4. SOPC Builder Components

QII54004-9.0.0

An SOPC Builder *component* is a hardware design block available within SOPC Builder that can be instantiated in an SOPC Builder system. This chapter defines SOPC Builder components, with emphasis on the structure of custom components.

A component includes the following:

- 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 any parameters that specify the structure of the component logic and component.
- A GUI for configuring an instance of the component in SOPC Builder.
- Scripts and other information SOPC Builder requires to generate the HDL files for the component and integrate the component instance into the SOPC Builder system.
- Other component-related information, such as reference to software drivers, necessary for development steps downstream of SOPC Builder.

This chapter discusses the design flow for new and classic custom-defined SOPC Builder components, in the following sections:

Component Providers

SOPC Builder components can be obtained from many providers, including the following:

- The components automatically installed with the Quartus® II software.
- Third-party IP developers can provide IP blocks as SOPC Builder-ready components, including software drivers and documentation. A list of third-party components can be found in SOPC Builder by clicking **IP MegaStore** on the Tools menu.
- Altera development kits, such as the Nios® II Development Kit, can provide SOPC Builder components as features.
- You can use the SOPC Builder component editor to convert your own HDL files into custom components.

The GUI interfaces for classic components run slower in newer versions of SOPC Builder when you add or modify your component settings. These components are marked by a gray dot in the **System Contents** tab. You have better performance when you upgrade to the Hardware Component Description File (**_hw.tcl**) component format in newer versions of SOPC Builder. These components are marked by a green dot.

For more information about the **_hw.tcl** file, refer to the *Component Editor* chapter in volume 4 of the *Quartus II Handbook*.

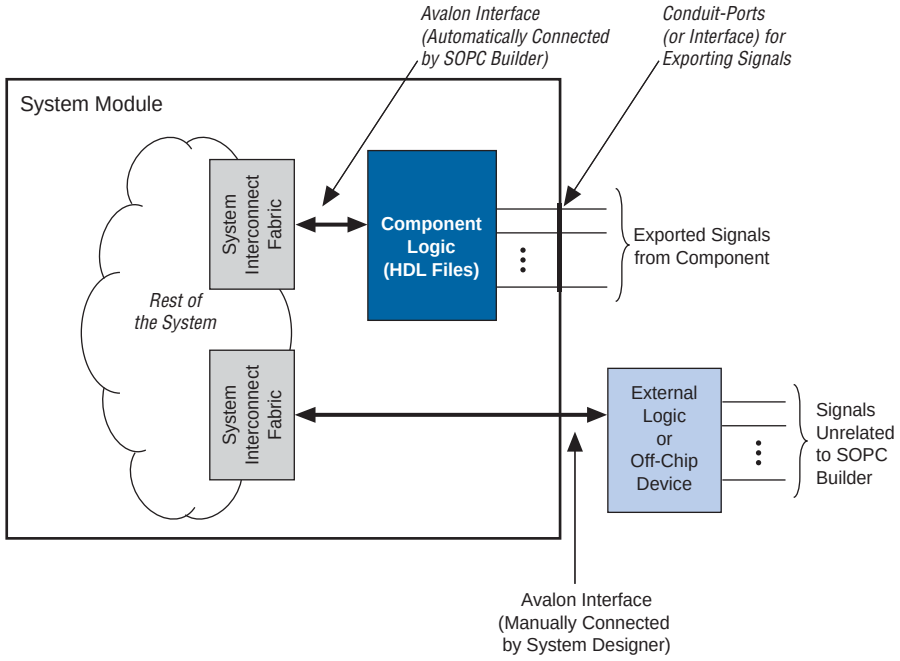
Component Hardware Structure

There are the following types of components in an SOPC Builder system, based on where the associated component logic resides:

- Components that include their associated logic inside the SOPC Builder system
- Components that interface to logic outside the SOPC Builder system

Figure 4–1 shows an example of both types of components.

Figure 4–1. Component Logic Inside and Outside the SOPC Builder System



Components Inside the SOPC Builder System

For components that are instantiated inside the SOPC Builder system, the component defines its logic in an associated HDL file. During system generation, SOPC Builder instantiates the component and connects it to the rest of the system. The component can include exported signals in conduit interfaces. Conduit interfaces become ports on the system, so they can be connected to logic outside the SOPC Builder system in the board-level schematic.

For more information about conduit interfaces, refer to the *Conduit Interfaces* chapter in the *Avalon Interface Specifications*.

In general, components connect to the system interconnect fabric using the Avalon® Memory-Mapped (Avalon-MM) interface or the Avalon Streaming (Avalon-ST) interface. A single component can provide more than one Avalon port. For example, a component might provide an Avalon-ST source port for high-throughput data, in addition to an Avalon-MM slave for control.

Static HDL Components

You can define SOPC Builder components whose parameters are all assigned values during the initial editing session. Examples of parameters whose values are typically known at instantiation time are address and data widths and FIFO depths. If all of a component's parameters are assigned when it is instantiated, the HDL for the component is *static*. SOPC Builder automatically generates the top-level HDL wrapper file to apply parameter values to your component.

Dynamic HDL Components

You can also create SOPC Builder components whose parameters are defined by a generation callback. Examples of parameters that might be assigned during generation callback are baud rate and output directory. When you create components that include parameters defined using a generation callback, you must provide a custom generation callback routine to create the top-level wrapper for your component.

For more information about defining your own generation program, refer to the *Generation Callback* section in the *Component Interface Tcl Reference* chapter in volume 4 of the *Quartus II Handbook*.

Components Outside the SOPC Builder System

For components that interface to external logic or off-chip devices with Avalon-compatible signals outside the SOPC Builder system, the component files describe only the interface to the external logic. During system generation, SOPC Builder exports an interface for the component in the top-level SOPC Builder system. You must manually connect the signals at the top-level of SOPC Builder to pins or logic defined outside the system that already has Avalon-compatible signals.

Exported Connection Points—Conduit Interfaces

Conduit interfaces are brought to the top level of the system as additional ports. Exported signals are usually either application-specific signals or the Avalon interface signals.

Application-specific signals are exported to the top level of the system by the conduit interface(s) defined in the **_hw.tcl** file. These are I/O signals in a component's HDL logic that are not part of any Avalon interfaces and connect to an external device, for example DDR SDRAM memory, or logic defined outside of the SOPC Builder system. You use conduit interfaces to connect application-specific signals of the external device and the SOPC Builder system.

You can also export the Avalon interfaces to manually connect them to external devices or logic defined outside a system with Avalon-compatible signals. This method allows a direct connection to the Avalon interface from any device that has Avalon-compatible signals. You can also export the Avalon interface in either an HDL file using conduit interfaces, or in the **_hw.tcl** file without an HDL file.

You export the Avalon interface signals as an HDL file with simple wire connections in the HDL description. The Avalon interface port signals are directly connected to external I/O signals in the HDL description. The conduit interface in the **_hw.tcl** file exports the external I/O signals to the top level of the system.

In the **_hw.tcl** file, no HDL files are specified and only the Avalon signals and interface ports are declared in the file.

SOPC Builder Component Search Path

Each time SOPC Builder starts, it searches for component files. The components that SOPC Builder finds are displayed in the list of available components on the SOPC Builder **System Contents** tab. When you launch SOPC Builder certain directories are searched for two kinds of files:

- **_hw.tcl** files. Each **_hw.tcl** file defines a single component.
- IP Index (**.ipx**) files. Each file indexes a collection of available components.

In general, **.ipx** files facilitate faster startup for SOPC Builder and other tools because fewer files need to be read and analyzed.

Some directories are searched recursively; others 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.

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

In SOPC Builder, you can extend the default search path by including additional directories by clicking **Options**, then clicking **IP Search Path** and **Add**. These additional paths apply to all projects; that is, the paths are global to the current version of SOPC Builder.

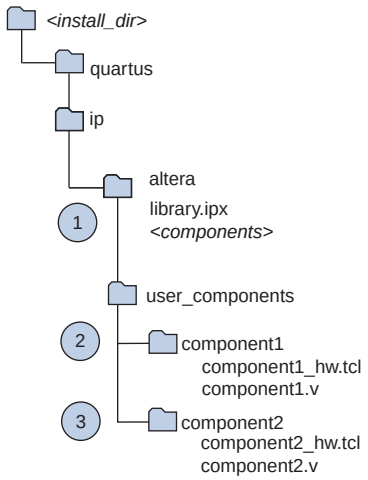
Installing Additional Components

There are two additional ways to make your components available to SOPC Builder projects. The following sections describe these methods.

Copy to the IP Root Directory

The simplest strategy is to copy your components into the standard IP directory provided by Altera. Figure 4–2 illustrates this approach.

Figure 4–2. User Library Included In Subdirectory of $IP_ROOTDIR



In Figure 4–2, the circled numbers identify three steps of the algorithm that SOPC follows during initialization. These steps are explained in the following paragraphs.

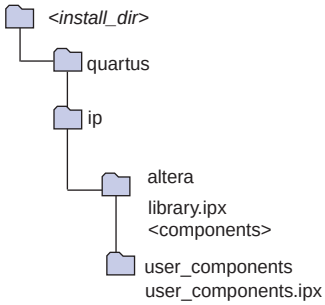
1. SOPC Builder recursively searches the <*install_dir*>**/ip/** directory by default. It finds the file in the `altera` subdirectory, which tells it about all of the Altera components. **library.ipx** includes listings for all components found in its subdirectories. The recursive search stops when SOPC Builder finds this **.ipx** file.
2. As part of its recursive search, SOPC Builder also looks in the adjacent **user_components** directory. One level down SOPC Builder finds the **component1** directory, which contains **component1_hw.tcl**. SOPC Builder finds that component stops the recursive descent.
3. SOPC Builder then searches in the adjacent **component2** directory, which includes **component2_hw.tcl**. If SOPC Builder finds that component, the recursive descent stops.

If you save your **.ipx** file in the <*install_dir*>**/ip/** directory, SOPC Builder finds your **.ipx** file and stops. SOPC Builder does not conduct the search just described.

Reference Components in an .ipx File

A second approach is to specify your IP directory in a **user_components.ipx** file under <*install_dir*>**/ip** path. Figure 4–3 illustrates this approach.

Figure 4–3. Specifying A User .ipx directory



The **user_components.ipx** file includes a single line of code redirecting SOPC Builder to the location of the user library. Example 4–1 shows the code for this redirection.

Example 4–1. Redirect to User Library

```
<library>
  <path path="c:/<user_install_dir>/user_ip/**/*" />
<library>
```

For both of these approaches, if you install a new version of the Quartus II software, you must also update the installation to include your libraries.

Understanding IPX File Syntax

An **.ipx** file is an XML file whose top-level element is <*library*> with a <*path*> subelements are <*path*> and <*component*>. Altera recommends that you only add or edit the <*path*> subelement.

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**—SOPC Builder 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 SOPC Builder is less than if SOPC Builder must discover the files in a directory. Example 4–2 shows two <*component*> elements. Note that the paths for file names are specified relative to the **.ipx** file.

Example 4–2. Component Elements

```
<library>
  <component
    name="An SOPC Component"
    displayName="SOPC Component"
    version="2.1"
    file="./components/sopc_component/sc_hw.tcl"
   />
  <component
    name="legacy_component"
    displayName="Legacy Component (Classic Edition!)"
    version="0.9"
    file="./components/legacy/old_component/class.ptf"
   />
<library>
```

Upgrading from Earlier Versions

If you specified a custom search path in SOPC Builder prior to v8.1 using the **IP Search Path** option, or by adding it to the `$SOPC_BUILDER_PATH`, SOPC Builder automatically adds those directories to the **user_components.ipx** file in your home directory. This file is saved in *<home_dir>***/altera.quartus/ip/8.1/ip_search_path/user_components.ipx.** Go to the **IP Search Path** option in the **Options** dialog box to see the directories listed here**.**

Component Structure

Most components are defined with a **_hw.tcl** file, a text file written in the Tcl scripting language that describes the components in to SOPC Builder. You can add a component to SOPC Builder by either writing a Tcl description or you can use the component editor to generate an automatic Tcl description of it. This section describes the structure of Tcl components and how they are stored.

For details about the SOPC Builder component editor, refer to the *Component Editor* chapter in volume 4 of the *Quartus II Handbook*. For details about the SOPC Builder Tcl commands, refer to the *Component Interface Tcl Reference* chapter in volume 4 of the *Quartus II Handbook*.

Component Description File (_hw.tcl)

A Tcl component consists of:

- A component description file, which is a Tcl file with file name of the form *<entity name>***_hw.tcl.**
- Verilog HDL or VHDL files that define the top-level module of the custom component (optional).

The **_hw.tcl** file defines everything that SOPC Builder requires about the name and location of component design files.

The SOPC Builder component editor 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, SOPC Builder automatically saves the earlier version as **_hw.tcl~**.

For more information about the information that you can include in the **_hw.tcl** file, refer to the *Component Interface Tcl Reference* chapter in volume 4 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.
- You are not required to create a special sub-directory for component HDL files. However, you are required to follow the naming conventions given here.
  - <*component_dir*>**/**
  - <*name*>**_hw.tcl**
  - <*name*>**.v** or **.vhd**
  - <*name*>**_sw.tcl**
- <*software*>**/**—a directory that contains software drivers or libraries related to the component, if any. Altera recommends that the software directory be subdirectory of the directory that contains the **_hw.tcl** file.

For information on writing a device driver or software package suitable for use with the Nios® II IDE design flow, 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.

Classic Components in SOPC Builder

If you use classic components created with an earlier version of SOPC Builder, read through this section to familiarize yourself with the differences. This document uses the term *classic components* to refer to class.ptf-based components created with a previous version of the Quartus II software. If you do not use classic components, skip this section.

Classic components are compatible with newer versions of SOPC Builder, but be aware of the following caveats:

- Classic components configured with the **More Options** tab in SOPC Builder, such as complex IP components provided by third-party IP developers, are not supported in the Quartus II software in version 7.1 and beyond. If your component has a bind program, you cannot use the component without recreating it with the component editor or with Tcl scripting.
- To make changes to a classic component with the component editor, you must first upgrade the component by editing the classic component and saving it in the **_hw.tcl** component format in the component editor.

Referenced Documents

This chapter references the following documents:

- *Component Interface Tcl Reference* chapter in volume 4 of the *Quartus II Handbook*
- *Component Editor* chapter in volume 4 of the *Quartus II Handbook*
- *Conduit Interfaces* chapter in the *Avalon Interface Specifications*
- *Embedded Peripherals* section in volume 5 of the *Quartus II Handbook*
- *Hardware Abstraction Layer* section of the *Nios II Software Developer's Handbook*
- *Nios II Software Build Tools Reference* chapter of the *Nios II Software Developer's Handbook*

Document Revision History

Table 4–1 shows the revision history for this chapter.

Table 4–1. Document Revision History

Date and Document Version	Changes Made	Summary of Changes
March 2009, v9.0.0	■ Added 2 paragraphs introducing custom generations for dynamic components.	Updated component descriptions.
November 2008, v8.1.0	■ Revised section on component search paths. ■ Added meaning of green and gray dots next to components on the **System Contents** tab. ■ Changed page size to 8.5 x 11 inches	Revised to reflect changes to the component search path in 8.1.
May 2008, v8.0.0	■ Added paragraph about IP Search Path.	—

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