

Introduction

This chapter describes the SOPC Builder component editor. The component editor provides a GUI to support the creation and editing of the `_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 that in your component hardware.
- Define the interfaces on the component and provide information about how the interface functions.
- 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 on using the component editor in a development flow, refer to *SOPC Builder Component Development Flow Using the Component Editor Overview*.



For information on 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 on upgrading components, refer to *Upgrading Your Component with SOPC Builder Component Editor Version 7.2 and Later*.

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

- To start the component editor, refer to *“Starting the Component Editor” on page 6–2*.
- For information about specifying HDL files that describe a component, refer to *“HDL Files Tab” on page 6–3*.
- For information about specifying interface signals, refer to *“Signals Tab” on page 6–4*.

- For information about specifying the Avalon-MM type of interface signals, refer to “[Interfaces Tab](#)” on page 6–6.
- For information about specifying parameters, refer to “[Component Wizard Tab](#)” on page 6–6.
- To save a component, refer to “[Saving a Component](#)” on page 6–7.
- For information about changing a component after it has been saved, refer to “[Editing a Component](#)” on page 6–8.
- For information about the UI used to parameterize instances of your component, refer to “[Component GUI](#)” 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-MM master or slave. 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
 - 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 table on the **HDL Files** tab specifies all the HDL files in the component. You 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 SOPCBuilder system. In this case, you do not provide HDL files. Instead, you use the component editor to manually 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 checkboxes for synthesis and simulation. By default, all files are added with both checked. To add a simulation-only file, uncheck the synthesis checkbox for that file. To add a synthesis-only file, uncheck the simulation file. You use the top-level file checkbox to select the top-level file for synthesis. You can use the up and down arrows to specify the HDL file analysis order.



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.

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.

If the component does not use an HDL file (a non-HDL based component) to interface to external logic that is Avalon compatible, you must manually add the signals that comprise the interface to the external logic. The **Interface** list also allows creation of a new interface. You can also use the Templates menu to quickly add typical interface signals to your signal list.

Each signal must belong to an interface and be assigned a legal signal type for that interface.

You assign each signal to an interface using the **Interface** list. 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 follow naming conventions. [Table 6-1](#) lists the signal naming conventions.

Type of Signal	Name Convention
Signal associated with a specific interface	<code><interface type>_<interface name>_<signal type>[_n]</code>

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. You can append `_n` to indicate an active-low signal. [Table 6-2](#) lists the valid values for `Interface Type`.

Value	Meaning
avs	Avalon-MM slave
avm	Avalon-MM master
ats	Avalon-MM tristate slave

Table 6–2. Valid Values for <Interface Type> (Part 2 of 2)

Value	Meaning
atm	Avalon-MM Tristate Master
aso	Avalon-ST Source
asi	Avalon-ST Sink
cso	Clock Output
csi	Clock Input
inr	Interrupt Receiver
ins	Interrupt Sender
cos	Coe
coe	Coe
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

If you create an interface to external logic, you can use the Templates menu in the component editor to add a set of signals, such as the following templates:

- 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 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 also appears in the SOPC Builder connection panel. The interface name is also used to uniquely identify any signals that are exposed on the top-level SOPC Builder system.

The **Interfaces** tab also 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.

If you convert an older Avalon-MM slave to the new model, 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 internal name of the component. Use the internal name when saving a system that contains an instance of this component, and when it is the name you use for the component type when you create a system using a script.
- **Display Name Version**—Specifies the user-visible name for this component in SOPC Builder.

- **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 associate the component with a file path relative to the component. The icon can be a **.jpg**, **.gif**, or **.png** file.
- **Parameters**—Allows you to specify the parameters for creating the component, as described below.

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 HDL, HDL, 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.

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 *<name of top-level file>_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.

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. The following example 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 result is that the assignments go into the **.sopcinfo** file, available for use for downstream components.

Component GUI

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

Referenced Documents

This chapter references the following documents:

- *Avalon Component Interfaces Supported in the Component Editor Version 7.2*
- *Avalon Interface Specifications*
- *Component Interface Tcl Reference* chapter in volume 4 of the *Quartus II Handbook*
- *Examples of Changes to Typical Avalon Interfaces for the Component Editor Version 7.2 and Later*
- *Nios II Software Developer's Handbook*

- *SOPC Builder Components* chapter in volume 4 of the *Quartus II Handbook*
- *SOPC Builder Component Development Flow Using the Component Editor Overview*
- *Upgrading Your Component with SOPC Builder Component Editor Version 7.2 and Later*

Document Revision History

Table 6–3 shows the revision history for this chapter.

Date and Document Version	Changes Made	Summary of Changes
May 2008, v8.0.0	Extensive edits to this chapter, including: <ul style="list-style-type: none"> ● Chapter renumbered. ● Added new section on software assignments. 	—
October 2007, v7.2.0	Updated several paragraphs describing the latest GUI.	—
May 2007, v7.1.0	Updated all sections to reflect significant functional differences in version 7.1. Added section “Changes to Component Editor in Version 7.1” on page 5–2. Updated section “Component Editor Output” and “Re-editing Components” to accommodate new component structure with 7.1 release. 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.” Removed screen shots that simply reflect what user sees when using the tool without illustrating a particular point. Added Referenced Documents section which links to all referenced documents. Added statement that all simulation files, not just top-level file, must be added using the HDL files tab.	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.	—

<i>Table 6–3. Document Revision History</i>		
Date and Document Version	Changes Made	Summary of Changes
May 2006, v6.0.0	No change from previous release.	—
December 2005, v5.1.1	<ul style="list-style-type: none"> ● Added section “Naming Signals for Automatic Type and Interface Recognition” on page 5–4. ● Added section “Templates for Interfaces to External Logic” on page 5–6. ● Clarified operation of the Save command. ● Updated all screenshots. 	—
October 2005, v5.1.0	No change from previous release.	—
May 2005, v5.0.0	Initial release.	—