Avalon Blocks in DSP Builder

October 2005, ver. 1.0 Application Note 403

Introduction

SOPC Builder is a system development tool for creating systems based on processors, peripherals, and memories. SOPC Builder automates the task of integrating hardware components into a larger system. In addition, SOPC Builder supports peripherals that use the Avalon® switch fabric. The Avalon Interface specification provides peripheral designers with a basis for describing the address-based read/write interface found on master (for example, microprocessors and DMA controllers) and slave peripherals (for example, memory, UARTs, and timers).

This application note describes what you need to do to create a DSP Builder design functioning as a custom peripheral to SOPC Builder. To integrate a DSP Builder design into your SOPC Builder system, your peripheral must meet the Avalon Interface specification and qualify as an SOPC Builder-ready component.

The Avalon Master/Slave blocks in DSP Builder provide a seamless flow for creating a DSP Builder block as a custom peripheral and integrating the block into your SOPC Builder system. The Avalon Master/Slave blocks provide you with the following benefits:

- Automates the process of specifying Avalon ports that are compatible with the Avalon bus
- Supports multiple Avalon Master and Slave instantiations
- Generates a component descriptor file (**class.ptf**), which SOPC Builder can recognize as a component with master or slave ports

This document gives you an overview of the Avalon Master/Slave blocks provided in DSP Builder. It then steps you through the procedure for developing custom peripherals using Avalon blocks and importing them to be SOPC Builder-ready components.

For more information on SOPC Builder, refer to the *Quartus II Handbook Volume 4: SOPC Builder.*

For more information on the Avalon interface, refer to the *Avalon Interface Specification*.

Avalon Blocks

An SOPC Builder component is a design module that SOPC Builder recognizes and can automatically integrate into a system. For an SOPC Builder hardware component, the peripheral has two fundamental pieces:

- Hardware design files that describe the component hardware.
- A component descriptor file called **class.ptf** for use by SOPC Builder. This file defines the structure of the component and provides SOPC Builder with the information it needs to integrate the component into a system.

With the Avalon blocks provided in the DSP Builder library, you can design the DSP function and add an Avalon block to turn it into a custom peripheral within the Simulink environment. Each Avalon block can be instantiated multiple times in a design to implement an SOPC component with multiple master and/or slave ports. You can then invoke the SignalCompiler in DSP Builder to convert the Simulink model into hardware design files and generate the **class.ptf** file.

The Avalon Master/Slave block provided in DSP Builder is configurable with the following types of signals:

- Fundamental signals
- Wait-State signals
- Pipeline signals
- Burst signals
- Flow Control signals
- Other signals

The specific signals available differ between the master and slave blocks. The following sections describe the master and slave blocks in more detail.

Avalon Slave Block

Table 1 lists the signals that the Avalon Slave block supports.

Table 1. Avalon Slave Block Signals (Part 1 of 2)

Signal Type	Signals
Fundamental	`clk, address, read, readdata, write, writedata, byteenable`
Wait-State	`waitrequest`
Pipeline	`readdatavalid`
Burst	`burstcount, beginbursttransfer`

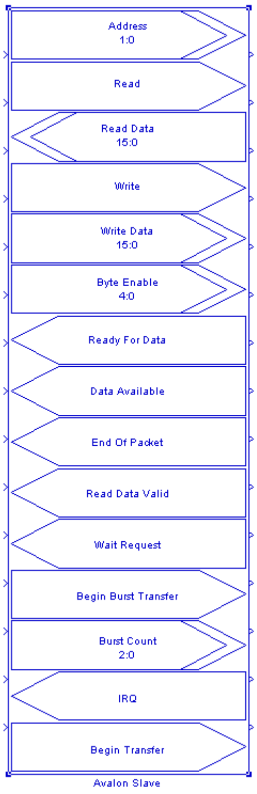
Table 1. Avalon Slave Block Signals (Part 2 of 2)

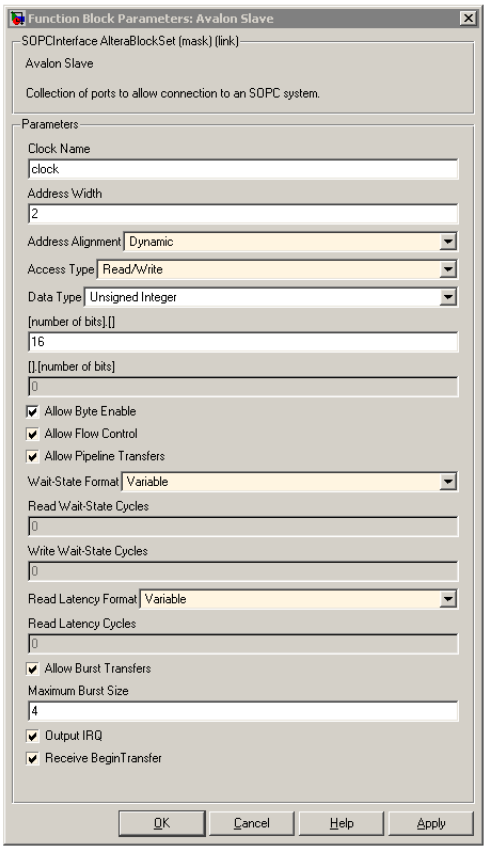
Signal Type	Signals
Flow Control	`readyfordata, dataavailable, endofpacket`
Other	`irq`

The block shown in Figure 1 describes an Avalon Slave interface in which all of the Avalon signals have been enabled. Each of the input and output ports of the block corresponds to the input and output ports of the pin or bus shown between the ports. Inputs to the DSP Builder core are displayed as right-pointing pins/buses, while outputs from the core are displayed as left-pointing pins/buses. The opposite end of any pin can be used to provide "pass-through" test data from the Simulink domain.

Figure 1 shows the Avalon Slave interface and the Block Parameters dialog box in Simulink.

Figure 1. Avalon Slave





SOPCInterface AlteraBlockSet (mask) (link)

Avalon Slave

Collection of ports to allow connection to an SOPC system.

Parameters

Clock Name: clock

Address Width: 2

Address Alignment: Dynamic

Access Type: Read/Write

Data Type: Unsigned Integer

[number of bits].[]: 16

[].[number of bits]: 0

[✓] Allow Byte Enable

[✓] Allow Flow Control

[✓] Allow Pipeline Transfers

Wait-State Format: Variable

Read Wait-State Cycles: 0

Write Wait-State Cycles: 0

Read Latency Format: Variable

Read Latency Cycles: 0

[✓] Allow Burst Transfers

Maximum Burst Size: 4

[✓] Output IRQ

[✓] Receive BeginTransfer

Avalon Master Block

In some applications, you may want to incorporate a DMA controller in your SOPC Builder system and have it function as an Avalon Master in the system. The Avalon Master block is similar to the Avalon Slave block in DSP Builder.

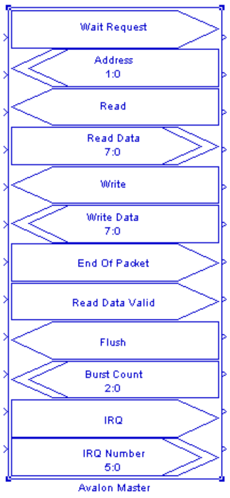
Table 2 lists the signals that the Avalon Master block supports.

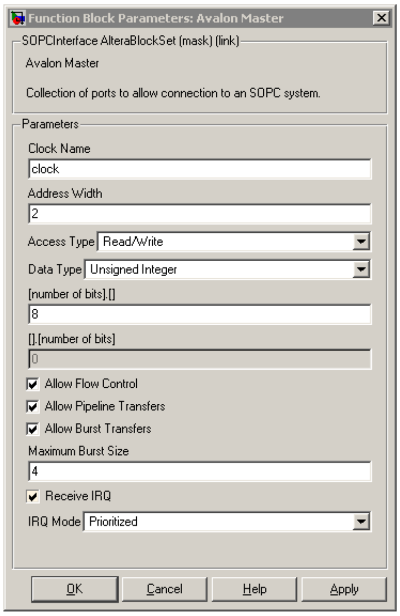
Table 2. Avalon Master Block Signals

Signal Type	Signals
Fundamental	`clk, waitrequest, address, read, readdata, write, writedata, byteenable`
Pipeline	`readdatavalid, flush`
Burst	`burstcount`
Flow Control	`endofpacket`
Other	`irq, irqnumber`

Figure 2 shows the Avalon Master interface and the Block Parameters dialog box in Simulink.

Figure 2. Avalon Master




Function Block Parameters: Avalon Master

SOPCInterface AlteraBlockSet (mask) (link)

Avalon Master

Collection of ports to allow connection to an SOPC system.

Parameters

Clock Name: clock

Address Width: 2

Access Type: Read/Write

Data Type: Unsigned Integer

[number of bits].[]: 8

[].[number of bits]: 0

[✓] Allow Flow Control

[✓] Allow Pipeline Transfers

[✓] Allow Burst Transfers

Maximum Burst Size: 4

[✓] Receive IRQ

IRQ Mode: Prioritized

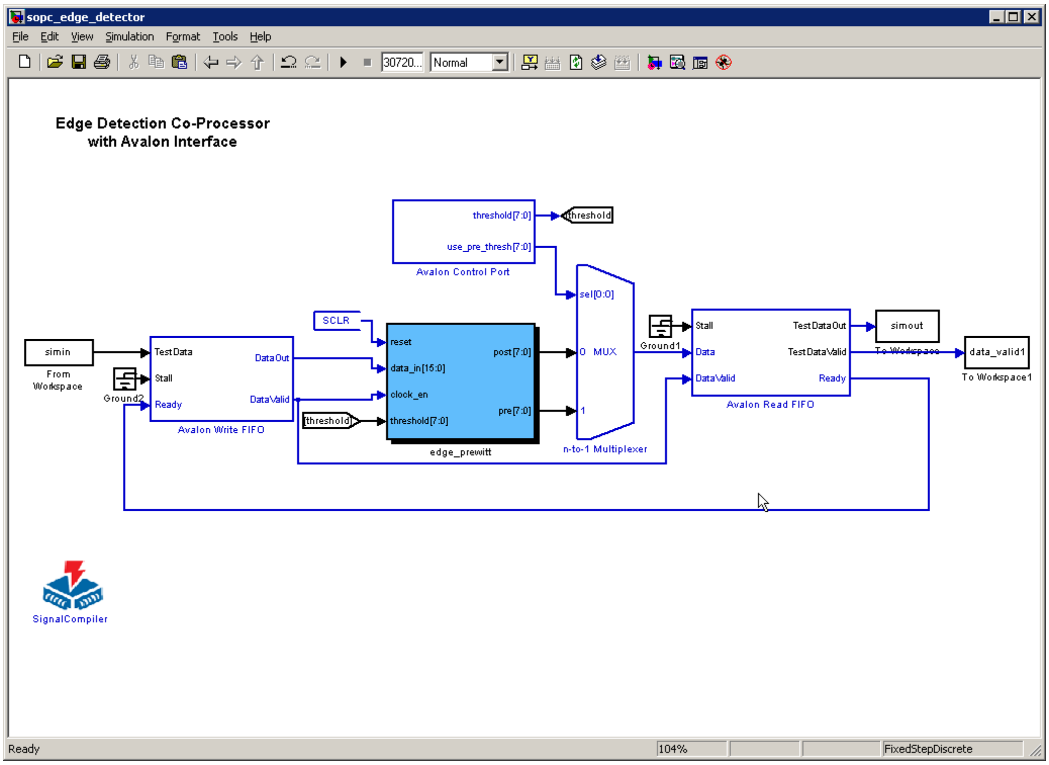
Avalon FIFO Blocks

While the Avalon Master and Slave interface blocks allow you to generate an SOPC component in DSP Builder, they do little to mask the complexities of the interface. FIFO blocks in the **SOPC Builder Links** library give you a higher level of abstraction.

A typical DSP core may be implemented to handle data in a streaming manner, using the signals `Data`, `Valid`, and `Ready` to move the data. Configurable blocks are provided to map Avalon signals to this protocol.

Figure 3 shows an example with Avalon Write FIFO and Avalon Read FIFO blocks.

Figure 3. Avalon Write FIFO and Avalon Read FIFO Example



The blocks are provided in a hierarchical form, so you can change the block functionality if required (for example, if an Avalon address input is being used to split incoming streams). Double-clicking a block brings up a Block Parameters dialog box, in which you can set parameters such as FIFO size, data width, maximum burst size, and so on.

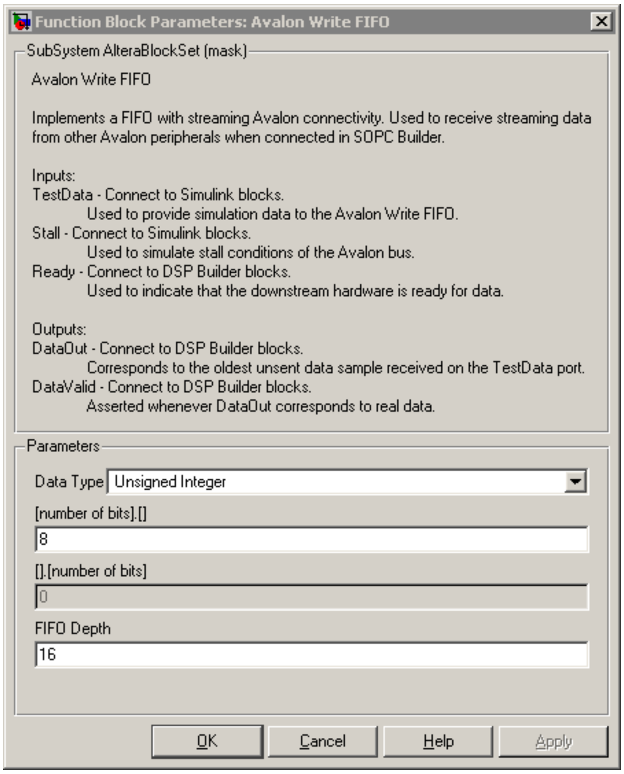
Avalon Write FIFO Blocks

Avalon Write FIFO blocks contain the following ports:

- `TestData`—This input port should be connected to Simulink blocks. It provides simulation data to the Avalon Write FIFO block. The data is passed to the `DataOut` port one cycle after the `Ready` input port is asserted.
- `Stall`—This input port should be connected to Simulink blocks. It simulates stall conditions of the Avalon bus and hence underflow to the SOPC component. For any simulation cycle where `Stall` is asserted, the data provided by `TestData` is cached by the Streaming Avalon Write Converter and released in order, one sample per clock, when `Stall` is deasserted.
- `Ready`—This input port should be connected to DSP Builder blocks. It indicates that the downstream hardware is ready for data.
- `DataOut`—This output port should be connected to DSP Builder blocks. It corresponds to the oldest unsent data sample received on the `TestData` port.
- `DataValid`—This output port should be connected to DSP Builder blocks. It is asserted whenever `DataOut` corresponds to real data.

Figure 4 shows the Avalon Write FIFO dialog box.

Figure 4. Avalon Write FIFO Interface



Function Block Parameters: Avalon Write FIFO

SubSystem AlteraBlockSet (mask)

Avalon Write FIFO

Implements a FIFO with streaming Avalon connectivity. Used to receive streaming data from other Avalon peripherals when connected in SOPC Builder.

Inputs:
TestData - Connect to Simulink blocks.
Used to provide simulation data to the Avalon Write FIFO.
Stall - Connect to Simulink blocks.
Used to simulate stall conditions of the Avalon bus.
Ready - Connect to DSP Builder blocks.
Used to indicate that the downstream hardware is ready for data.

Outputs:
DataOut - Connect to DSP Builder blocks.
Corresponds to the oldest unsent data sample received on the TestData port.
DataValid - Connect to DSP Builder blocks.
Asserted whenever DataOut corresponds to real data.

Parameters

Data Type: Unsigned Integer

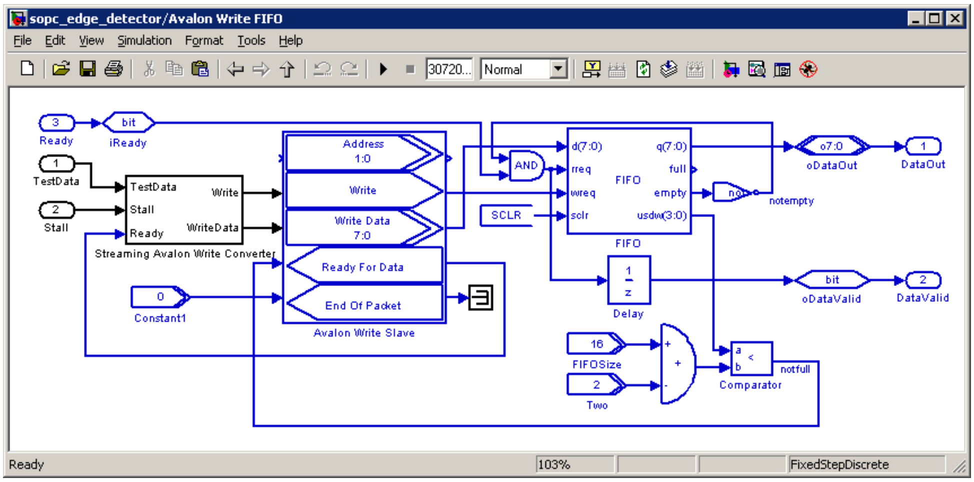
[number of bits].[]: 8

[].[number of bits]: 0

FIFO Depth: 16

Figure 5 shows the internals of the Avalon Write FIFO block. To customize the block, right-click on the block and click **Look Under Mask** to change the block's internal properties. The Streaming Avalon Write Converter handles caching and conversion of Simulink/MATLAB data into accesses over the Avalon interface and can be used to test the functionality of your design. The Streaming Avalon Write Converter is simulation only and does not synthesize to HDL.

Figure 5. Avalon Write FIFO Block Internals



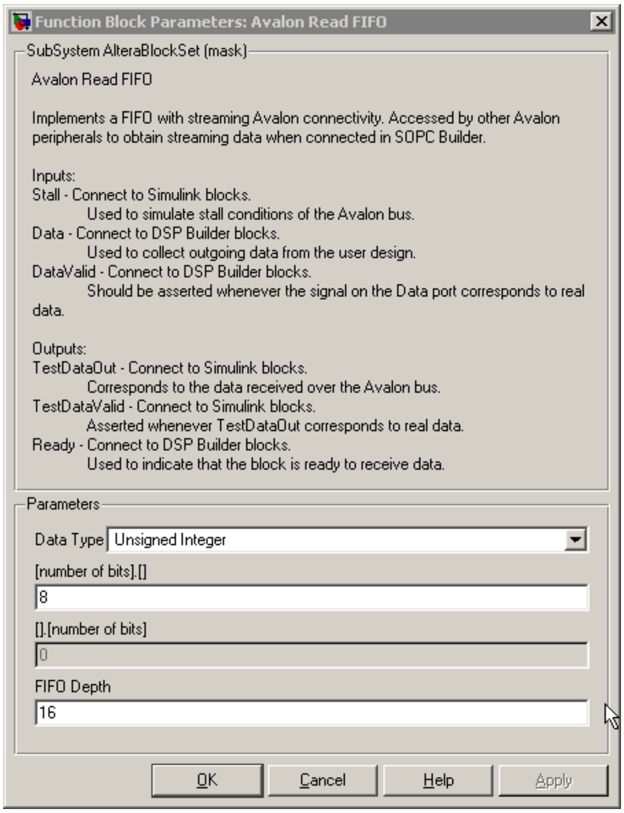
Avalon Read FIFO Blocks

Avalon Read FIFO blocks consist of the following ports:

- `Stall`—This input port should be connected to Simulink blocks. It simulates stall conditions of the Avalon bus and hence backpressure to the SOPC component. For any simulation cycle where `Stall` is asserted, no Avalon reads take place and so the internal FIFO buffer fills up. When full, the `Ready` output is deasserted, ensuring that data is not lost.
- `Data`—This input port should be connected to DSP Builder blocks. It connects to outgoing data from the user design.
- `DataValid`—This input port should be connected to DSP Builder blocks. It is asserted whenever the signal on the `Data` port corresponds to real data.
- `TestDataOut`—This output port should be connected to Simulink blocks. It corresponds to the data received over the Avalon bus.
- `TestDataValid`—This output port should be connected to Simulink blocks. It is asserted whenever `TestDataOut` corresponds to real data.
- `Ready`—This output port should be connected to DSP Builder blocks. It indicates the data is ready to read from the FIFO.

Figure 6 shows the Avalon Read FIFO dialog box.

Figure 6. Avalon Read FIFO Interface



Function Block Parameters: Avalon Read FIFO

SubSystem AlteraBlockSet (mask)

Avalon Read FIFO

Implements a FIFO with streaming Avalon connectivity. Accessed by other Avalon peripherals to obtain streaming data when connected in SOPC Builder.

Inputs:
Stall - Connect to Simulink blocks.
Used to simulate stall conditions of the Avalon bus.
Data - Connect to DSP Builder blocks.
Used to collect outgoing data from the user design.
DataValid - Connect to DSP Builder blocks.
Should be asserted whenever the signal on the Data port corresponds to real data.

Outputs:
TestDataOut - Connect to Simulink blocks.
Corresponds to the data received over the Avalon bus.
TestDataValid - Connect to Simulink blocks.
Asserted whenever TestDataOut corresponds to real data.
Ready - Connect to DSP Builder blocks.
Used to indicate that the block is ready to receive data.

Parameters

Data Type: Unsigned Integer

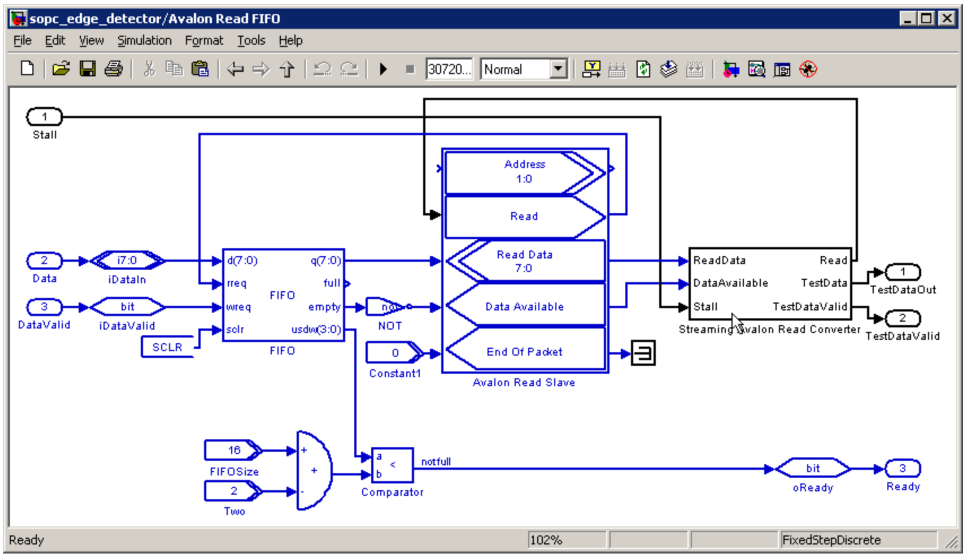
[number of bits].[]: 8

[].[number of bits]: 0

FIFO Depth: 16

The internals of the Avalon Read FIFO block are shown in Figure 7. To customize the block, right-click on the block and click **Look Under Mask** to change the block’s internal properties. The Streaming Avalon Read Converter handles caching and conversion of Simulink/MATLAB data from the Avalon interface and can be used to test the functionality of your design. The Streaming Avalon Read Converter is simulation only and does not synthesize to HDL.

Figure 7. Avalon Read FIFO Block Internals



Generating the SOPC Component in DSP Builder

The Avalon block allows you to create Avalon ports for any DSP Builder block to import to an SOPC Builder system. You must add the Avalon ports to the DSP design block to indicate the boundaries for the SOPC Builder component. After you finish your DSP design using blocks provided in the Altera® DSP Builder blockset, perform the following steps to add the Avalon ports to the design block and create the SOPC Builder-ready peripheral.

1. In MATLAB, open the **Simulink Library Browser**.
2. Open the **SOPC Builder links** library in the **Altera DSP Builder** blockset.
3. Open the **Avalon Blocks** library.
4. Drag and drop the **Avalon Master** or **Avalon Slave** block into your model, positioning it beside the design block.
5. Rename the Avalon Master or Avalon Slave block to suit your SOPC component.
6. Double-click on the Avalon Master or Avalon Slave block to set the parameters.

7. Connect the signals of the Avalon Master or Avalon Slave port to your design block. You may have to create extra logic surrounding their design to comply with the Avalon interface specification.

For more information on the Avalon interface, refer to the *Avalon Interface Specification*.

8. Drag and drop the **SignalCompiler** block into your model. It is located in the **AltLab** library in the Altera DSP Builder blockset in the Simulink Library Browser.

9. Double-click the new SignalCompiler block in your model.

10. The SignalCompiler dialog box appears. Click **Analyze**.

11. Click the **SOPC Info** tab. Turn on **Generate SOPC Builder PTF File**.

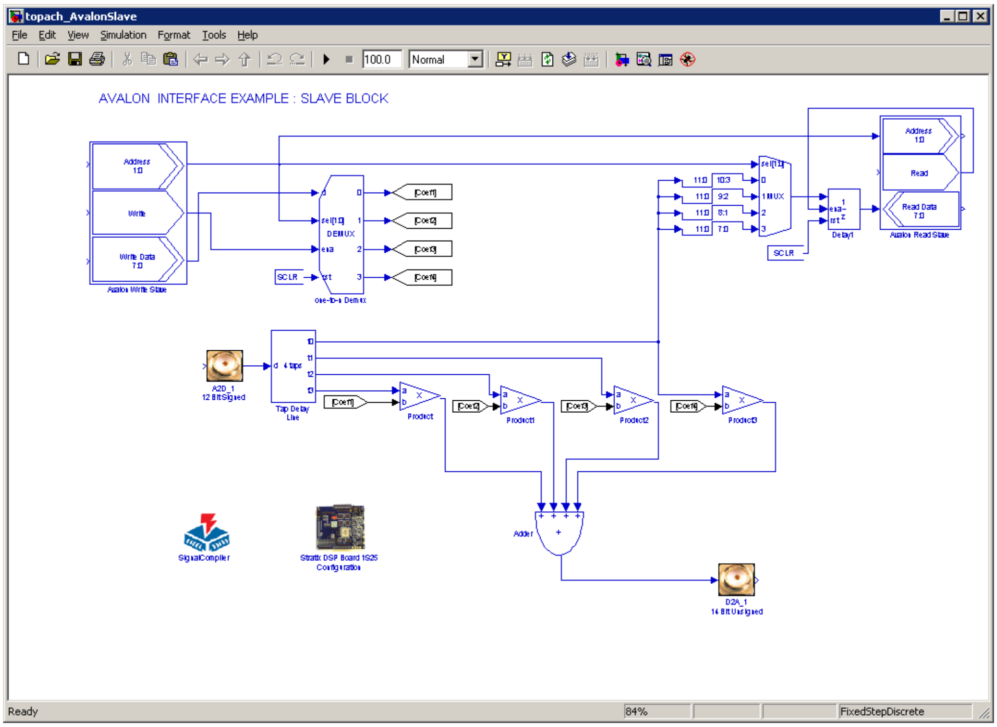
12. (Optional for simulation purpose) Click the **Testbench** tab. Turn on **Generate Stimuli for VHDL Testbench**.

13. Click **1-Convert MDL to VHDL**.

14. (Optional for simulation purpose) Run the simulation in Simulink to generate the input stimulus files by clicking **Start** from the Simulation menu. SignalCompiler generates a simulation script and a VHDL testbench that imports the Simulink input stimuli.

Figure 8 shows a DSP filter design block with one write slave and one read slave in Simulink.

Figure 8. Avalon Slave Interface Example



For a complete tutorial on the Avalon Block in DSP Builder, refer to the *Using the SOPC Builder Links Library* chapter in the *DSP Builder User Guide*.

Verifying Custom Peripherals

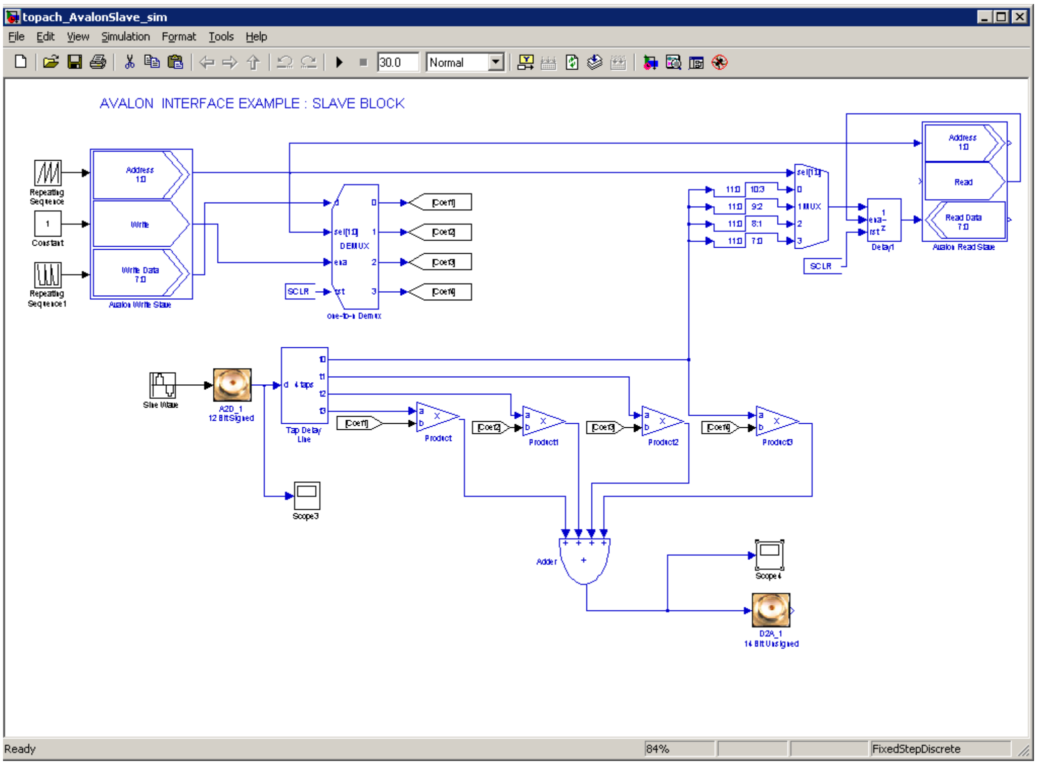
You can use either Simulink or ModelSim simulation to verify the functionality of your system.

Simulink Simulation

You can use Simulink blocksets from the library to apply a stimulus to the DSP design with Avalon ports. By observing the output on a Simulink scope block, you can verify the functionality of the DSP block and the connectivity between the DSP block and Avalon ports.

Figure 9 shows an Avalon Interface example in a DSP Builder system with Simulink simulation.

Figure 9. Avalon Interface Design Example With Simulink Simulation



ModelSim Simulation

By turning on **Generate Stimuli for VHDL Testbench** in the Testbench pane in SignalCompiler, SignalCompiler generates a VHDL testbench and Tcl script for the model. You can use both files with the ModelSim software or use the testbench in another simulation tool.

For information on how to perform a Verilog HDL simulation of your DSP Builder model, see the *SignalCompiler Block* section in the *AltLab Library* chapter in the *DSP Builder Reference Manual*.

If you are using the peripherals in a Nios II system, refer to *AN 351: Simulating Nios II Embedded Processor Designs*.

Integrating a DSP Peripheral Into an SOPC Builder System

With the **class.ptf** file generated by SignalCompiler, the DSP block now shows as a component under **DSP Builder (Matlab – Simulink) Peripheral** on the drop down list in SOPC Builder system. You may instantiate multiple DSP components in the SOPC Builder system and connect to other peripherals.

1. Start the Quartus® II software.

2. On the File menu, click **New Project Wizard** and create a new project.

3. On the Tools menu, click **SOPC Builder**.

4. The **Create New System** window appears. Type your system name and click **OK**.

5. In the SOPC Builder list of available components, select the component you created in the **DSP Builder (Matlab – Simulink) Peripheral** group. Click **Add**.

   ☞ For the peripheral to appear in SOPC Builder, the working directory for your SOPC Builder project must be the same as your DSP Builder working directory.

   ☞ If you want to keep separate directories for SOPC Builder and the DSP Builder project, copy the **class.ptf** and VHDL files generated by SignalCompiler to the directory in which the SOPC Builder project resides.

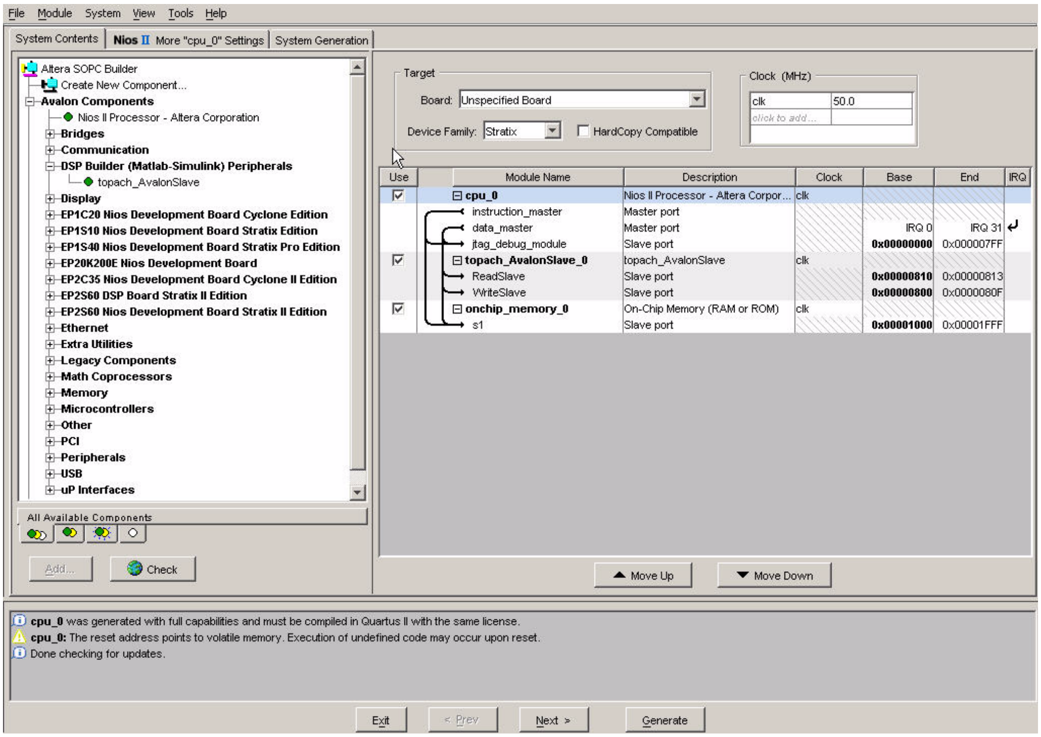
6. Repeat step 5 for other components you would like to add to the system.

7. Connect the master or slave port of your customized component to the other peripherals.

For a complete tutorial on SOPC Builder, see the *Quartus II Handbook Volume 4: SOPC Builder.*

Figure 10 shows an SOPC Builder system that contains an Avalon Slave peripheral (two slaves) connected to a Nios® II processor (master) and on-chip memory (slave).

Figure 10. SOPC Builder User Interface



- Altera SOPC Builder
  - Create New Component...
  - Avalon Components
    - Nios II Processor - Altera Corporation
    - Bridges
    - Communication
    - DSP Builder (Matlab-Simulink) Peripherals
      - topach_AvalonSlave
    - Display
    - EP1C20 Nios Development Board Cyclone Edition
    - EP1S10 Nios Development Board Stratix Edition
    - EP1S40 Nios Development Board Stratix Pro Edition
    - EP20K200E Nios Development Board
    - EP2C35 Nios Development Board Cyclone II Edition
    - EP2S60 DSP Board Stratix II Edition
    - EP2S60 Nios Development Board Stratix II Edition
    - Ethernet
    - Extra Utilities
    - Legacy Components
    - Math Coprocessors
    - Memory
    - Microcontrollers
    - Other
    - PCI
    - Peripherals
    - USB
    - uP Interfaces

Target
Board: Unspecified Board
Device Family: Stratix
HardCopy Compatible

Clock (MHz)

clk	50.0
click to add ...	

Use	Module Name	Description	Clock	Base	End	IRQ
✓	cpu_0	Nios II Processor - Altera Corpor...	clk			
	instruction_master	Master port				
	data_master	Master port		IRQ 0	IRQ 31	
	jtag_debug_module	Slave port		0x00000000	0x000007FF	
✓	topach_AvalonSlave_0	topach_AvalonSlave	clk			
	ReadSlave	Slave port		0x00000810	0x00000813	
	WriteSlave	Slave port		0x00000800	0x0000080F	
✓	onchip_memory_0	On-Chip Memory (RAM or ROM)	clk			
	s1	Slave port		0x00001000	0x00001FFF	

```
cpu_0 was generated with full capabilities and must be compiled in Quartus II with the same license.
cpu_0: The reset address points to volatile memory. Execution of undefined code may occur upon reset.
Done checking for updates.
```

For the peripheral to appear in the SOPC Builder, the working directory for your SOPC Builder project must be the same as your DSP Builder working directory.

For information on how to use SOPC Builder to create Nios II designs, see the *Nios II Hardware Development Tutorial*.

Conclusion

The Avalon Master and Slave feature gives you a simple and easy way to integrate your blocks into SOPC Builder without being exposed to the complexity of the interface. Depending on your needs, you can specify which mode of the Avalon switch fabric to use without connecting each port manually. Also, with the additional support of the Avalon block, you no longer need to code the standard glue logic in a hardware description language. This feature makes the DSP Builder design environment more conducive to creating larger and more complex system level designs.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
literature@altera.com

I.S. EN ISO 9001