface.
- Declare any parameters that alter the component structure or functionality, and define a user interface to let users parameterize instances of the component.

After you define your component in the component editor the component is available in the component library. The following sections explain how to use the component editor.


Component Hardware Structure

The component editor allows you to define components with one or more interfaces. For a description of the available interface types refer to “[Component Interfaces](#)” on [page 6-2](#). You can specify exported interfaces which appear at the top-level of the Qsys system. You can connect exported interfaces to devices on the PCB or to other Qsys subsystems in hierarchical designs.

You can also use the component editor to generate an early version of the `_hw.tcl` file and then manually edit this file to complete the component definition.

Starting the Component Editor

To start the component editor in Qsys, on the **File** menu, click **New Component**. When the component editor starts, the **Introduction** tab describes how to use the component editor.

 Each tab in the component editor provides on-screen information that describes how to use the tab. Click the triangle labeled **About** at the top-left of each tab to view these instructions.

 You can also refer to [Component Editor \(Qsys\)](#) in Quartus II Help for additional information about the component editor.

HDL Files Tab

The **HDL Files** tab allows you to create a Qsys component from existing Verilog HDL or VHDL files, or to create an HDL template in either Verilog HDL or VHDL for a Qsys component by first specifying its interfaces. The following sections describe both the bottom-up and top-down approaches to component design.

Bottom-Up Component Design

You can use the **HDL Files** tab to specify Verilog HDL or VHDL files that describe the component logic. Files are provided to downstream tools such as the Quartus II software and ModelSim® in the same order as they appear in the **HDL Files** table.

You can also use the component editor to define the interface to components outside the Qsys system. In this case, you do not provide HDL files. Instead, you use the component editor to interactively define the hardware interface.

After you specify an HDL file, the Quartus II Analysis and Elaboration analyzes signals and parameters declared for all modules in the top-level file. After successful analysis, the component editor **Signals** tab lists all design modules in the **Top Level Module** list. If your HDL contains more than one module, you must select the appropriate top-level module from the **Top Level Module** list.

All files are managed in a single table, with options for **Synth** and **Sim**. You can select the **Top** option to specify the top-level file for synthesis. When the top-level module is changed, the component editor performs best-effort signal matching against the existing port definitions. If a port is absent from the module, it is removed from the port list. You can use the up and down arrows to specify the HDL file analysis order.

By default, all files are added with both **Synth** and **Sim** options turned on. To add a simulation-only file, turn off the **Synth** option for that file. Simulation files are passed to ModelSim for simulation. To add a synthesis-only file, turn off the **Sim** file option.



The component editor determines the signals on the component when only the top-level module or entity is added to the table, but all of the files required for the component must be added for the component to compile in Quartus II software or work in simulation.

Top-Down Component Design

The **Create HDL Template** button on the **HDL Files** tab allows you to create an HDL template for a component if you have not provided a HDL description for it. Clicking the **Create HDL Template** button shows you the component HDL and lets you choose between Verilog HDL and VHDL. Altera recommends that you define your signals, interfaces, parameters and basic component information, including the component name, before creating the HDL template by clicking **Save**. The component editor writes `<component_name>.v` or `<component_name>.vhd` to your project directory.

After you have created the component's HDL code, you can add other files that are required to define your component, including the `_hw.tcl` file, and synthesis and simulation files using the **Add** button on the **HDL Files** tab.

Signals Tab

You use the **Signals** tab to specify the purpose of each signal on the top-level component module. If you specified a file on the **HDL Files** tab, the signals on the top-level module appear on the **Signals** tab.

The **Interface** list also allows creation of a new interface so that you can assign a signal to a different interface without first switching to the **Interfaces** tab. Each signal must belong to an interface and be assigned a legal signal type for that interface. In addition to Avalon-MM and Avalon-ST interfaces, components may have Avalon Tri-state Conduit (Avalon-TC), Avalon clock, Avalon Interrupt, Avalon Reset, and Avalon Conduit interfaces.

Naming Signals for Automatic Type and Interface Recognition

The component editor recognizes signal types and interfaces based on the names of signals in the source HDL file, if they conform to the following naming conventions:

Signal associated with a specific interface—*<interface type>_<interface name>_<signal type>[_n]*

For any value of *<interface_name>* the component editor automatically creates an interface by that name, if necessary, and assigns the signal to it. The *<signal_type>* must match one of the valid signal types for the type of interface. Refer to the *Avalon Interface Specifications* for the signal types available for each interface type. You can append *_n* to indicate an active-low signal. [Table 6-1](#) lists the valid values for *<interface_type>*.

Table 6-1. Valid Values for <Interface Type>

Value	Meaning
avs	Avalon-MM slave
avm	Avalon-MM master
aso	Avalon-ST source
asi	Avalon-ST sink
cs0	Clock output
csi	Clock input
coe	Conduit
inr	Interrupt receiver
ins	Interrupt sender
ncm	Nios II custom instruction master
ncs	Nios II custom instruction slave
rsi	Reset sink
rso	Reset source
tcm	Avalon-TC master
tcs	Avalon-TC slave

Example 6-4 shows a Verilog HDL module declaration with signal names that infer two Avalon-MM slaves.

Example 6-4. Verilog HDL Module With Automatically Recognized Signal Names

```
module my_slave_irq_component (

    csi_clockreset_clk; // clock interface
    csi_clockreset_reset_n; //reset clock interface

    avs_sl_address; //s1 slave interface
    avs_sl_read; //s1 slave interface
    avs_sl_write; //s1 slave interface
    avs_sl_writedata; //s1 slave interface
    avs_sl_readdata; //s1 slave interface
    ins_irq0_irq; //irq0 interrupt sender interface
);

    input csi_clockreset_clk;
    input csi_clockreset_reset_n;
    input [7:0] avs_sl_address;
    input avs_sl_read;
    input avs_sl_write;
    input [31:0] avs_sl_writedata;
    output wire[31:0] avs_sl_readdata;
    output wire ins_irq0_irq;

/* Insert your logic here */

endmodule
```

Templates for Interfaces to External Logic

You can use the **Create HDL Template** command to generate an HDL template for the component. Then, you connect these signals outside of the Qsys system. If your component uses an Avalon interface to interface outside of the Qsys system, you can use the Templates menu in the component editor to add typical interface signals to your signal list. There are templates for the following interfaces:

- Avalon-MM Slave
- Avalon-MM Master
- Avalon-MM Slave with Interrupt
- Avalon-MM Master with Interrupt
- Avalon-TC Slave
- Avalon-TC Master
- Avalon-ST Sink
- Avalon-ST Source
- Nios Custom Instruction Slave – Combinational
- Nios Custom Instruction Slave – Variable Multi-cycle
- Nios Custom Instruction Slave – Fixed Multi-cycle

- Nios Custom Instruction Slave – Extended
- Nios Custom Instruction Slave – Internal Register File

After adding a typical Avalon interface using a template, you can add or delete signals to customize the interface.

Interfaces Tab

The **Interfaces** tab allows you to configure the interfaces on your component and specify a name for each interface. The interface name identifies the interface and appears in the Qsys connection panel. The interface name is also used to uniquely identify any signals that are ports on the top-level Qsys system.

The **Interfaces** tab allows you to configure the type and properties of each interface. For example, an Avalon-MM slave interface has timing parameters that you must set appropriately. The **Interfaces** tab displays waveforms that illustrate the timing that you specify. If you update the timing parameters, the waveforms automatically update to illustrate the new timing. The waveforms are available for the following interface types:

- Avalon-MM
- Avalon-ST
- Interrupts

HDL Parameters Tab

You specify the parameters that users of your component can set to configure your component on the **HDL Parameters** tab. The **Parameters** table included on this tab displays Verilog HDL parameters or VHDL generics that you declared in the top-level HDL module. Using the **Parameters** table, you can specify the following information about each parameter:

- Default value
- Whether or not it is user-editable
- Type
- Group
- Tooltip

Click **Preview the GUI** at any time to see how the component GUI appears.

The following rules apply to HDL parameters exposed via the component parameter editor:

- Editable parameters cannot contain computed expressions.
- If a parameter $\langle n \rangle$ defines the width of a signal, the signal width must be of the form $\langle n-1 \rangle : 0$.
- When a VHDL component is used in a Verilog HDL Qsys system, or a Verilog HDL component is used in a VHDL Qsys system, numeric parameters must be 32-bit decimal integers. When passing other numeric parameter types, unpredictable results occur. (The interconnect is written in Verilog HDL and SystemVerilog.)

- Refer to *Component Interface Tcl Reference* chapter in volume 1 of the *Quartus II Handbook* for detailed information about creating and displaying parameters using Tcl scripts.

Library Info

The **Library Info** tab allows you to specify the following information about your component:

- **Name**—Specifies the component name. When you save your component, the component editor saves your component to the string that you specified concatenated to the `_hw.tcl` suffix, for example, `my_component_hw.tcl`
- **Display Name**—Specifies the user-visible name for this component in Qsys.
- **Version**—Specifies the version number of the component.
- **Group**—Specifies which group in Qsys displays your component in the list of available components. If you enter a previously unused group name, Qsys creates a new group by that name.
- **Description**—Allows you to describe the component.
- **Created By**—Allows you to specify the author of the component.
- **Icon**—Allows you to place an image in the title bar of your component, in place of the MegaCore logo. The icon can be a `.jpg`, `.gif`, or `.png` file. The directory for the icon is relative to the directory that contains the `_hw.tcl` file.
- **Documentation**—Allows you to specify multiple documents that pertain to your component. You can use this property to specify a file on the Internet or in your company's file system. The specified file can be in either `.html` or `.pdf` format. To specify an Internet file, begin your path with `http://`, for example: `http://mydomain.com/datasheets/my_memory_controller.html`. To specify a file in your company's file system, you begin your path with `file:///` for Linux and `file://` for Windows, for example: `file:///company_server/datasheets/my_memory_controller.pdf`. For handwritten `_hw.tcl` files, you can specify documentation using the `add_documentation_link` Tcl command. [Example 6-5](#) shows how to specify documentation that is included in the component directory.

Example 6-5. Documentation Link for Documentation Stored with Component HDL Files

```
set_module_property DATASHEET_URL
"file:[get_module_property MODULE_DIRECTORY]Modular_SGDMA_Dispatcher_Core_UG.pdf"
```

- For more information refer to the `add_documentation_link` command in the *Component Tcl Interface Reference*.

Saving a Component

You can save the component by clicking **Finish** on any of the tabs, or by clicking **Save** on the File menu. Based on the settings you specify in the component editor, the component editor creates a component description file with the file name `<class-name>_hw.tcl`. The component editor saves the file in the same directory as the HDL file that describes the component's hardware interface. If you did not specify an HDL file, you can save the component description file to any location you choose.

You can relocate component files later. For example, you could move component files into a subdirectory and store it in a central network location so that other users can instantiate the component in their systems. The `_hw.tcl` file contains relative paths to the other files, so if you move the `_hw.tcl` file you should move all the HDL and other files associated with it.



Altera recommends that you store `_hw.tcl` files for a project in the `ip/<class-name>` directory for the project. You should store the HDL and other files in the same directory as the `_hw.tcl` file.

Editing a Component

After you save a component and exit the component editor, you can edit it in Qsys. To edit a component, right-click it in the list of available components on the **System Contents** tab and click **Edit Component**. The component editor appears.



You cannot edit components that were created outside of the component editor, such as Altera-provided components.

If you edit the HDL for a component and change the interface to the top-level module, you need to edit the component to reflect the changes you made to the HDL.

Registering Software Assignments

You can use Tcl commands to create software assignments. You can register any software assignment that you want, as arbitrary key-value pairs. [Example 6-6](#) shows a typical Tcl API script:

Example 6-6. Typical Software Assignment with Tcl API Scripting

```
set_module_assignment name value  
set_interface_assignment name value
```

The assignments are added to the Qsys information file (`.sopcinfo`), available for use for downstream components.



For more information about these software assignments, refer to the [Publishing Component Information to Embedded Software](#) chapter in the Nios II software Developer's Handbook.

Component Parameterization

To edit component instance parameters, select a component in the **System Contents** tab of Qsys and click **Edit**.


Document Revision History

Table 6–2 shows the revision history for this document.

Table 6–2. Document Revision History

Date	Version	Changes
November 2011	11.1.0	Template update.
May 2011	11.0.0	<ul style="list-style-type: none"> ■ Removed beta status. ■ Added Avalon Tri-state Conduit (Avalon-TC) interface type. ■ Added many interface templates for Nios custom instructions and Avalon-TC interfaces.
December 2010	10.1.0	Initial release.

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

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