9. Building Memory Subsystems Using SOPC Builder

QII54006-6.0.0

Introduction

Most systems generated with SOPC Builder require memory. For example, embedded processor systems require memory for software code, while digital signal processing (DSP) systems require memory for data buffers. Many systems use multiple types of memories. For example, a processor-based DSP system can use off-chip SDRAM to store software code, and on-chip RAM for fast access to data buffers. You can use SOPC Builder to integrate almost any type of memory into your system.

This chapter describes the process for building a memory subsystem as part of a larger system created with SOPC Builder. This chapter focuses on the kinds of memory most commonly used in SOPC Builder systems:

- On-chip RAM and ROM
- EPCS serial configuration devices
- SDRAM
- Off-chip RAM and ROM, such as SRAM and common flash interface (CFI) flash memory

This chapter assumes that you are familiar with the following:

- Creating FPGA designs and making pin assignments with the Quartus® II software. For details, see the *Introduction to Quartus II Manual*.
- Building simple systems with SOPC Builder. For details, see the *Introduction to SOPC Builder* and *Tour of the SOPC Builder User Interface chapters* in volume 4 of the *Quartus II Handbook*.
- SOPC Builder components. For details, see the *SOPC Builder Components* chapter in volume 4 of the *Quartus II Handbook*.
- Basic concepts of the Avalon® interface. You do not need extensive knowledge of the Avalon interface, such as transfer types or signal timing. However, to create your own custom memory subsystem with external memories, you need to understand the Avalon interface. For details, see the *Avalon Switch Fabric* chapter in volume 4 of the *Quartus II Handbook* and the *Avalon Interface Specification*.

Example Design

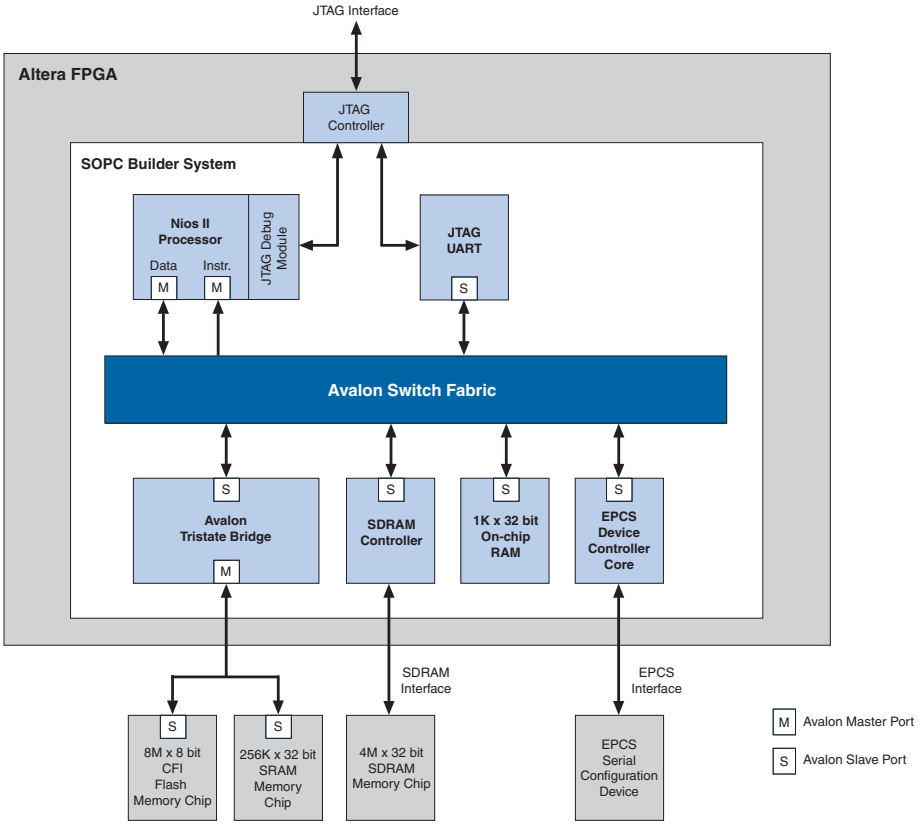
This chapter demonstrates the process for building a system that contains one of each type memory as shown in Figure 9–1. Each section of the chapter builds on previous sections, culminating in a complete system.

By following the example design through this chapter, you will learn how to create a complete memory subsystem for your own custom system. The memory components in the example design are independent. For a custom system, you can instantiate exactly the memories you need, and skip the memories you don't need. Furthermore, you can create multiple instantiations of the same type of memory, limited only by on-chip memory resources or FPGA pins to interface with off-chip memory devices.

Example Design Structure

Figure 9–1 shows a block diagram of the example system.

Figure 9–1. Example Design Block Diagram



In Figure 9–1, all blocks shown below the Avalon switch fabric comprise the memory subsystem. For demonstration purposes, this system uses a Nios® II processor core to master the memory devices, and a JTAG UART core to communicate with the processor. However, the memory subsystem could be connected to any master component, either on-chip or off-chip.

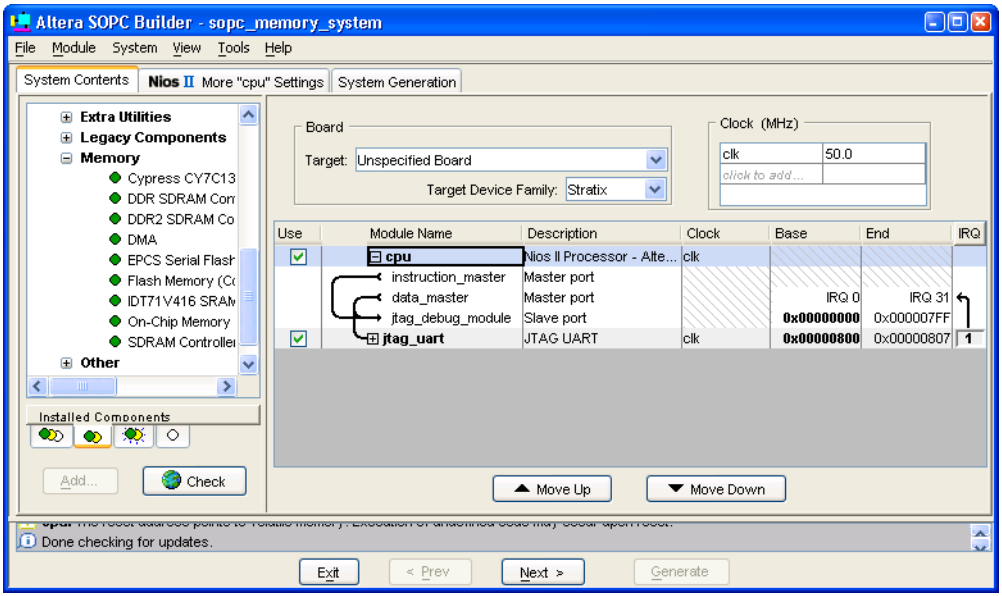
Example Design Starting Point

The following elements comprise the example design:

- A Quartus II project named **quartus2_project**.A block diagram file (BDF) named **toplevel_design**. **toplevel_design** is the top-level design file for **quartus2_project**. **toplevel_design** instantiates the SOPC Builder system module, as well as other pins and modules required to complete the design.
- An SOPC Builder system named **sopc_memory_system**. **sopc_memory_system** is a subdesign of **toplevel_design**. **sopc_memory_system** instantiates the memory components and other SOPC Builder components required for a functioning system module.

The starting point for this chapter assumes that **quartus2_project** already exists, that **sopc_memory_system** has been started in SOPC Builder, and that the Nios II core and the JTAG UART core are already instantiated. This example design uses the default settings for the Nios II/s core and the JTAG UART core; these settings do not affect the rest of the memory subsystem. Figure 9–2 shows the starting point in SOPC Builder.

Figure 9–2. Starting Point for the Example Design



All sections in this chapter build on this starting point.

Hardware & Software Requirements

To build a memory subsystem similar to the example design in this chapter, you need the following:

- Quartus II Software version 5.0 or higher –Both Quartus II Web Edition and the fully licensed version support this design flow.
- Nios II Embedded Design Suite (EDS) version 5.0 or higher –Both the evaluation edition and the fully licensed version support this design flow. The Nios II EDS provides the SOPC Builder memory components described in this chapter. It also provides several complete example designs which demonstrate a variety of memory components instantiated in working systems.

The Quartus II Web Edition software and the Nios II EDS, Evaluation Edition are available free for download from the Altera® website. Visit **www.altera.com/download**.

This chapter does not go as far as downloading and verifying a working system in hardware. Therefore, there are no hardware requirements for the completion of this chapter. However, the example memory subsystem has been tested in hardware.

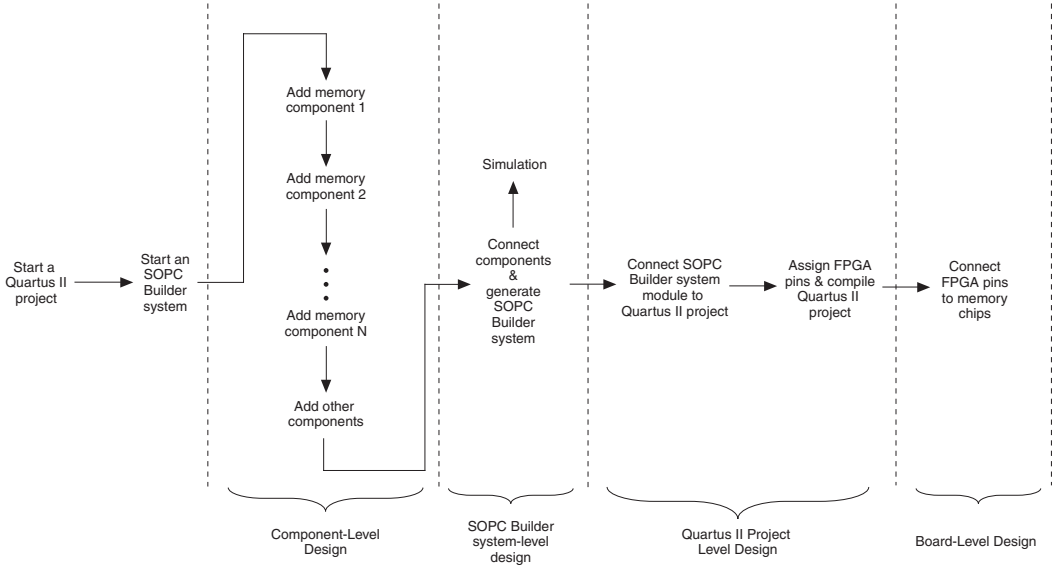
Design Flow

This section describes the design flow for building memory subsystems with SOPC Builder.

The design flow for building a memory subsystem is similar to other SOPC Builder designs. After starting a Quartus II project and an SOPC Builder system, there are five steps to completing the system, as shown in Figure 9–3:

1. Component-level design in SOPC Builder
2. SOPC Builder system-level design
3. Simulation
4. Quartus II project-level design
5. Board-level design

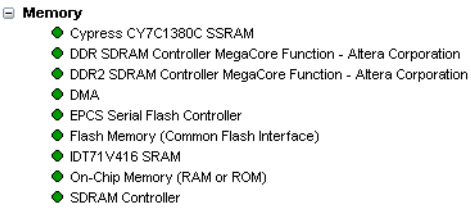
Figure 9–3. Design Flow



Component-Level Design in SOPC Builder

In this step, you specify which memory components to use, and you configure each component to meet the needs of the system. All memory components are available from the **Memory** category in the SOPC Builder list of available components, shown in Figure 9–4.

Figure 9–4. List of Available Memory Components in SOPC Builder



SOPC Builder System-Level Design

In this step, you connect components together and configure the SOPC Builder system as a whole. Similar to the process for adding non-memory SOPC Builder components, you use the SOPC Builder System Contents tab to do the following:

- Rename the component instance (optional).
- Connect the memory component to master ports in the system. Each memory component must be connected to at least one master port.
- Assign a base address.
- Assign a clock domain. A memory component can operate on the same or different clock domain as the master port(s) that access it.

Simulation

In this step, you verify the functionality of the SOPC Builder system module. For systems with memories, this step depends on simulation models for each of the memory components, in addition to the system testbench generated by SOPC Builder. See "Simulation Considerations" on page 9–7.

Quartus II Project-Level Design

In this step, you integrate the SOPC Builder system module with the rest of the Quartus II project. This step includes wiring the system module to FPGA pins, and wiring the system module to other design blocks (such as other HDL modules) in the Quartus II project.

In the example design in this chapter, the SOPC Builder system module comprises the entire FPGA design. There are no other design blocks in the Quartus II project.

Board-Level Design

In this step, you connect the physical FPGA pins to memory devices on the board. If the SOPC Builder system interfaces with off-chip memory devices, then you must make board-level design choices.

Simulation Considerations

SOPC Builder can automatically generate a testbench for register transfer level (RTL) simulation of the system. This testbench instantiates the system module and can also instantiate memory models for external memory components. The testbench is plain-text hardware description

language (HDL), located at the bottom of the top-level system module HDL design file. To explore the contents of the auto-generated testbench, open the top-level HDL file, and search on keyword `test_bench`.

Generic Memory Models

The memory components described in this chapter, except for the SRAM, provide generic simulation models. Therefore, it is very easy to simulate an SOPC Builder system with memory components immediately after generating the system.

The generic memory models store memory initialization files, such as Data [file name extension] (**.dat**) and Hexadecimal (**.hex**) files, in a directory named *<Quartus II project directory>/<SOPC Builder system name>*_**sim**. When generating a new system, SOPC Builder creates empty initialization files. You can manually edit these files to provide custom memory initialization contents for simulation.

For Nios II processor users, the Nios II integrated development environment (IDE) generates initialization contents automatically.

Vendor-Specific Memory Models

You can also manually connect vendor-specific memory models to the system module. In this case, you must manually edit the testbench and connect the vendor memory model. You might also need to edit the vendor memory model slightly for time delays. The SOPC Builder testbench assumes zero delay.

There are special sections of the system module design file that you can edit safely. You must edit only these sections, because SOPC Builder overwrites the rest of the system module every time you generate the system. These sections are marked by the following text:

Verilog HDL

```
// <ALTERA_NOTE> CODE INSERTED BETWEEN HERE
//  add your signals and additional architecture here
// AND HERE WILL BE PRESERVED </ALTERA_NOTE>
```

VHDL

```
-- <ALTERA_NOTE> CODE INSERTED BETWEEN HERE
--add your component and signal declaration here
-- AND HERE WILL BE PRESERVED </ALTERA_NOTE>
```

On-Chip RAM & ROM

Altera FPGAs include on-chip memory blocks, that can be used as RAM or ROM in SOPC Builder systems. On-chip memory has the following benefits for SOPC Builder systems:

- On-chip memory has fast access time, compared to off-chip memory.
- SOPC Builder automatically instantiates on-chip memory inside the system module, so you do not have to make any manual connections.
- Certain memory blocks can have initialized contents when the FPGA powers up. This feature is useful, for example, for storing data constants or processor boot code.

FPGAs have limited on-chip memory resources, which limits the maximum practical size of an on-chip memory to approximately one megabyte in the largest FPGA family.

Component-Level Design for On-Chip Memory

In SOPC Builder you instantiate on-chip memory by adding an **On-chip Memory (RAM or ROM)** component. The configuration wizard for the **On-chip Memory (RAM or ROM)** component has the following options: **Memory Type**, **Size**, and **Read Latency**.

Memory Type

The **Memory Type** options define the structure of the on-chip memory.

- **RAM (writeable)** – This setting creates a readable and writeable memory.
- **ROM (read only)** – This setting creates a read-only memory.
- **Dual-Port Access** – Turning on this setting creates a memory component with two slave ports, which allows two master ports to access the memory simultaneously.
- **Block Type** – This setting forces the Quartus II software to use a specific type of memory block when fitting the on-chip memory in the FPGA. The following choices are available:
  - **Automatic** – This setting allows the Quartus II software to choose the most appropriate memory resource.
  - **M512** – This setting forces the Quartus II software to use M512 blocks.
  - **M4K** – This setting forces the Quartus II software to use M4K blocks.
  - **M-RAM** – This setting forces the Quartus II software to use M-RAM blocks. The 64 Kbit M-RAM blocks are appropriate for larger RAM data buffers. However, M-RAM blocks do not allow pre-initialized contents at power up.

Size

The **Size** options define the size and width of the memory.

- **Memory Width** – This setting determines the data width of the memory. The available choices are 8, 16, 32, 64, or 128 bits. Assign **Memory Width** to match the width of the master port that accesses this memory the most frequently or has the most critical timing requirements.
- **Total Memory Size** – This setting determines the total size of the on-chip memory block. The total memory size must be less than the available memory in the target FPGA.

Read Latency

On-chip memory components use synchronous, pipelined Avalon slave ports. Pipelined access improves f_{MAX} performance, but also adds latency cycles when reading the memory. The **Read Latency** option allows you to specify the number of read latency cycles required to access data. If the **Dual-Port Access** setting is turned on, you can specify a different read latency for each slave port.

SOPC Builder System-Level Design for On-Chip Memory

There are not many SOPC Builder system-level design considerations for on-chip memories. See "SOPC Builder System-Level Design" on page 9–7.

When generating a new system, SOPC Builder creates a blank initialization file in the Quartus II project directory for each on-chip memory that can power up with initialized contents. The name of this file is <*Name of memory component*>**.hex**.

Simulation for On-Chip Memory

At system generation time, SOPC Builder generates a simulation model for the on-chip memory. This model is embedded inside the system module, and there are no user-configurable options for the simulation testbench.

You can provide memory initialization contents for simulation in the file <*Quartus II project directory*>/<*SOPC Builder system name*>**_sim/**<*Memory component name*>**.dat**.

Quartus II Project-Level Design for On-Chip Memory

The on-chip memory is embedded inside the SOPC Builder system module, and therefore there are no signals to connect to the Quartus II project.

To provide memory initialization contents, you must fill in the file <*Name of memory component*>**.hex.** The Quartus II software recognizes this file during design compilation and incorporates the contents into the configuration files for the FPGA.

For Nios II processor users, the Nios II integrated development environment (IDE) generates memory initialization file automatically.

Board-Level Design for On-Chip Memory

The on-chip memory is embedded inside the SOPC Builder system module, and therefore there is nothing to connect at the board level.

Example Design with On-Chip Memory

This section demonstrates adding a 4 Kbyte on-chip RAM to the example design. This memory uses a single slave port with read latency of one cycle.

Figure 9–5 shows the **On-Chip Memory (RAM or ROM)** configuration wizard settings for the example design.

Figure 9–5. On-Chip Memory (RAM or ROM) Configuration Wizard

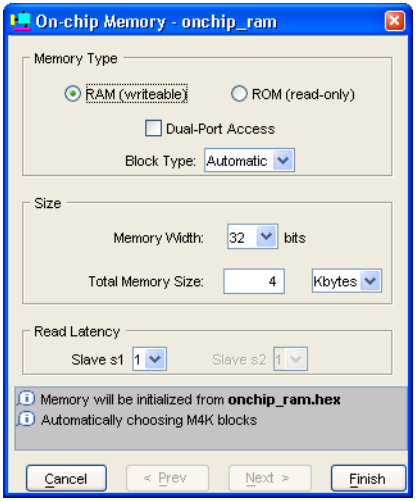
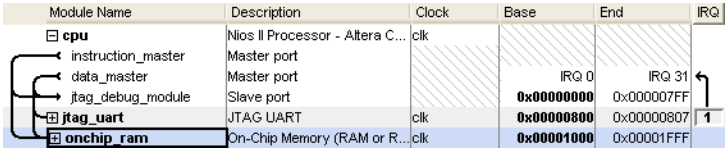


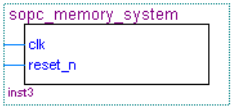
Figure 9–6 shows the SOPC Builder system after adding an instance of the on-chip memory component, renaming it to `onchip_ram`, and assigning it a base address.

Figure 9–6. SOPC Builder System with On-Chip Memory



For demonstration purposes, Figure 9–7 shows the result of generating the system module at this stage. (In a normal design flow, you generate the system only after adding all system components.)

Figure 9–7. System Module with On-Chip Memory



Because the on-chip memory is contained entirely within the system module, **sopc_memory_system** has no I/O signals associated with **onchip_ram**. Therefore, you do not need to make any Quartus II project connections or assignments for the on-chip RAM, and there are no board-level considerations.

EPCS Serial Configuration Device

Many systems use an Altera EPCS serial configuration device to configure the FPGA. Altera provides the EPCS device controller core, which allows SOPC Builder systems to access the memory contents of the EPCS device. This feature provides flexible design options:

- The FPGA design can reprogram its own configuration memory, providing a mechanism for in-field upgrades.
- The FPGA design can use leftover space in the EPCS as nonvolatile storage.

Physically the EPCS device is a serial flash memory device, which has slow access time. Altera provides software drivers to control the EPCS core for the Nios II processor only. Therefore, EPCS controller core features are available only to SOPC Builder systems that include a Nios II processor.

For further details on the features and usage of the EPCS device controller core, see the *EPCS Device Controller Core with Avalon Interface* chapter in volume 5 of the *Quartus II Handbook*.

Component-Level Design for an EPCS Device

In SOPC Builder you instantiate an EPCS controller core by adding an **EPCS Serial Flash Controller** component. There is only one setting for this component: **Reference Designator**. When targeting a board that declares a reference designator for the EPCS device, the **Reference Designator** setting is fixed.

SOPC Builder uses reference designators to specify a unique identifier for flash memory devices on the board. This convention is a requirement of the Nios II EDS, specifically the Nios II Flash Programmer utility.

For details, see the *Nios II Flash Programmer User Guide*.

SOPC Builder System-Level Design for an EPCS Device

There are not many SOPC Builder system-level design considerations for EPCS devices:

- Assign a base address.
- Set the IRQ connection to **NC** (disconnected). The EPCS controller hardware is capable of generating an IRQ. However, the Nios II driver software does not use this IRQ, and therefore you can leave the IRQ signal disconnected.

There can only be one EPCS controller core per FPGA, and the instance of the core is always named `epcs_controller`.

Simulation for an EPCS Device

The EPCS controller core provides a limited simulation model:

- Functional simulation does not include the FPGA configuration process, and therefore the EPCS controller does not model the configuration features.
- The simulation model does not support read and write operations to the flash region of the EPCS device.
- A Nios II processor can boot from the EPCS device in simulation. However, the boot loader code is different during simulation. The EPCS controller boot loader code assumes that all other memory simulation models are pre-initialized, and therefore the boot load process is unnecessary. During simulation, the boot loader simply forces the Nios II processor to jump to start, skipping the boot load process.

Verification in hardware is the best way to test features related to the EPCS device.

Quartus II Project-Level Design for an EPCS Device

The Quartus II software automatically connects the EPCS controller core in the SOPC Builder system to the dedicated configuration pins on the FPGA. This connection is invisible to the user. Therefore there are no EPCS-related signals to connect in the Quartus II project.

Board-Level Design for an EPCS Device

You must connect the EPCS device to the FPGA as described in the Altera *Configuration Handbook*. No other connections are necessary.

Example Design with an EPCS Device

This section demonstrates adding an EPCS device controller core to the example design.

Figure 9–8 shows the **EPCS Serial Flash Controller** configuration wizard settings for the example design. In this example, the target board declares a reference designator U59 for the EPCS device on the board.

Figure 9–8. EPCS Serial Flash Controller Configuration Wizard

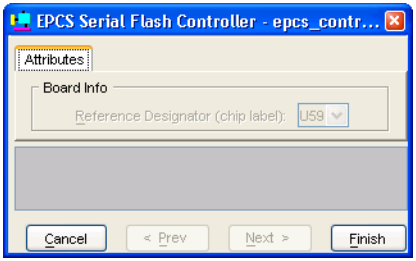
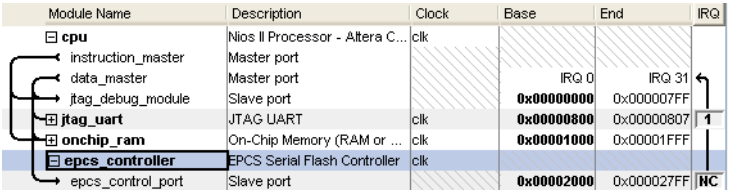


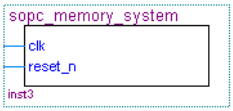
Figure 9–9 shows the SOPC Builder system after adding an instance of the EPCS controller core and assigning it a base address.

Figure 9–9. SOPC Builder System with EPCS Device



For demonstration purposes only, Figure 9–10 shows the result of generating the system module at this stage.

Figure 9–10. System Module with EPCS Device



Because the Quartus II software automatically connects the EPCS controller core to the FPGA pins, the system module has no I/O signals associated with **epcs_controller**. Therefore, you do not need to make any Quartus II project connections or assignments for the EPCS controller core.

This chapter does not cover the details of configuration using the EPCS device. For further information, see Altera's *Configuration Handbook*.

SDRAM

Altera provides a free SDRAM controller core, which lets you use inexpensive SDRAM as bulk RAM in your FPGA designs. The SDRAM controller core is necessary, because Avalon signals cannot describe the complex interface on an SDRAM device. The SDRAM controller acts as a bridge between the Avalon switch fabric and the pins on an SDRAM device. The SDRAM controller can operate in excess of 100 MHz.

For further details on the features and usage of the SDRAM controller core, see the *SDRAM Controller Core with Avalon Interface* chapter in volume 5 of the *Quartus II Handbook*.

Component-Level Design for SDRAM

The choice of SDRAM device(s) and the configuration of the device(s) on the board heavily influence the component-level design for the SDRAM controller. Typically, the component-level design task involves parameterizing the SDRAM controller core to match the SDRAM device(s) on the board. You must specify the structure (address width, data width, number of devices, number of banks, etc.) and the timing specifications of the device(s) on the board.

For complete details on configuration options for the SDRAM controller core, see the *SDRAM Controller Core with Avalon Interface* chapter in volume 5 of the *Quartus II Handbook*.

SOPC Builder System-Level Design for SDRAM

In SOPC Builder on the System Contents tab, the SDRAM controller looks like any other memory component. Similar to on-chip memory, there are not many SOPC Builder system-level design considerations for SDRAM. See "SOPC Builder System-Level Design" on page 9–7.

Simulation for SDRAM

At system generation time, SOPC Builder can generate a generic SDRAM simulation model and include the model in the system testbench. To use the generic SDRAM simulation model, you must turn on a setting in the

SDRAM controller configuration wizard. You can provide memory initialization contents for simulation in the file <*Quartus II project directory*>/<*SOPC Builder system name*>**_sim/**<*Memory component name*>**.dat.**

Alternately, you can provide a specific vendor memory model for the SDRAM. In this case, you must manually wire up the vendor memory model in the system testbench.

For further details, see "Simulation Considerations" on page 9–7 and "Hardware Simulation Considerations" in the chapter *SDRAM Controller Core with Avalon Interface* in volume 5 of the *Quartus II Handbook*.

Quartus II Project-Level Design for SDRAM

SOPC Builder generates a system module with top-level I/O signals associated with the SDRAM controller. In the Quartus II project, you must connect these I/O signals to FPGA pins, which connect to the SDRAM device on the board. In addition, you might have to accommodate clock skew issues.

Connecting & Assigning the SDRAM-Related Pins

After generating the system with SOPC Builder, you can find the names and directions of the I/O signals in the top-level HDL file for the SOPC Builder system module. The file has the name <*Quartus II project directory*>/<*SOPC Builder system name*>**.v** or <*Quartus II project directory*>/<*SOPC Builder system name*>**.vhd**. You must connect these signals in the top-level Quartus II design file.

You must assign a pin location for each I/O signal in the top-level Quartus II design to match the target board. Depending on the performance requirements for the design, you might have to assign FPGA pins carefully to achieve performance.

Accommodating Clock Skew

As SDRAM frequency increases, so does the possibility that you must accommodate skew between the SDRAM clock and I/O signals. This issue affects all synchronous memory devices, including SDRAM. To accommodate clock skew, you can instantiate an `altpll` megafunction in the top-level Quartus II design to create a phase-locked loop (PLL) clock output. You use a phase-shifted PLL output to drive the SDRAM clock and overcome clock-skew issues. The exact settings for the `altpll` depend on your target hardware; you must experiment to tune the phase shift to match the board.

For details, see the *altpll Megafunction User Guide*.

Board-Level Design for SDRAM

Memory requirements largely dictate the board-level configuration of the SDRAM device(s). The SDRAM controller core can accommodate various configurations of SDRAM on the board, including multiple banks and multiple devices.

For further details, see the "Example Configurations" section in the *SDRAM Controller Core with Avalon Interface* chapter in volume 5 of the *Quartus II Handbook*.

Example Design with SDRAM

This section demonstrates adding a 16 Mbyte SDRAM device to the example design. This SDRAM is a single device with 32-bit data. Figure 9–11 shows the **SDRAM Controller** configuration wizard settings for the example design.

Figure 9–11. SDRAM Controller Configuration Wizard

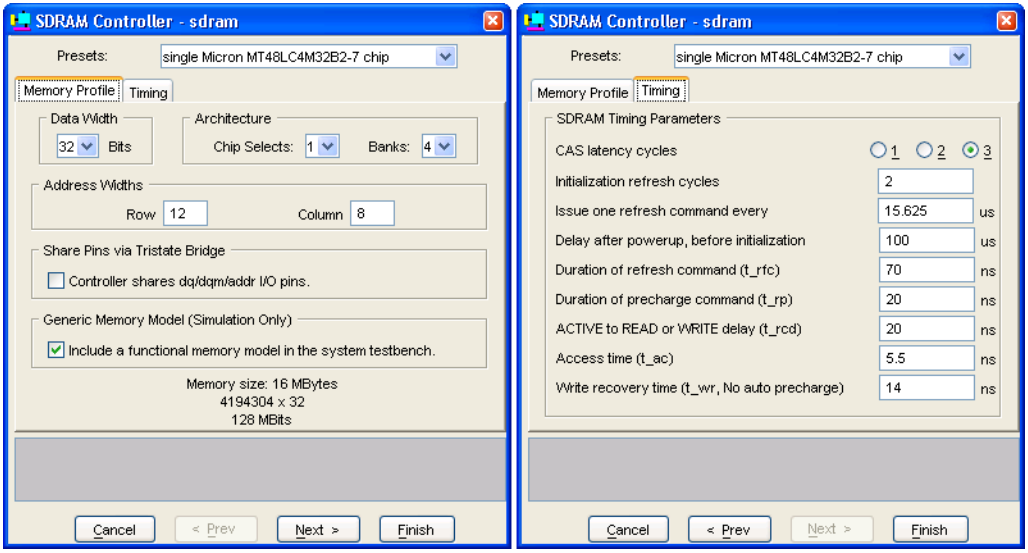
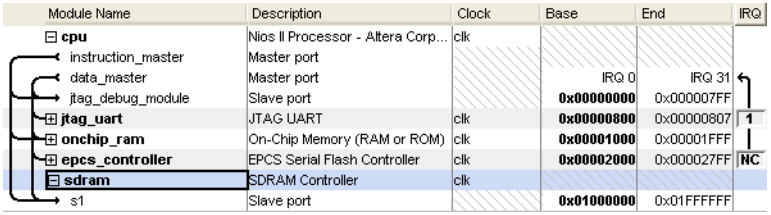


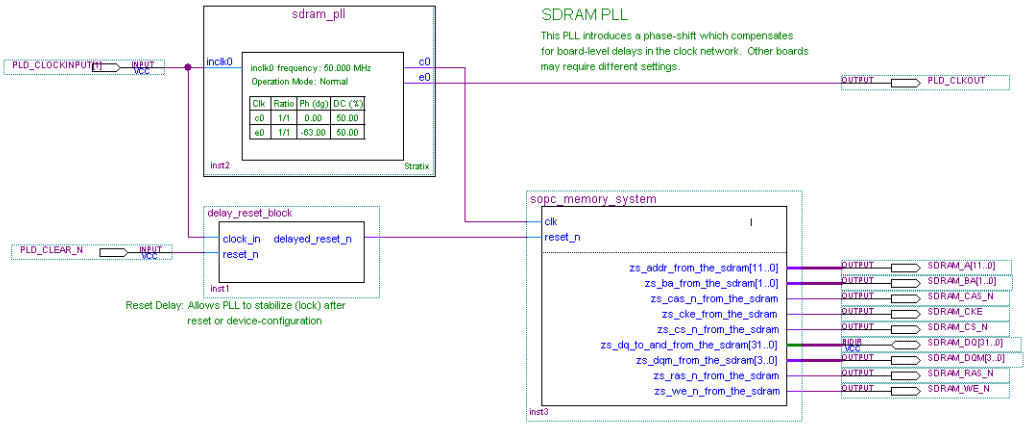
Figure 9–12 shows the SOPC Builder system after adding an instance of the SDRAM controller, renaming it to **sdram**, and assigning it a base address.

Figure 9–12. SOPC Builder System with SDRAM



For demonstration purposes, Figure 9–13 shows the result of generating the system module at this stage, and connecting it in **toplevel_design.bdf**.

Figure 9–13. toplevel_design.bdf with SDRAM



After generating the system, the top-level system module file **sopc_memory_system.v** contains the list of SDRAM-related I/O signals which must be connected to FPGA pins:

```
output   [ 11: 0] zs_addr_from_the_sdram;
output   [  1: 0] zs_ba_from_the_sdram;
output            zs_cas_n_from_the_sdram;
output            zs_cke_from_the_sdram;
output            zs_cs_n_from_the_sdram;
inout    [ 31: 0] zs_dq_to_and_from_the_sdram;
output   [  3: 0] zs_dqm_from_the_sdram;
output            zs_ras_n_from_the_sdram;
output            zs_we_n_from_the_sdram;
```

As shown in Figure 9–13, **toplevel_design.bdf** uses an instance of `sdram_pll` to phase shift the SDRAM clock by -63 degrees. **toplevel_design.bdf** also uses a subdesign `delay_reset_block` to insert a delay on the `reset_n` signal for the system module. This delay is necessary to allow the PLL output to stabilize before the SOPC Builder system begins operating.

Figure 9–14 shows pin assignments in the Quartus II assignment editor for some of the SDRAM pins. The correct pin assignments depend on the target board.

Figure 9–14. Pin Assignments for SDRAM

	To	Location	I/O Bank	I/O Standard	General Function	Special Function	Reserved
188	SDRAM_A[0]	PIN_AE4	7	LVTTL	Column I/O		
189	SDRAM_A[10]	PIN_Y11	7	LVTTL	Column I/O		
190	SDRAM_A[11]	PIN_AB7	7	LVTTL	Column I/O		
191	SDRAM_A[1]	PIN_W12	7	LVTTL	Column I/O	PGM0	
192	SDRAM_A[2]	PIN_AC11	7	LVTTL	Column I/O	nRS	
193	SDRAM_A[3]	PIN_W10	7	LVTTL	Column I/O	RUnLU	
194	SDRAM_A[4]	PIN_AA11	7	LVTTL	Column I/O	PGM1	
195	SDRAM_A[5]	PIN_AC10	7	LVTTL	Column I/O	RDN7	
196	SDRAM_A[6]	PIN_AB11	7	LVTTL	Column I/O	RUP7	
197	SDRAM_A[7]	PIN_AC8	7	LVTTL	Column I/O	FCLK5	
198	SDRAM_A[8]	PIN_AB10	7	LVTTL	Column I/O	FCLK4	
199	SDRAM_A[9]	PIN_V11	7	LVTTL	Column I/O		
200	SDRAM_BA[0]	PIN_AG19	8	LVTTL	Column I/O	DQ6B4	
201	SDRAM_BA[1]	PIN_AF19	8	LVTTL	Column I/O	DQ6B5	
202	SDRAM_CAS_N	PIN_AD18	8	LVTTL	Column I/O	DQ6B2	
203	SDRAM_CKE	PIN_AE18	8	LVTTL	Column I/O	DQ6B1	
204	SDRAM_CS_N	PIN_AG18	8	LVTTL	Column I/O	DQ6B0	
205	SDRAM_DQM[0]	PIN_AE14	7	LVTTL	Column I/O	CLK6n	
206	SDRAM_DQM[1]	PIN_Y13	7	LVTTL	Column I/O	CLK7n	
207	SDRAM_DQM[2]	PIN_AE7	7	LVTTL	Column I/O	DQS1B	
208	SDRAM_DQM[3]	PIN_AG10	7	LVTTL	Column I/O	DQS3B	

Off-Chip SRAM & Flash Memory

SOPC Builder systems can directly access many off-chip RAM and ROM devices, without a controller core to drive the off-chip memory. Avalon signals can exactly describe the interfaces on many standard memories, such as SRAM and flash memory. In this case, I/O signals on the SOPC Builder system module can connect directly to the memory device.

While off-chip memory usually has slower access time than on-chip memory, off-chip memory provides the following benefits:

- Off-chip memory is less expensive than on-chip memory resources.
- The size of off-chip memory is bounded only by the 32-bit Avalon address space.
- Off-chip ROM, such as flash memory, can be used for bulk storage of nonvolatile data.
- Multiple off-chip RAM and ROM memories can share address and data pins to conserve FPGA I/O resources.

Adding off-chip memories to an SOPC Builder system also requires the **Avalon Tristate Bridge** component.

This section describes the process of adding off-chip flash memory and SRAM to an SOPC Builder system.

Component-Level Design for SRAM & Flash Memory

There are several ways to instantiate an interface to an off-chip memory device:

- For common flash interface (CFI) flash memory devices, add the **Flash Memory (Common Flash Interface)** component in SOPC Builder.
- For Altera development boards, Altera provides SOPC Builder components that interface to the specific devices on each development board. For example, the Nios II EDS includes the components **Cypress CY7C1380C SSRAM** and **IDT71V416 SRAM**, which appear on Nios development boards.

These components make it easy for you to create memory systems targeting Altera development boards. However, these components target only the specific memory device on the board; they do not work for different devices.

- For general memory devices, RAM or ROM, you can create a custom interface to the device with the SOPC Builder component editor. Using the component editor, you define the I/O pins on the memory device and the timing requirements of the pins.

In all cases, you must also instantiate the **AvalonTristateBridge** component as well. Multiple off-chip memories can connect to a single tristate bridge.

Avalon Tristate Bridge

A tristate bridge connects off-chip devices to on-chip Avalon switch fabric. The tristate bridge creates I/O signals on the SOPC Builder system module, which you must connect to FPGA pins in the top-level Quartus II project. These pins represent the Avalon switch fabric to off-chip devices.

The tristate bridge creates address and data pins which can be shared by multiple off-chip devices. This feature lets you conserve FPGA pins when connecting the FPGA to multiple devices with mutually exclusive access.

You must use a tristate bridge in either of the following cases:

- The off-chip device has bidirectional data pins.
- Multiple off-chip devices share the address and/or data buses.

In SOPC Builder, you instantiate a tristate bridge by instantiating the **AvalonTristateBridge** component. The Avalon Tristate Bridge configuration wizard has a single option: To register incoming (to the FPGA) signals or not.

- **Registered** – This setting adds registers to all FPGA input pins associated with the tristate bridge (outputs from the memory device).
- **Not Registered** – This setting does not add registers between the memory device output pins and the Avalon switch fabric.

The Avalon tristate bridge automatically adds registers to output signals from the tristate bridge to off-chip devices.

Registering the input and output signals shortens the register-to-register delay from the memory device to the FPGA, resulting in higher system f_{MAX} performance. However, in each direction, the registers add one additional cycle of latency for Avalon master ports accessing memory connected to the tristate bridge. The registers do not affect the timing of the transfers from the perspective of the memory device.

For details on the Avalon tristate interface, refer to the *Avalon Interface Specification*.

Flash Memory

In SOPC Builder, you instantiate an interface to CFI flash memory by adding a **Flash Memory (Common Flash Interface)** component. If the flash memory is not CFI compliant, you must create a custom interface to the device with the SOPC Builder component editor.

The choice of flash device(s) and the configuration of the device(s) on the board heavily influence the component-level design for the flash memory configuration wizard. Typically, the component-level design task involves parameterizing the flash memory interface to match the device(s) on the board. Using the Flash Memory (Common Flash Interface) configuration wizard, you must specify the structure (address width and data width) and the timing specifications of the device(s) on the board.

For details on features and usage, refer to chapter *Common Flash Interface Controller Core with Avalon Interface* in volume 5 of the *Quartus II Handbook*.

For an example of instantiating the Flash Memory (Common Flash Interface) component in an SOPC Builder system, see "Example Design with SRAM & Flash Memory" on page 9–26.

SRAM

To instantiate an interface to off-chip RAM, you perform the following steps:

1. Create a new component with the SOPC Builder component editor that defines the interface.
2. Instantiate the new interface component in the SOPC Builder system.

The choice of RAM device(s) and the configuration of the device(s) on the board determine how you create the interface component. The component-level design task involves entering parameters into the component editor to match the device(s) on the board.

For details on using the component editor, refer to the *Component Editor* chapter in volume 4 of the *Quartus II Handbook*.

SOPC Builder System-Level Design for SRAM & Flash Memory

In SOPC Builder on the System Contents tab, the Avalon tristate bridge has two ports:

- Avalon slave port – This port faces the on-chip logic in the SOPC Builder system. You connect this slave port to on-chip master ports in the system.
- Avalon tristate master port – This port faces the off-chip memory devices. You connect this master port to the Avalon tristate slave ports on the interface components for off-chip memories.

You assign a clock to the Avalon tristate bridge which determines the Avalon clock cycle time for off-chip devices connected to the tristate bridge.

You must assign base addresses to each off-chip memory. The Avalon tristate bridge does not have an address; it passes unmodified addresses from on-chip master ports to off-chip slave ports.

Simulation for SRAM & Flash Memory

The SOPC Builder output for simulation depends on the type of memory component(s) in the system:

- **Flash Memory (Common Flash Interface)** component – This component provides a generic simulation model. You can provide memory initialization contents for simulation in the file <*Quartus II project directory*>/<*SOPC Builder system name*>**_sim/**<*Flash memory component name*>**.dat**.
- Custom memory interface created with the component editor – In this case, you must manually connect the vendor simulation model to the system testbench. SOPC Builder does not automatically connect simulation models for custom memory components to the system module.
- Altera-provided interfaces to memory devices – Altera provides simulation models for these interface components. You can provide memory initialization contents for simulation in the file <*Quartus II project directory*>/<*SOPC Builder system name*>**_sim/**<*Memory component name*>**.dat**. Alternately, you can provide a specific vendor simulation model for the memory. In this case, you must manually wire up the vendor memory model in the system testbench.

For further details, see "Simulation Considerations" on page 9–7.

Quartus II Project-Level Design for SRAM & Flash Memory

SOPC Builder generates a system module with top-level I/O signals associated with the tristate bridge and the memory interface components. In the Quartus II project, you must connect the I/O signals to FPGA pins, which connect to the memory device(s) on the board.

After generating the system with SOPC Builder, you can find the names and directions of the I/O signals in the top-level HDL file for the SOPC Builder system module. The file has the name <*Quartus II project directory*>/<*SOPC Builder system name*>**.v** or <*Quartus II project directory*>/<*SOPC Builder system name*>**.vhd**. You must connect these signals in the top-level Quartus II design file.

You must assign a pin location for each I/O signal in the top-level Quartus II design to match the target board. Depending on the performance requirements for the design, you might have to assign FPGA pins carefully to achieve performance.

SOPC Builder inserts synthesis directives in the top-level system module HDL to assist the Quartus II fitter with signals that interface with off-chip devices. An example is below:

```
reg [ 22: 0] tri_state_bridge_address /* synthesis
ALTERA_ATTRIBUTE =  "FAST_OUTPUT_REGISTER=ON" */;
```

Board-Level Design for SRAM & Flash Memory

Memory requirements largely dictate the board-level configuration of the SRAM & flash memory device(s). You can lay out memory devices in any configuration, as long as the resulting interface can be described with Avalon signals.

Special consideration is required when connecting the Avalon `address` signal to the address pins on the memory devices.

The system module presents the smallest number of address lines required to access the largest off-chip memory, which is usually less than 32 address bits. Not all memory devices connect to all address lines.

Aligning the Least-Significant Address Bits

The Avalon tristate `address` signal always presents a byte address. Each address location in many memory devices contains more than one byte of data. In this case, the memory device must ignore one or more of the least-significant Avalon `address` lines. For example, a 16-bit memory device must ignore Avalon `address[0]` (which is a byte address), and connect Avalon `address[1]` to the least-significant address line.

Table 9–1 shows the relationship between Avalon address lines and off-chip address pins for all possible Avalon data widths.

Table 9–1. Connecting the Least-Significant Avalon Address Line

Avalon address Line	Address Line on Memory Device				
	8-bit Memory	16-bit Memory	32-bit Memory	64-bit Memory	128-bit Memory
address[0]	A0	No connect	No connect	No connect	No connect
address[1]	A1	A0	No connect	No connect	No connect
address[2]	A2	A1	A0	No connect	No connect
address[3]	A3	A2	A1	A0	No connect
address[4]	A4	A3	A2	A1	A0
address[5]	A5	A4	A3	A2	A1
address[6]	A6	A5	A4	A3	A2
address[7]	A7	A6	A5	A4	A3
address[8]	A8	A7	A6	A5	A4
address[9]	A9	A8	A7	A6	A5
address[10]	A10	A9	A8	A7	A6
...	...	...	...	...	...

Aligning the Most-Significant Address Bits

The Avalon address signal contains enough address lines for the largest memory on the tristate bridge. Smaller off-chip memories might not use all of the most-significant address lines.

For example, a memory device with 2^{10} locations uses 10 address bits, while a memory with 2^{20} locations uses 20 address bits. If both these devices share the same tristate bridge, then the smaller memory ignores the ten most-significant Avalon address lines.

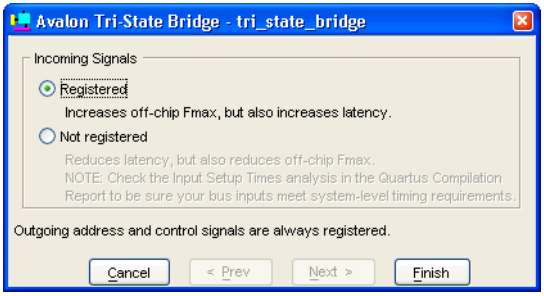
Example Design with SRAM & Flash Memory

This section demonstrates adding a 1 Mbyte SRAM and an 8Mbyte flash memory to the example design. These memory devices connect to the Avalon switch fabric through an Avalon tristate bridge.

Adding the Avalon Tristate Bridge

Figure 9–15 shows the **Avalon Tristate Bridge** configuration wizard for the example design. The example design uses registered inputs and outputs to maximize system f_{MAX}, which increases the read latency by two for both the SRAM and flash memory.

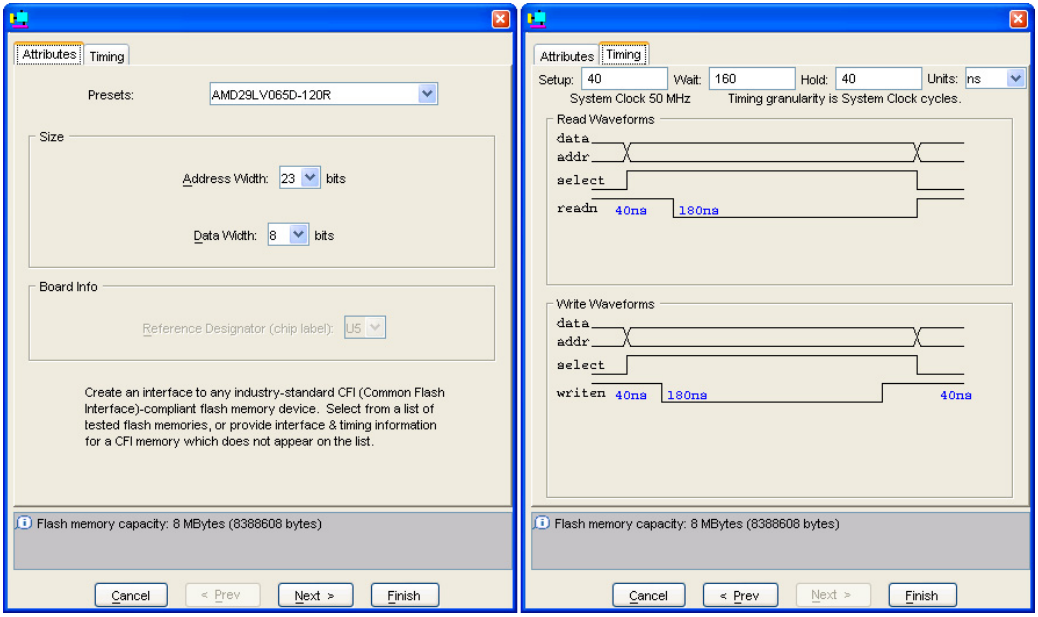
Figure 9–15. Avalon Tristate Bridge Configuration Wizard



Adding the Flash Memory Interface

The flash memory is 8M x 8-bit, which requires 23 address bits and 8 data bits. Figure 9–16 shows the **Flash Memory (Common Flash Interface)** configuration wizard settings for the example design. In this example, the target board declares a reference designator U5 for the flash device on the board.

Figure 9–16. Flash Memory Configuration Wizard



Adding the SRAM Interface

The SRAM device is 256K x 32-bit, which requires 18 address bits and 32 data bits. The example design uses a custom memory interface created with the SOPC Builder component editor. Figure 9–17 – Figure 9–21 shows the settings required on the various component editor tabs to implement an interface to this SRAM.

Figure 9–17. SRAM Interface Component Editor HDL Files Tab

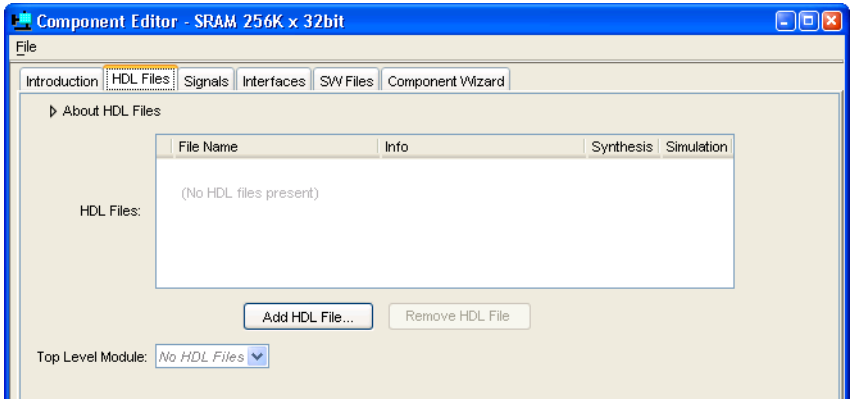


Figure 9–18. SRAM Interface Component Editor Signals Tab

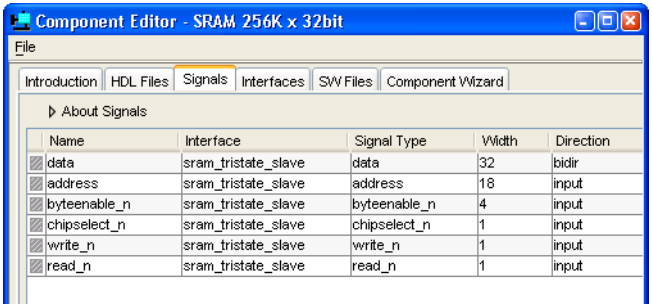


Figure 9–19. SRAM Interface Component Editor Interfaces Tab

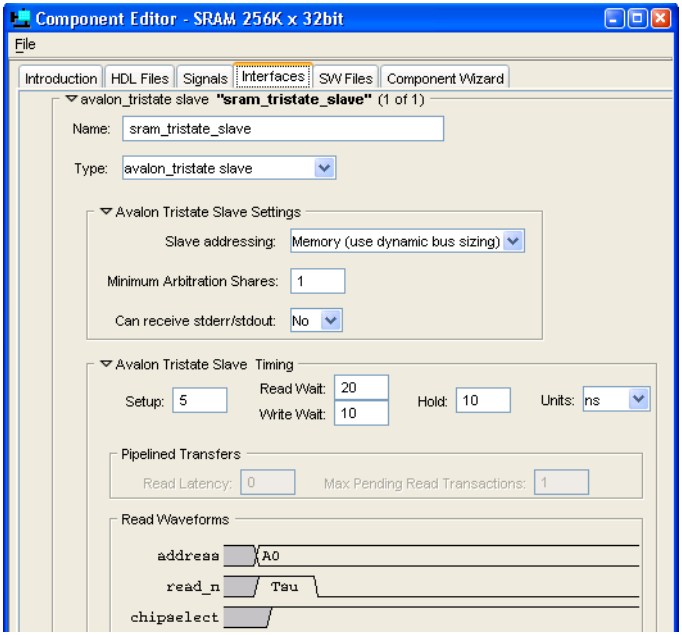
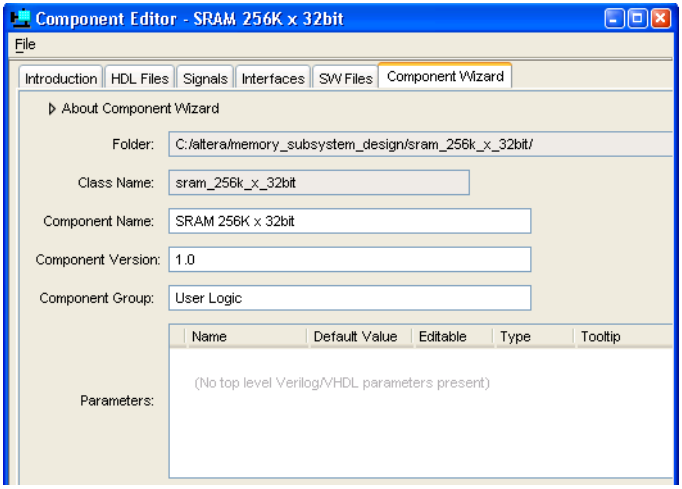


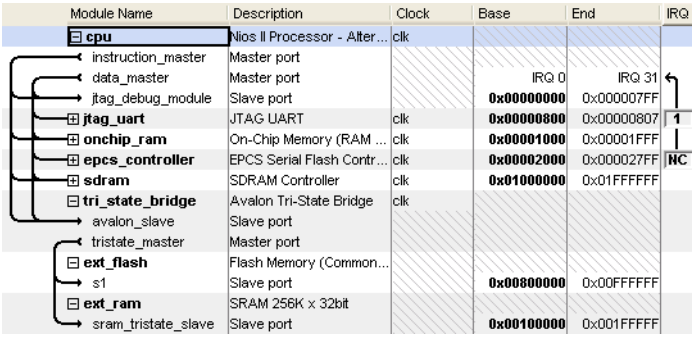
Figure 9–20. SRAM Interface Component Editor Component Wizard Tab



SOPC Builder System Contents Tab

Figure 9–21 shows the SOPC Builder system after adding the tristate bridge and memory interface components, and configuring them appropriately on the **System Contents** tab. Figure 9–21 represents the complete example design in SOPC Builder.

Figure 9–21. SOPC Builder System with SRAM & Flash Memory



After generating the system, the top-level system module file **sopc_memory_system.v** contains the list of I/O signals for SRAM and flash memory which must be connected to FPGA pins:

```
output            chipselect_n_to_the_ext_ram;
output            read_n_to_the_ext_ram;
output            select_n_to_the_ext_flash;
output   [ 22: 0] tri_state_bridge_address;
output   [  3: 0] tri_state_bridge_byteenablen;
inout    [ 31: 0] tri_state_bridge_data;
output            tri_state_bridge_readn;
output            write_n_to_the_ext_flash;
output            write_n_to_the_ext_ram;
```

The Avalon tristate bridge signals which can be shared are named after the instance of the tristate bridge component, such as `tri_state_bridge_data[31:0]`.

Connecting & Assigning Pins in the Quartus II Project

Figure 9–22 shows the result of generating the system module for the complete example design, and connecting it in **toplevel_design.bdf**.

Figure 9–22. toplevel_design.bdf with SRAM & Flash Memory

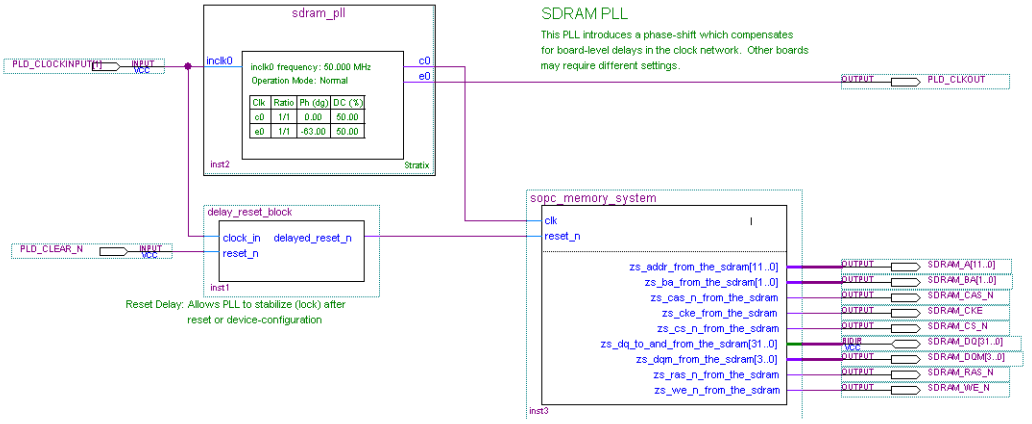


Figure 9–23 shows the pin assignments in the Quartus II assignment editor for some of the SRAM and flash memory pins. The correct pin assignments depend on the target board.

Figure 9–23. Pin Assignments for SRAM & Flash Memory

	To	Location	I/O Bank	I/O Standard	General Function	Special Function	Reser
243	SRAM_BE_N[0]	PIN_M18	3	LVTTL	Column I/O		
244	SRAM_BE_N[1]	PIN_F17	3	LVTTL	Column I/O		
245	SRAM_BE_N[2]	PIN_J18	3	LVTTL	Column I/O	RUP3	
246	SRAM_BE_N[3]	PIN_L17	3	LVTTL	Column I/O	CLK15n	
247	SRAM_CS_N	PIN_B24	3	LVTTL	Column I/O	DQ9T4	
248	SRAM_OE_N	PIN_B26	3	LVTTL	Column I/O	DQ9T7	
249	SRAM_WE_N	PIN_C24	3	LVTTL	Column I/O	DQ59T	

Connecting FPGA Pins to Devices on the Board

Table 9–2 shows the mapping between the Avalon address lines and the address pins on the SRAM and flash memory devices.

Table 9–2. FPGA Connections to SRAM & Flash Memory (Part 1 of 2)

Avalon Address Line	Flash Address (8M x 8-bit Data)	SRAM Address (256K x 32-bit data)
tri_state_bridge_address[0]	A0	No connect
tri_state_bridge_address[1]	A1	No connect
tri_state_bridge_address[2]	A2	A0

Table 9–2. FPGA Connections to SRAM & Flash Memory (Part 2 of 2)

Avalon Address Line	Flash Address (8M x 8-bit Data)	SRAM Address (256K x 32-bit data)
tri_state_bridge_address[3]	A3	A1
tri_state_bridge_address[4]	A4	A2
tri_state_bridge_address[5]	A5	A3
tri_state_bridge_address[6]	A6	A4
tri_state_bridge_address[7]	A7	A5
tri_state_bridge_address[8]	A8	A6
tri_state_bridge_address[9]	A9	A7
tri_state_bridge_address[10]	A10	A8
tri_state_bridge_address[11]	A11	A9
tri_state_bridge_address[12]	A12	A10
tri_state_bridge_address[13]	A13	A11
tri_state_bridge_address[14]	A14	A12
tri_state_bridge_address[15]	A15	A13
tri_state_bridge_address[16]	A16	A14
tri_state_bridge_address[17]	A17	A15
tri_state_bridge_address[18]	A18	A16
tri_state_bridge_address[19]	A19	A17
tri_state_bridge_address[20]	A20	No connect
tri_state_bridge_address[21]	A21	No connect
tri_state_bridge_address[22]	A22	No connect