



This chapter describes the SOPC Builder component editor. The component editor provides a GUI to support the creation and editing of the Hardware Component Description File (`_hw.tcl`) file that describes a component to SOPC Builder. You use the component editor to do the following:

- Specify the Verilog HDL or VHDL files that describe the modules in your component hardware.
- 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.

 For information about using the component editor in a development flow, refer to the following pages on the Altera® website: *SOPC Builder Component Development Flow Using the Component Editor Overview*. For information about Avalon® component interfaces, refer to *Avalon Component Interfaces Supported in the Component Editor Version 7.2 and Later*. For examples of changes to typical Avalon interfaces, refer to *Examples of Changes to Typical Avalon Interfaces for the Component Editor Version 7.2 and Later*. For information about upgrading components, refer to *Upgrading Your Component with SOPC Builder Component Editor Version 7.2 and Later*.

For information about the use of the component editor, see the following sections:

- “Starting the Component Editor” on page 6–2.
- “HDL Files Tab” on page 6–2.
- “Signals Tab” on page 6–3.
- “Interfaces Tab” on page 6–6.
- “Component Wizard Tab” on page 6–6.
- “Saving a Component” on page 6–8.
- “Editing a Component” on page 6–8.
- “Component Parameterization” on page 6–8.

 For more information about components, refer to the *Component Interface Tcl Reference* chapter in volume 4 of the *Quartus II Handbook*. For more information about the Avalon-MM interface, refer to the *Avalon Interface Specifications*.

## Component Hardware Structure

The component editor creates components with the following characteristics:

- A component has one or more interfaces. Typically, an *interface* means an Avalon® Memory-Mapped (Avalon-MM) master or slave or an Avalon Streaming (Avalon-ST) source or sink. You can also specify exported component signals that appear at the top-level of the SOPC Builder system, which can be connected to logic outside the SOPC Builder system. The component editor lets you build a component with any combination of Avalon interfaces, which include:
  - Avalon-MM master and slave
  - Avalon-ST source and sink
  - Avalon-MM tristate slave
  - Interrupt sender and receiver
  - Clock input and output
  - Nios II custom instruction master and slave interfaces
  - Conduit (for exporting signals to the top level)
- Each interface is comprised of one or more signals.
- The component can represent logic that is instantiated inside the SOPC Builder system, or can represent logic outside the system with an interface to it on the generated system.

## Starting the Component Editor

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

The component editor presents several tabs that group related settings. A message window at the bottom of the component editor displays warning and error messages.



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 Quartus® II online Help for additional information about the component editor.

You navigate through the tabs from left to right as you progress through the component creation process.

## HDL Files Tab

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

## Bottom-Up 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 table.

You can also use the component editor to define the interface to components outside the SOPC Builder 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 component editor analyzes the file by invoking the Quartus II Analysis and Elaboration module. The component editor analyzes signals and parameters declared for all modules in the top-level file. If the file is successfully analyzed, the component editor's **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 select 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. Files that turn on the **Sim** option are passed to ModelSim® for simulation. To add a synthesis-only file, turn off the **Sim** file option. You can add the Synopsis Design Constraint File (**.sdc**) for your component using the **Synth** option. Only files that you mark for **Synth** are added to the Quartus II IP File (**.qip**) for your project.



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 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 component 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 Memory-Mapped and Streaming interfaces, components typically have clock interfaces, interrupt interfaces, and perhaps a conduit interface for exported signals.

## 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
ats	Avalon-MM tristate slave
aso	Avalon-ST source
asi	Avalon-ST sink
cso	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

**Example 6-1** shows a Verilog HDL module declaration with signal names that infer two Avalon-MM slaves.

---

**Example 6-1.** Verilog HDL Module With Automatically Recognized Signal Names

---

```
module my_slave_irq_component (

    // Signals for Avalon-MM slave port "s1" with irq

    csi_clockreset_clk; //clockreset clock interface
    csi_clockreset_reset_n; //clockreset clock interface

    avs_s1_address; //s1 slave interface
    avs_s1_read; //s1 slave interface
    avs_s1_write; //s1 slave interface
    avs_s1_writedata; //s1 slave interface
    avs_s1_readdata; //s1 slave interface
    ins_irq0_irq; //irq0 interrupt sender interface
);

input csi_clockreset_clk;
input csi_clockreset_reset_n;
input [7:0]avs_s1_address;
input avs_s1_read;
input avs_s1_write;
input [31:0]avs_s1_writedata;
output [31:0]avs_s1_readdata;
output ins_irq0_irq;

/* Insert your logic here */

endmodule
```

---

## Templates for Interfaces to External Logic

You can use the **Create HDL Template** to generate an HDL template for the component. Then, you connect these signals outside of the SOPC Builder system. If your component uses an Avalon interface to interface outside of SOPC Builder, 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 Slave with Interrupt
- Avalon-MM Master
- Avalon-MM Master with Interrupt
- Avalon-ST Source
- Avalon-ST Sink

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 SOPC Builder connection panel. The interface name is also used to uniquely identify any signals that are ports on the top-level SOPC Builder 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 specified. 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 Memory-Mapped
- Avalon Memory-Mapped tristate
- Avalon Streaming
- Interrupts

If you convert a component from a **class.ptf** to a **\_hw.tcl** file, you may require three interfaces: a clock input, the Avalon slave, and an interrupt sender. A parameter in the interrupt sender must be set to reference the Avalon slave.

## Component Wizard Tab

The **Component Wizard** tab provides options that affect the presentation of your new component.

### Identifying Information

You can specify information that identifies the component as follows:

- **Folder**—Specifies the location of the component, determined by the location of the top-level HDL file.
- **Class Name**—Specifies the name used internally to store the component in the component library. The class name is stored in the SOPC Builder design file (**.sopc**). Use the class name when saving a system that contains an instance of this component. It is also the name you use for the component type when you create a system using a **.tcl** script. If you change the class name of a component, existing **.sopc** files that use the component may break.



SOPC builder uses the class name and version to find components. If two components with the same class name and version are available to SOPC builder at the same time, the behavior of SOPC builder is undefined.

- **Display Name**—Specifies the user-visible name for this component in SOPC Builder.
- **Version**—Specifies the version number of the component.
- **Group**—Specifies which group in SOPC Builder displays your component in the list of available components. If you enter a previously unused group name, SOPC Builder 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.
- Data sheet URL—Allows you to specify a URL for the datasheet. 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 a relative path using the following Tcl command:  

```
set_module_property DATASHEET_URL [get_module_property
MODULE_DIRECTORY]/<relative_path_to_hw.tcl>
```
- Parameters—Allows you to specify the parameters for creating the component, as described in the next section.

## Parameters

The **Parameters** table allows you to specify the user-configurable parameters for the component.

If the top-level module of the component HDL declares any parameters (*parameters* for Verilog HD or *generics* for VHDL), those parameters appear in the **Parameters** table. The parameters are presented to you when you create or edit an instance of your component. Using the **Parameters** table, you can specify whether or not each parameter is user-editable.

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

- Editable parameters cannot contain computed expressions.
- If a parameter **<N>** defines the width of a signal, the signal width must be of the form **<N-1>:0**.
- When a VHDL component is used in a Verilog HDL SOPC Builder system, or vice versa, numeric parameters must be 32-bit decimal integers. When passing other numeric parameter types, unpredictable results occur.

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




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

## 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 SOPC Builder. To edit a component, right-click it in the list of available components on the **System Contents** tab and click **Edit Component**.

 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.

## 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 6-2](#) shows a typical Tcl API script:

---

**Example 6-2.** Typical Software Assignment with Tcl API Scripting

---

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

---

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

## Component Parameterization

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

## Document Revision History

Table 6-2 shows the revision history for this chapter.

**Table 6-2.** Document Revision History (Sheet 1 of 2)

Date and Document Version	Changes Made	Summary of Changes
November 2009, v9.1.0	<ul style="list-style-type: none"> <li>■ No changes from the previous release.</li> </ul>	—
March 2009, v9.0.0	<ul style="list-style-type: none"> <li>■ Revised description of the <b>Create HDL Template</b> functionality and the Templates menu.</li> <li>■ Interfaces tab now includes waveforms that illustrate timing parameters.</li> <li>■ Added reference to <i>Component Interface Tcl Reference</i> chapter for detailed information about defining and displaying GUI parameters.</li> <li>■ Added data sheet URL to Component Wizard tab.</li> </ul>	Updated to reflect new functionality.
November 2008, v8.1.0	<ul style="list-style-type: none"> <li>■ Added information about new HDL template feature</li> <li>■ Changed page size to 8.5 x 11 inches</li> </ul>	—
May 2008, v8.0.0	<p>Extensive edits to this chapter, including:</p> <ul style="list-style-type: none"> <li>■ Chapter renumbered.</li> <li>■ Added new section on software assignments.</li> </ul>	—
October 2007, v7.2.0	Updated several paragraphs describing the latest GUI.	—
May 2007, v7.1.0	<p>Updated all sections to reflect significant functional differences in version 7.1.</p> <p>Added section “Changes to Component Editor in Version 7.1” on page 5-2.</p> <p>Updated section “Component Editor Output” and “Re-editing Components” to accommodate new component structure with 7.1 release.</p> <p>Updated Avalon terminology because of changes to Avalon technologies. Changed old “Avalon switch fabric” term to “system interconnect fabric.” Changed old “Avalon interface” terms to “Avalon Memory-Mapped interface.”</p> <p>Removed screen shots that simply reflect what user sees when using the tool without illustrating a particular point.</p> <p>Added Referenced Documents section which links to all referenced documents.</p> <p>Added statement that all simulation files, not just top-level file, must be added using the HDL files tab.</p>	The file structure of SOPC Builder components changed significantly in this release, which required substantial functional change to the component editor. This document changed significantly to reflect the functional changes. Updated to improve readability.
March 2007, v7.0.0	No change from previous release.	—
November 2006, v6.1.0	No change from previous release.	—
May 2006, v6.0.0	No change from previous release.	—

**Table 6-2.** Document Revision History (Sheet 2 of 2)

<b>Date and Document Version</b>	<b>Changes Made</b>	<b>Summary of Changes</b>
December 2005, v5.1.1	<ul style="list-style-type: none"> <li>■ Added section “Naming Signals for Automatic Type and Interface Recognition” on page 5-4.</li> <li>■ Added section “Templates for Interfaces to External Logic” on page 5-6.</li> <li>■ Clarified operation of the Save command.</li> <li>■ Updated all screenshots.</li> </ul>	—
October 2005, v5.1.0	No change from previous release.	—
May 2005, v5.0.0	Initial release.	—