DSP Builder Handbook

Volume 2: DSP Builder Standard Blockset

Feedback

Contents

Section I. DSP Builder Standard Blockset User Guide

Chapter 1. About DSP Builder Standard Blockset

Chapter 2. Getting Started

Chapter 3. Design Rules and Procedures

Chapter 10. Adding a Board Library

Chapter 11. Using the State Machine Library

Chapter 12. Managing Projects and Files

Chapter 13. Troubleshooting

Section II. DSP Builder Standard Blockset Libraries

Chapter 14. AltLab Library

Chapter 15. Arithmetic Library

Chapter 16. Boards Library

Chapter 17. Complex Type Library

Chapter 18. Gate & Control Library

Chapter 19. Interfaces Library

Chapter 20. IO & Bus Library

Chapter 21. MegaCore Functions Library

Chapter 22. Rate Change Library

Chapter 23. Simulation Library

Chapter 24. State Machine Functions Library

Chapter 25. Storage Library

Chapter 26. Design Examples

Additional Information

Section I. DSP Builder Standard Blockset User Guide

☞ Note: DSP Builder standard blockset is a legacy product and Altera recommends you do not use it for new designs, except as a wrapper for advanced blockset designs.

Device Family Support

DSP Builder supports the following Altera® device families:

- Arria V
- Arria V GZ
- Arria 10
- Cyclone IV GX
- Cyclone IV E
- Cyclone V
- Stratix® IV
- Stratix V

Features

DSP Builder standard blockset supports the following features:

- Altera DSP MegaCore® functions in a DSP Builder design model.
- Fixed-point arithmetic and logical operators for use with the Simulink software.
- Avalon® Memory-Mapped (Avalon-MM) interfaces including user configurable blocks, which you can use to build custom logic that works with the Nios® II processor and other Qsys designs.
- Avalon Streaming (Avalon-ST) interfaces including an Packet Format Converter block and configurable Avalon-ST sink and Avalon-ST source blocks.
- VHDL testbench.
- Rapid prototyping using Altera development boards.
- The SignalTap® II logic analyzer—an embedded signal analyzer that probes signals from the Altera device on the DSP board and imports the data into the MATLAB workspace to ease visual analysis.
- HDL import of VHDL or Verilog HDL design entities and HDL defined in a Quartus Prime project file.
- Hardware-in-the loop (HIL) to enable FPGA hardware accelerated cosimulation with Simulink.
- Tabular and graphical state machine editing.

For information about new features and errata in this release, refer to the *DSP Builder Release Notes*.

General Description

Digital signal processing (DSP) system design in Altera FPGAs requires both high-level algorithm and hardware description language (HDL) development tools.

Altera's DSP Builder integrates these tools by combining the algorithm development, simulation, and verification capabilities of The MathWorks MATLAB and Simulink system-level design tools with VHDL and Verilog HDL design flows, including the Altera Quartus Prime software.

DSP Builder shortens DSP design cycles by helping you create the hardware representation of a DSP design in an algorithm-friendly development environment.

You can combine existing MATLAB functions and Simulink blocks with Altera DSP Builder blocks and Altera intellectual property (IP) MegaCore functions to link system-level design and implementation with DSP algorithm development. In this way, DSP Builder allows system, algorithm, and hardware designers to share a common development platform.

The DSP Builder `Signal Compiler` block reads Simulink Model Files (.**mdl**) that contain other DSP Builder blocks and MegaCore functions. `Signal Compiler` then generates the VHDL files and Tcl scripts for synthesis, hardware implementation, and simulation.

You can use blocks from the standard blockset to create a hardware implementation of a system modeled in Simulink. DSP Builder contains bit- and cycle-accurate Simulink blocks—which cover basic operations such as arithmetic or storage functions—and takes advantage of key device features such as built-in PLLs, DSP blocks, and embedded memory.

You can integrate complex functions by including IP cores in your DSP Builder model. You can also use the faster performance and richer instrumentation of hardware cosimulation by implementing parts of your design in an FPGA.

The standard blockset supports imported HDL subsystems including HDL defined in a Quartus Prime project file.

High-Speed DSP with FPGAs

FPGAs give compelling performance advantage over dedicated DSP devices. You can configure FPGA's element arrays as complex processor routine.

You can link these routines together in serial (the same way that a DSP processor executes them), or connect them in parallel. When connected in parallel, they give many times better performance than standard DSP devices by executing hundreds of instructions at the same time.

Algorithms that benefit from this improved performance include forward-error correction (FEC), modulation and demodulation, and encryption.

Interoperability with the Advanced Blockset

Any standard blockset design can include an advanced blockset design as a single hierarchical entity.

For more information about the advanced blockset, refer to *Volume 3: DSP Builder Advanced Blockset* in the *DSP Builder Handbook*.

For more information about the differences between the standard and advanced blocksets and about design flows that combine both blocksets, refer to *Volume 1: Introduction to DSP Builder* in the *DSP Builder Handbook*.

This chapter describes the design flow and a getting started tutorial.

Creating a Design in DSP Builder

1. Create a Simulink design model in the MATLAB software.

2. Compile directly in the Quartus Prime software.

3. Output VHDL files for synthesis and Quartus II compilation or generate files for VHDL simulation.

☞ DSP Builder generates VHDL and does not generate Verilog HDL.

4. Use the `quartus_map` command in the Quartus Prime software to run a simulation netlist flow that generates files for Verilog HDL simulation.

For information about this flow, refer to the Quartus Prime help.

Design Flow

Figure 2–1 shows the system-level design flow using DSP Builder.

Figure 2–1. System-Level Design Flow
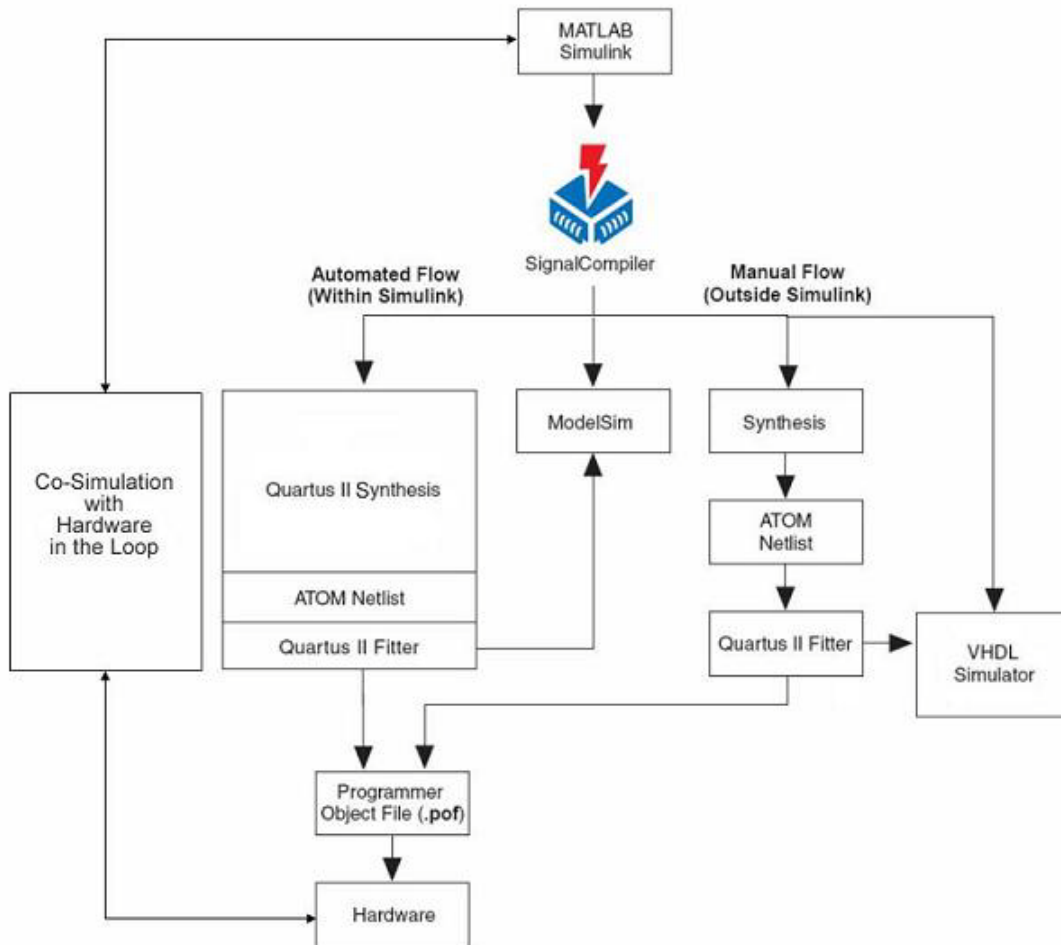
The design flow involves the following steps:

1. Use the MathWorks software to create a model with a combination of Simulink and DSP Builder blocks.

    ☞ Separate The DSP Builder blocks in your design from the Simulink blocks by `Input` and `Output` blocks from the DSP Builder IO and Bus library.

2. Include a `Clock` block from the DSP Builder AltLab library to specify the base clock for your design, which must have a period greater than 1ps but less than 2.1 ms.

    ☞ If no base clock exists in your design, DSP Builder creates a default clock with a 20ns real-world period and a Simulink sample time of 1. You can derive additional clocks from the base clock by adding `Clock_Derived` blocks.

3. Set a discrete (no continuous states) solver in Simulink. Choose a **Fixed-step** solver type if you are using a single clock domain or a **Variable-step** type if you use multiple clock domains.

    To set the solver options, click **Configuration Parameters** on the Simulation menu to open the **Configuration Parameters** dialog box and select the **Solver** page (Figure 2–2).

Figure 2–2. Configuration Parameters for Simulation
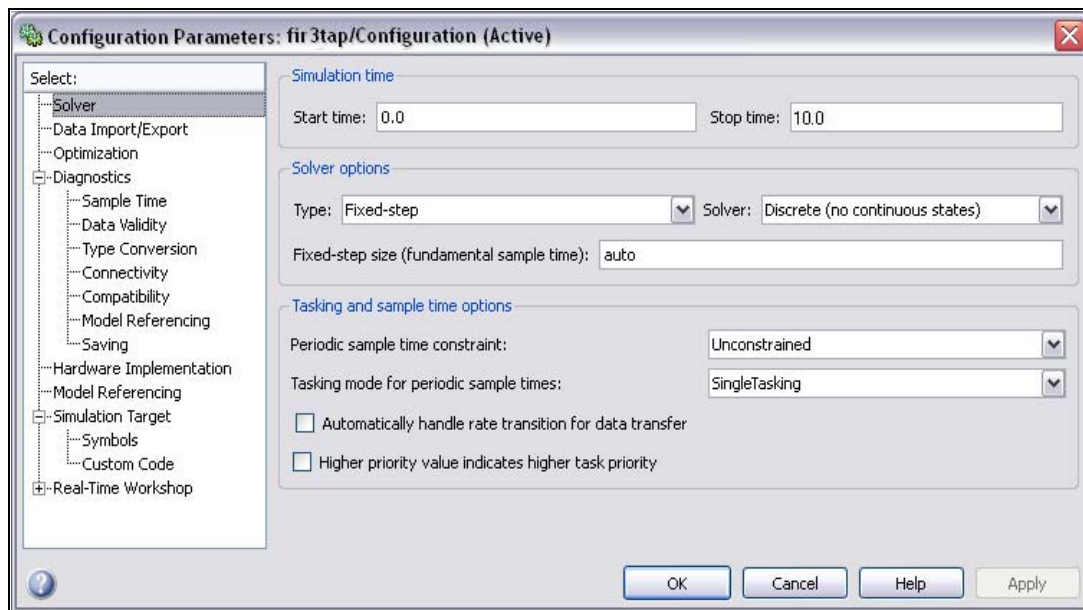
> For detailed information about solver options, refer to the description of the "Solver Pane" in the Simulink Help.

4. Simulate your model in Simulink using a `Scope` block to monitor the results.

5. Run `Signal Compiler` to setup RTL simulation and synthesis.

6. Perform RTL simulation. DSP Builder supports an automated flow for the ModelSim software (using the `TestBench` block). You can also use the generated VHDL for manual simulation in other simulation tools.

7. Use the output files generated by the DSP Builder `Signal Compiler` block to perform RTL synthesis. Alternatively, you can synthesize the VHDL files manually using other synthesis tools.

8. Compile your design in the Quartus II software.

9. Download to a hardware development board and test.

For an automated design flow, the `Signal Compiler` block generates VHDL and Tcl scripts for synthesis in the Quartus II software. The Tcl scripts let you perform synthesis and compilation automatically in the MATLAB and Simulink environment. You can synthesize and simulate the output files in other software tools without the Tcl scripts. In addition, the `Testbench` block generates a testbench and supporting files for VHDL simulation.

For information about controlling the DSP Builder design flow using `Signal Compiler`, refer to "Design Flows for Synthesis, Compilation and Simulation" on page 3–19.

For more information about the blocks in the DSP Builder blockset, refer to the *DSP Builder Standard Blockset Libraries* section in volume 2 of the *DSP Builder Handbook*.

Creating an Amplitude Modulation Design Example

The amplitude modulation design example, **singen.mdl**, demonstrates the DSP Builder design flow.

The amplitude modulation design example is a modulator that has a sine wave generator, a quadrature multiplier, and a delay element. Each block in the model is parameterizable. When you double-click a block in the model, a dialog box displays where you can enter the parameters for the block. Click the **Help** button in these dialog boxes to view help for a specific block.

This tutorial assumes the following points:

■ You are using a PC running Windows.

■ You are familiar with the MATLAB, Simulink, Quartus II, and ModelSim® software and the software is installed on your PC in the default locations.

■ You have basic knowledge of the Simulink software.

For information about using the Simulink software, refer to the Simulink Help.

You can use the **singen.mdl** model file in *<DSP Builder install path>***\DesignExamples\Tutorials\GettingStartedSinMdl** or you can create your own amplitude modulation model.

Creating a New Model

To create a new model:

1. Start the MATLAB software.

2. On the File menu, point to **New,** and click **Model** to create a new model window.

3. In the new model window, on the File menu click **Save**.

4. Browse to a directory, your working directory, to save the file. This tutorial uses the working directory *<DSP Builder install path>***\DesignExamples\Tutorials\GettingStartedSinMdl\my_SinMdl**.

5. Type the file name into the **File name** box. This tutorial uses the name **singen.mdl**.

6. Click **Save**.

7. Click the MATLAB **Start** button. Point to **Simulink** and click **Library Browser**.

Adding the Sine Wave Block

To add the `Sine Wave` block:

1. In the Simulink Library Browser, click **Simulink** and **Sources** to view the blocks in the Sources library.

2. Drag and drop a `Sine Wave` block into your model.

3. Double-click the `Sine Wave` block in your model to display the **Block Parameters** dialog box (Figure 2–3).

Figure 2–3. 500-kHz, 16-Bit Sine Wave Specified in the Sine Wave Dialog Box
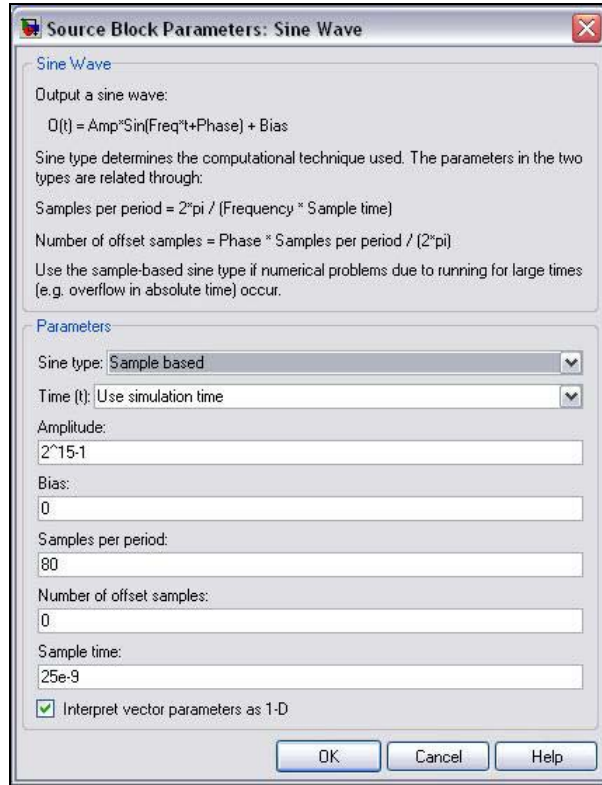
4. Set the `Sine Wave` block parameters (Table 2–1).

Table 2–1. Parameters for the Sine Wave Block

Parameter	Value
Sine type	Sample based
Time	simulation time
Amplitude	2^15–1
Bias	0
Samples per period	80
Number of offset examples	0
Sample time	25e-9
Interpret vector parameters a 1-D	On

5. Click **OK**.

☞ For information about how you can calculate the frequency., refer to the equation in "Frequency Design Rules" on page 3–8.

Adding the SinIn Block

To add the `SinIn` block:

1. In the Simulink Library Browser, expand the **Altera DSP Builder Blockset** folder to display the DSP Builder libraries (Figure 2–4).

Figure 2–4. Altera DSP Builder Folder in the Simulink Library Browser
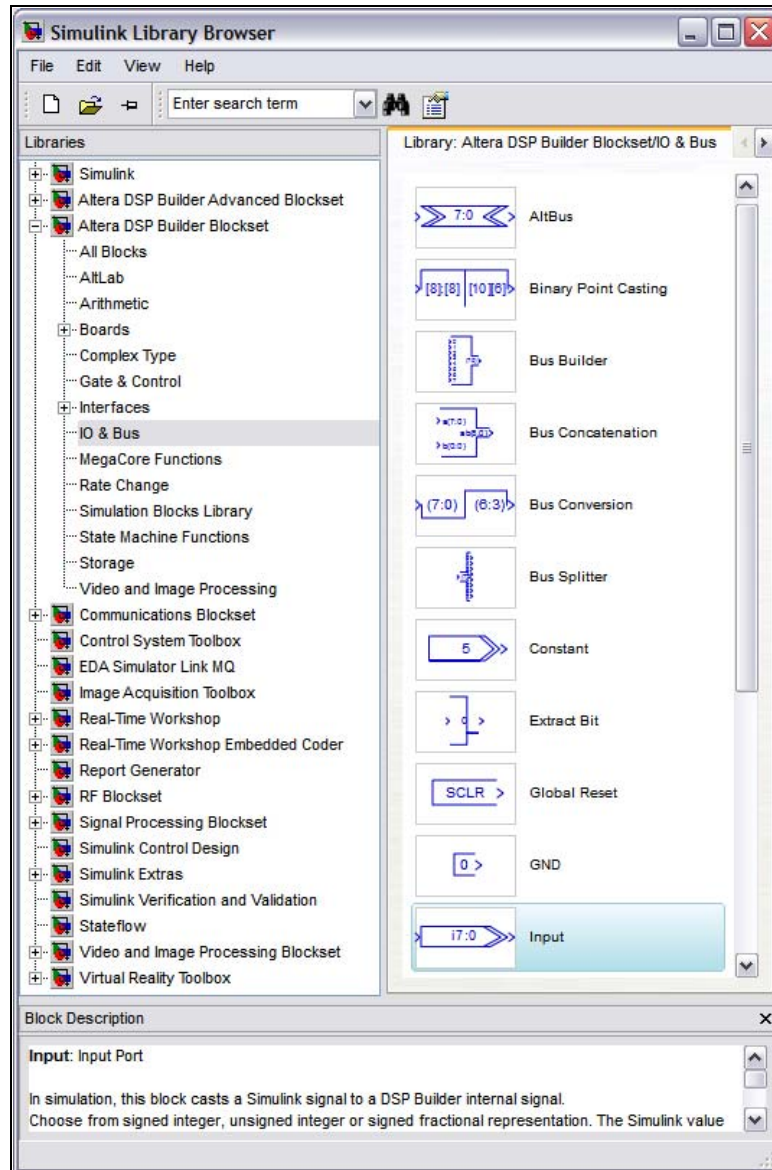
2. Select the **IO & Bus** library.

3. Drag and drop the `Input` block from the Simulink Library Browser into your model. Position the block to the right of the `Sine Wave` block.

   If you are unsure how to position the blocks or draw connection lines, refer to the completed design (Figure 2–7 on page 2–14).

   ☞ You can use the Up, Down, Right, and Left arrow keys to adjust the position of a block.

4. Click the text under the block icon in your model. Delete the text `Input` and type the text `SinIn` to change the name of the block instance.

5. Double-click the `SinIn` block in your model to display the **Block Parameters** dialog box.

6. Set the `SinIn` block parameters (Table 2–2).

Table 2–2. Parameters for the SinIn Block

Parameter	Value
Bus Type	Signed Integer
[number of bits].[]	16
Specify Clock	Off

7. Click **OK**.

8. Draw a connection line from the right side of the `Sine Wave` block to the left side of the `SinIn` block by holding down the left mouse button and dragging the cursor between the blocks.

   ☞ Alternatively, you can select a block, hold down the Ctrl key and click the destination block to automatically make a connection between the two blocks.

Adding the Delay Block

To add the `Delay` block:

1. Select the **Storage** library from the **Altera DSP Builder Blockset** folder in the Simulink Library Browser.

2. Drag and drop the `Delay` block into your model and position it to the right of the `SinIn` block.

3. Double-click the `Delay` block in your model to display the **Block Parameters** dialog box (Figure 2–5).

4. Type `1` as the **Number of Pipeline Stages** for the `Delay` block.

Figure 2–5. Setting the Downsampling Delay
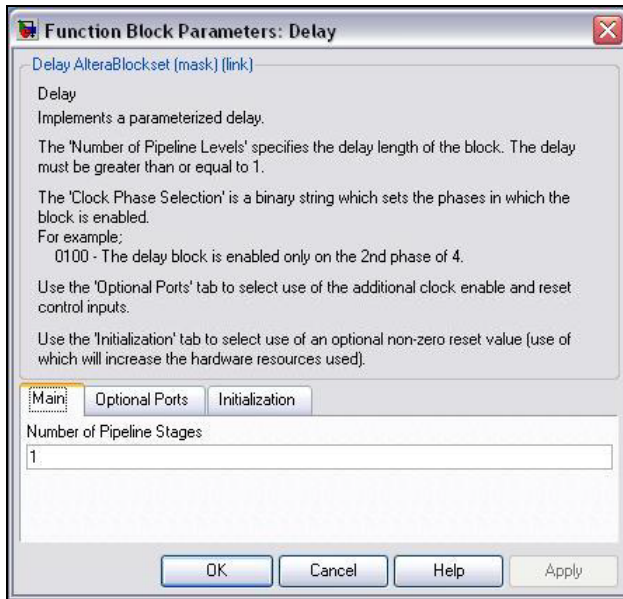
5. Click the **Optional Ports** tab and set the parameters (Table 2–3).

Table 2–3. Parameters for the Delay Block.

Parameter	Value
Clock Phase Selection	01
Use Enable Port	Off
Use Synchronous Clear port	Off

6. Click **OK**.

7. Draw a connection line from the right side of the `SinIn` block to the left side of the `Delay` block.

Adding the SinDelay and SinIn2 Blocks

To add the `SinDelay` and `SinIn2` blocks:

1. Select the **IO & Bus** library from the **Altera DSP Builder Blockset** folder in the Simulink Library Browser.

2. Drag and drop two `Output` blocks into your model, positioning them to the right of the `Delay` block.

3. Click the text under the block symbols in your model. Change the block instance names from `Output` and `Output1` to `SinDelay` and `SinIn2`.

4. Double-click the `SinDelay` block in your model to display the **Block Parameters** dialog box.

5. Set the `SinDelay` block parameters (Table 2–4).

Table 2–4. Parameters for the SinDelay Block

Parameter	Value
Bus Type	Signed Integer
[number of bits].[]	16
External Type	Inferred

6. Click **OK**.

7. Repeat steps 4 to 6 for the `SinIn2` block setting the parameters (Table 2–5).

Table 2–5. Parameters for the SinIn2 Block

Parameter	Value
Bus Type	Signed Integer
[number of bits].[]	16
External Type	Inferred

8. Draw a connection line from the right side of the `Delay` block to the left side of the `SinDelay` block.

Adding the Mux Block

To add the `Mux` block, follow these steps:

1. Select the Simulink **Signal Routing** library in the Simulink Library Browser.

2. Drag and drop a `Mux` block into your design, positioning it to the right of the `SinDelay` block.

3. Double-click the `Mux` block in your model to display the **Block Parameters** dialog box.

4. Set the `Mux` block parameters (Table 2–6).

Table 2–6. Parameters for the Mux Block

Parameter	Value
Number of Inputs	2
Display Options	bar

5. Click **OK**.

6. Draw a connection line from the bottom left of the `Mux` block to the right side of the `SinDelay` block.

7. Draw a connection line from the top left of the `Mux` block to the line between the `SinIn2` block.

8. Draw a connection line from the `SinIn2` block to the line between the `SinIn` and `Delay` blocks.

Adding the Random Bitstream Block

To add the `Random Bitstream` block, follow these steps:

1. Select the Simulink **Sources** library in the Simulink Library Browser.

2. Drag and drop a `Random Number` block into your model, positioning it underneath the `Sine Wave` block.

3. Double-click the `Random Number` block in your model to display the **Block Parameters** dialog box.

4. Set the `Random Number` block parameters (Table 2–7).

Table 2–7. Parameters for the Random number Block

Parameter	Value
Mean	0
Variance	1
Initial seed	0
Sample time	25e–9
Interpret vector parameters as 1-D	On

5. Click **OK**.

6. Rename the `Random Noise` block `Random Bitstream`.

Adding the Noise Block

To add the `Noise` block, follow these steps:

1. Select the **IO & Bus** library from the **Altera DSP Builder Blockset** folder in the Simulink Library Browser.

2. Drag and drop an `Input` block into your model, positioning it to the right of the `Random Bitstream` block.

3. Click the text under the block icon in your model. Rename the block `Noise`.

4. Double-click the `Noise` block to display the **Block Parameters** dialog box.

5. Set the `Noise` block parameters (Table 2–8).

Table 2–8. Parameters for the Noise Block

Parameter	Value
Bus Type	Single Bit
Specify Clock	Off

☞ The dialog box options change to display only the relevant options when you select a new bus type.

6. Click **OK**.

7. Draw a connection line from the right side of the `Random Bitstream` block to the left side of the `Noise` block.

Adding the Bus Builder Block

The `Bus Builder` block converts a bit to a signed bus. To add the `Bus Builder` block, follow these steps:

1. Select the **IO & Bus** library from the **Altera DSP Builder Blockset** folder in the Simulink Library Browser.

2. Drag and drop a `Bus Builder` block into your model, positioning it to the right of the `Noise` block.

3. Double-click the `Bus Builder` block in your model to display the **Block Parameters** dialog box.

4. Set the `Bus Builder` block parameters (Table 2–9).

Table 2–9. Parameters for the Bus Builder Block

Parameter	Value
Bus Type	Signer Integer
[number of bits].[]	2

5. Click **OK**.

6. Draw a connection line from the right side of the `Noise` block to the top left side of the `Bus Builder` block.

Adding the GND Block

To add the `GND` block, follow these steps:

1. Select the **IO & Bus** library from the **Altera DSP Builder Blockset** folder in the Simulink Library Browser.

2. Drag and drop a `GND` block into your model, positioning it underneath the `Noise` block.

3. Draw a connection line from the right side of the `GND` block to the bottom left side of the `Bus Builder` block.

Adding the Product Block

To add the `Product` block, follow these steps:

1. Select the **Arithmetic** library from the **Altera DSP Builder Blockset** folder in the Simulink Library Browser.

2. Drag and drop a `Product` block into your model, positioning it to the right of the `Bus Builder` block and slightly above it. Leave enough space so that you can draw a connection line under the `Product` block.

3. Double-click the `Product` block to display the **Block Parameters** dialog box.

4. Set the `Product` block parameters (Table 2–10).

Table 2–10. Parameters for the Product Block

Parameter	Value
Bus Type	Inferred
Number of Pipeline Stages	0

☞ The bit width parameters are set automatically when you select **Inferred** bus type. The parameters in the **Optional Ports and Settings** tab of this dialog box can be left with their default values.

5. Click **OK**.

6. Draw a connection line from the top left of the `Product` block to the line between the `Delay` and `SinDelay` blocks.

Adding the StreamMod and StreamBit Blocks

To add the `StreamMod` and `StreamBit` blocks, follow these steps:

1. Select the **IO & Bus** library from the **Altera DSP Builder Blockset** folder in the Simulink Library Browser.

2. Drag and drop two `Output` blocks into your model, positioning them to the right of the `Product` block.

3. Click the text under the block symbols in your model. Change the block instance names from `Output` and `Output1` to `StreamMod` and `StreamBit`.

4. Double-click the `StreamMod` block to display the **Block Parameters** dialog box.

5. Set the `StreamMod` block parameters (Table 2–11).

Table 2–11. Parameters for the StreamMod Block

Parameter	Value
Bus Type	Signed Integer
[number of bits].[]	19
External Type	Inferred

6. Click **OK**.

7. Double-click the `StreamBit` block to display the **Block Parameters** dialog box (Figure 2–6).

Figure 2–6. Set a Single-Bit Output Bus
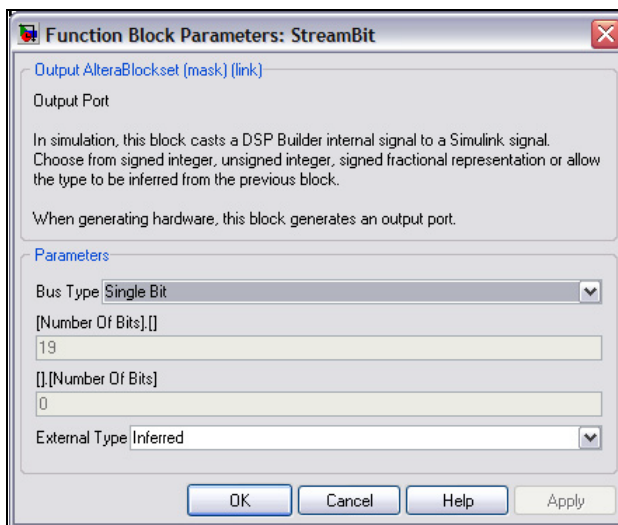
8. Set the `StreamBit` block parameters (Table 2–12).

Table 2–12. Parameters for the StreamBit Block

Parameter	Value
Bus Type	Single Bit
External Type	Inferred

9. Draw connection lines from the right side of the `Product` block to the left side of the `StreamMod` block, and from the right side of the `Bus Builder` block to the left side of the `StreamBit` block.

Adding the Scope Block

To add the `Scope` block, follow these steps:

1. Select the Simulink **Sinks** library in the Simulink Library Browser.

2. Drag and drop a `Scope` block into your model and position it to the right of the `StreamMod` block.

3. Double-click the `Scope` block and click the **Parameters** icon to display the **'Scope' parameters** dialog box.

4. Set the `Scope` parameters (Table 2–13).

Table 2–13. Parameters for the Scope Block

Parameter	Value
Number of axes	3
Time range	auto

Table 2–13. Parameters for the Scope Block

Parameter	Value
Tick labels	bottom axis only
Sampling	Decimation 1

5. Click **OK**.

6. Close the Scope.

7. Make connections to connect the complete your design as follows:

    a. From the right side of the `Mux` block to the top left side of the `Scope` block.

    b. From the right side of the `StreamMod` block to the middle left side of the `Scope` block.

    c. From the right side of the `StreamBit` block to the bottom left of the `Scope` block.

    d. From the bottom left of the `Product` block to the line between the `Bus Builder` block and the `StreamBit` block.

    Figure 2–7 shows the required connections.

Figure 2–7. Amplitude Modulation Design Example
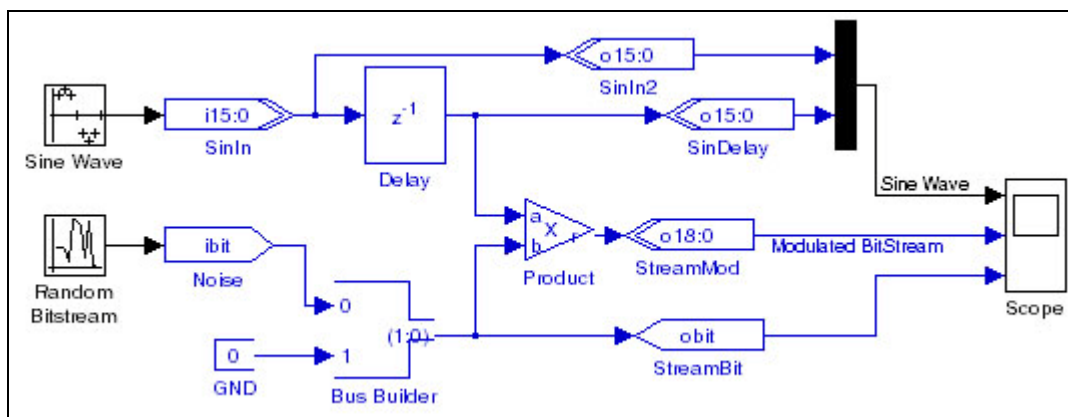
Adding a Clock Block

To add a `Clock` block, follow these steps:

1. Select the **AltLab** library from the **Altera DSP Builder Blockset** folder in the Simulink Library Browser.

2. Drag and drop a `Clock` block into your model.

3. Double-click on the `Clock` block to display the **Block Parameters** dialog box.

4. Set the Clock parameters (Table 2–14).

Table 2–14. Parameters for the Clock Block

Parameter	Value
Real-World Clock Period	20
Period Unit:	ns

Table 2–14. Parameters for the Clock Block

Parameter	Value
Simulink Sample Time	2.5e–008
Reset Name	aclr
Reset Type	Active Low
Export As Output Pin	Off

☞ A `clock` block is required to set a Simulink sample time that matches the sample time specified on the `Sine Wave` and `Random Bitstream` blocks. If no base clock exists in your design, a default clock with a 20ns real-world period and a Simulink sample time of 1 is automatically created.

5. Save your model.

Simulating the Model in Simulink

To simulate your model in the Simulink software, follow these steps:

1. Click **Configuration Parameters** on the Simulation menu to display the **Configuration Parameters** dialog box and select the **Solver** page (Figure 2–8 on page 2–16).

2. Set the parameters (Table 2–15).

Table 2–15. Configuration Parameters for the singen Model

Parameter	Value
Start time	0.0
Stop time	4e–6
Type	Fixed-step
Solver	discrete (no continuous states)

> For detailed information about solver options, refer to the description of the
> Solver Pane in the Simulink Help.

Figure 2–8. Configuration Parameters
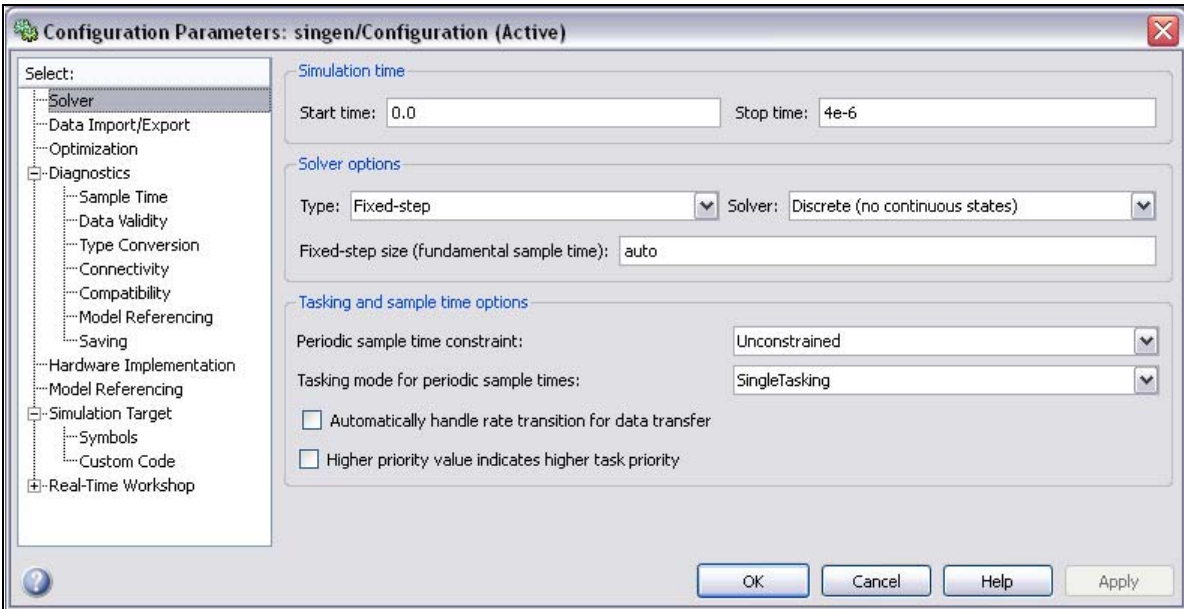
3. Click **OK**.

4. Start simulation by clicking **Start** on the Simulation menu.

5. Double-click the Scope block to view the simulation results.

6. Click the **Autoscale** icon (binoculars) to auto-scale the waveforms.

Figure 2–9 shows the scaled waveforms.

Figure 2–9. Scope Simulation Results
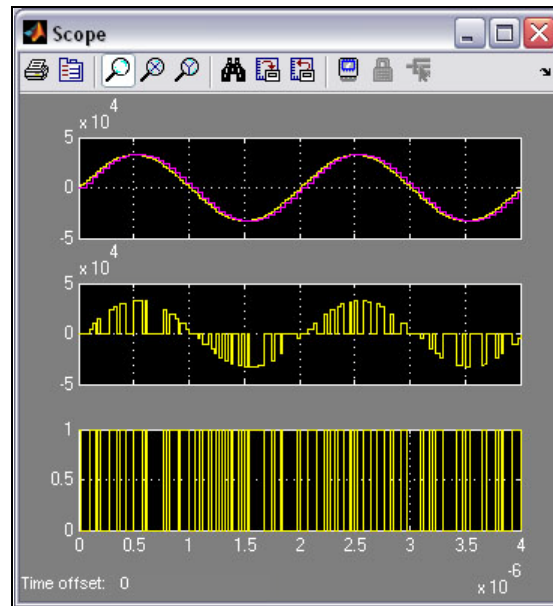
Compiling the Design

To create and compile a Quartus Prime project for your DSP Builder design, and to program your design onto an Altera FPGA, add a `Signal Compiler` block by following these steps:

1. Select the **AltLab** library from the **Altera DSP Builder Blockset** folder in the Simulink Library Browser.

2. Drag and drop a `Signal Compiler` block into your model.

3. Double-click the `Signal Compiler` block in your model to display the **Signal Compiler** dialog box (Figure 2–10).

The dialog box allows you to set the target device family. For this tutorial, you can use the default **Stratix** device family.

4. Click **Compile**.

Figure 2–10. Signal Compiler Block Dialog Box
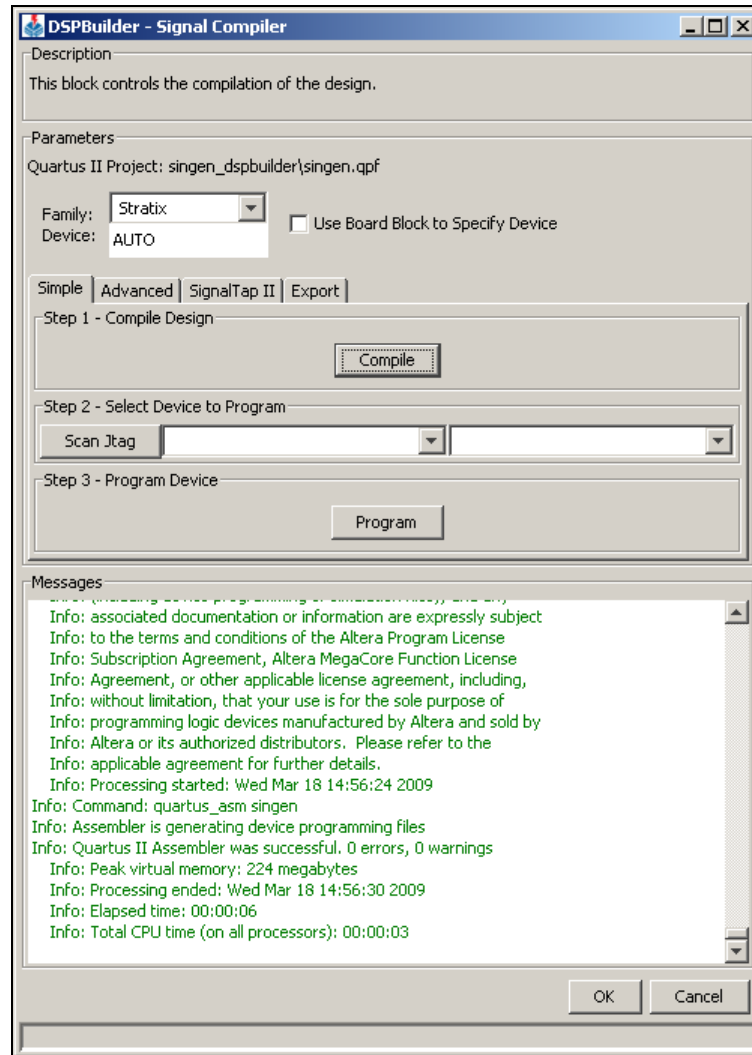
5. When the compilation completes successfully, click **OK**.

6. Click **Save** on the File menu to save your model.

Performing RTL Simulation

To perform RTL simulation with the ModelSim software, add a `TestBench` block, by
following these steps:

1. Select the **AltLab** library from the **Altera DSP Builder BlockSet** folder in the
    Simulink Library Browser.

2. Drag and drop a `TestBench` block into your model.

3. Double-click on the new `TestBench` block.

   The **Testbench Generator** dialog box appears (Figure 2–11).

Figure 2–11. Testbench Generator Dialog Box
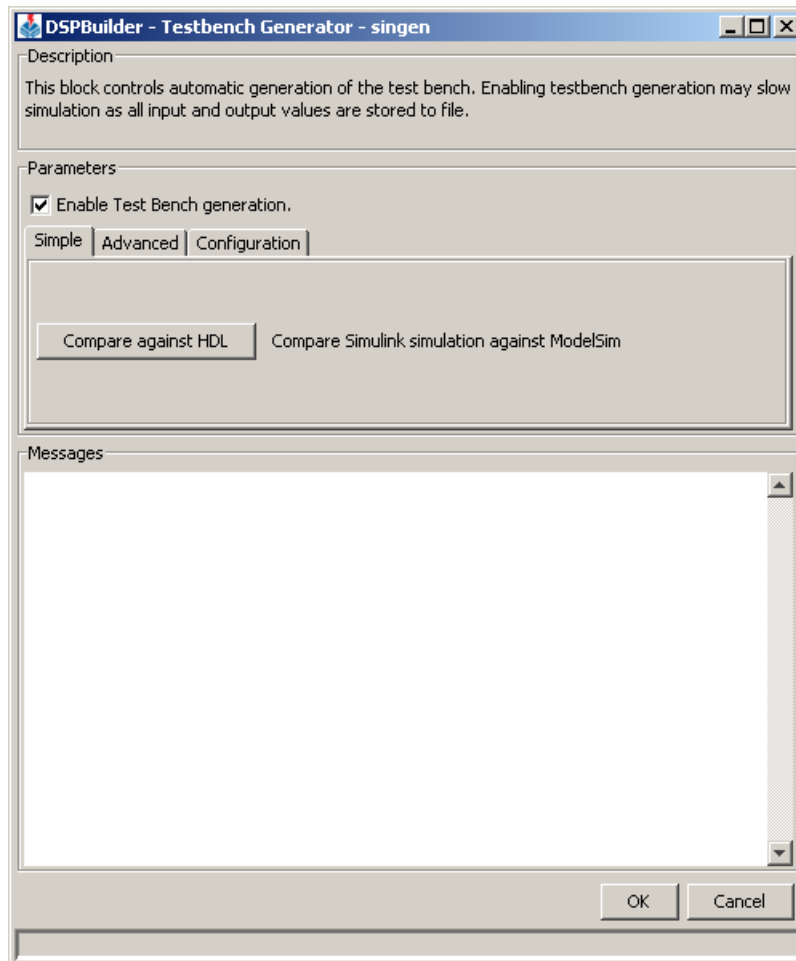
4. Ensure that **Enable Test Bench generation** is on.

5. Click the **Advanced** tab (Figure 2–12).

Figure 2–12. Testbench Generator Dialog Box Advanced Tab
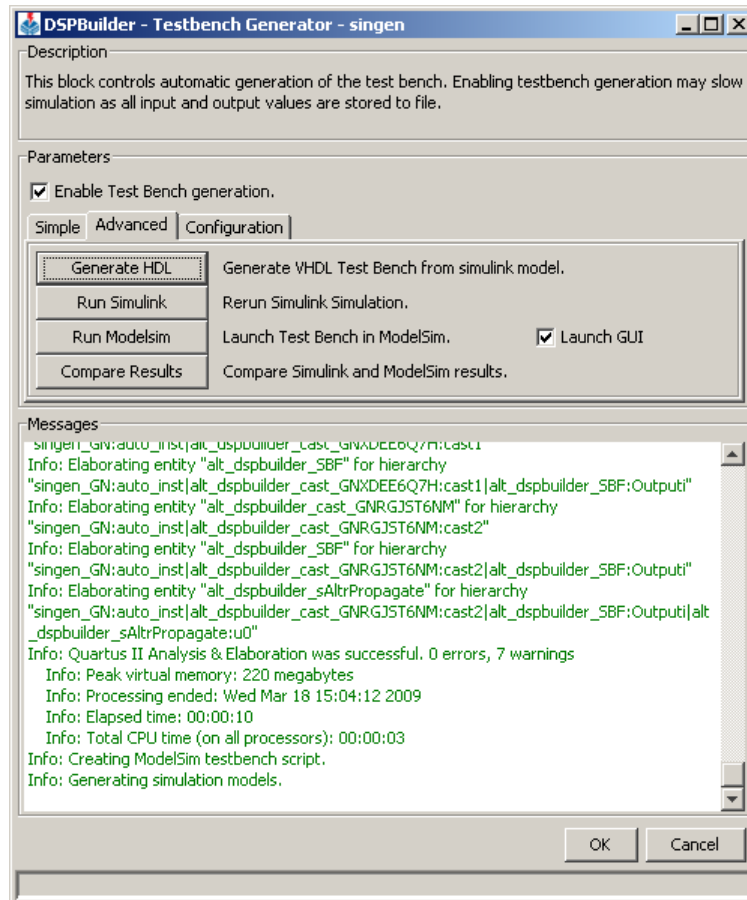
6. Turn on the **Launch GUI** option. This option causes the ModelSim GUI to launch when you invoke the ModelSim simulation.

7. Click **Generate HDL** to generate a VDHL-based testbench from your model.

8. Click **Run Simulink** to generate Simulink simulation results for the testbench.

9. Click **Run ModelSim** to load your design into ModelSim.

   Your design simulates with the output displaying in the ModelSim Wave window. The testbench initializes all your design registers with a pulse on the `aclr` input signal.

10. All waveforms initially show using digital format in the ModelSim Wave window. Change the format of the `sinin`, `sindelay` and `streammod` signals to analog.

☞ In ModelSim 6.4a, you can right-click to display the popup menu, point to **Format** and click on **Analog (Automatic)**. The user interface commands may be different in other versions of ModelSim.

11. Click **Zoom Full** on the right button pop-up menu in the ModelSim Wave window. The simulation results display as an analog waveform (Figure 2–13).

Figure 2–13. Analog Display
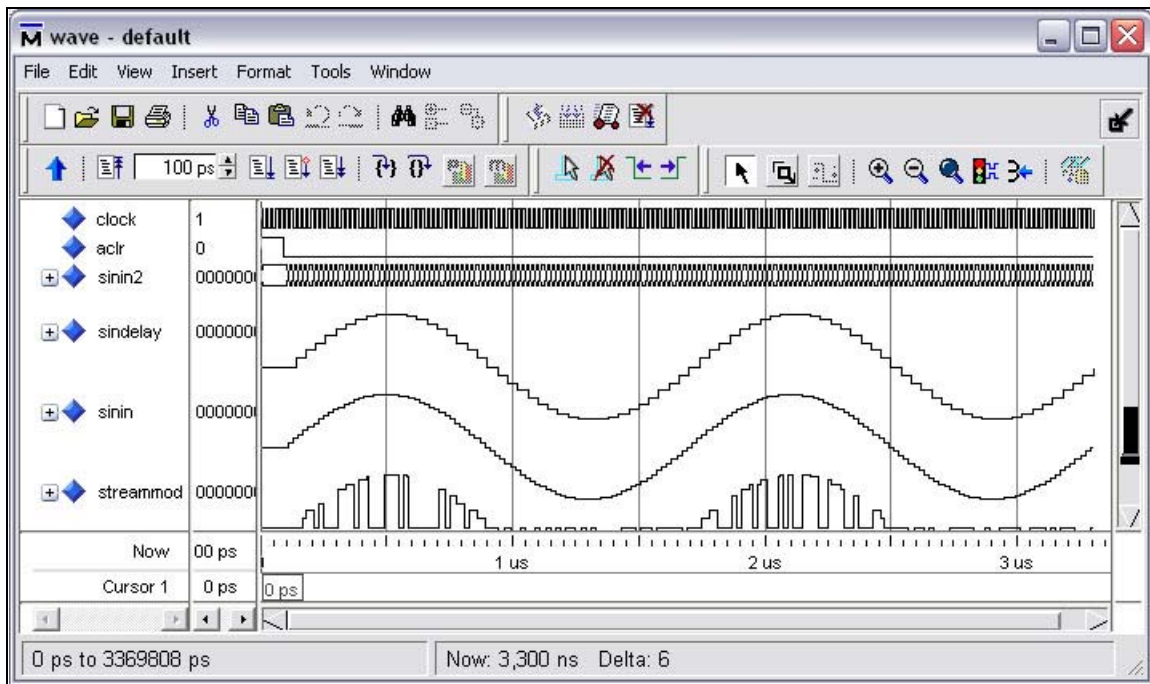
The introductory DSP Builder tutorial is complete. The next section shows how you can add a DSP Builder design to a new or existing Quartus Prime project.

Subsequent chapters in this user guide provide examples that illustrate some of the additional design features supported by DSP Builder.

Adding the Design to a Quartus Prime Project

DSP Builder uses the Quartus Prime project created by the Signal Compiler block. You can add your design to a new or existing Quartus Prime project.

Creating a Quartus Prime Project

To create a new Quartus Prime project:

1. Start the Quartus Prime software.

2. Click **New Project Wizard** on the File menu in the Quartus Prime software and specify the working directory for your project. For example, **D:\MyQuartusProject**.

3. Specify the name of the project. For example, **NewProject** and the name of the top-level design entity for the project.

   ☞ The name of the top-level design entity typically has the same name as the project.

4. Click **Next** to display the **Add Files** page. There are no files to add for this tutorial.

5. Click **Next** to display the **Family & Device Settings** page and check that the required device family is selected. This should normally be the same device family as specified for `Signal Compiler` in "Compiling the Design" on page 2–17.

6. Click **Finish** to close the wizard and create the new project.

☞ When you specify a directory that does not already exist, a message asks if the specified directory should be created. Click **Yes** to create the directory.

Adding the DSP Builder Design to the Project

To add your DSP Builder design to the project in the Quartus Prime software:

1. On the View menu in the Quartus Prime software, point to **Utility Windows** and click **Tcl Console** to display the Tcl Console.

2. Run the **singen_add.tcl** script that can be found in the <*DSP Builder install path*>**DesignExamples\Tutorials\GettingStartedSinMdl** directory by typing the following command in the Tcl Console window:

```
# source <install path>/DesignExamples/Tutorials/GettingStartedSinMdl/singen_add.tcl
```

☞ You must use / separators instead of \ separators in the command path name used in the Tcl console window. You can use a relative path if you organize your design data with the DSP Builder and Quartus Prime designs in subdirectories of the same design hierarchy.

An example instantiation is added to your Quartus Prime project.

3. Click the **Files** tab in the Quartus Prime software.

4. Right-click **singen.mdl** and click **Select Set as Top-Level Entity**.

5. Compile the Quartus Prime design by clicking **Start Compilation** on the Processing menu.

☞ You can copy the component declaration from the example file for your own code.

This chapter discusses the following topics:

- "DSP Builder Naming Conventions"
- "MATLAB Variables"
- "Fixed-Point Notation"
- "Bit Width Design Rule"
- "Frequency Design Rules"
- "About Timing Semantics Between Simulink and HDL Simulation"
- "Signal Compiler and TestBench Blocks"
- "Hierarchical Design"
- "Goto and From Block Support"
- "Black Box and HDL Import"
- "Using a MATLAB Array or .hex File to Initialize a Block"
- "Comparison Utility"
- "Adding Comments to Blocks"
- "Adding Quartus II Constraints"
- "Displaying Port Data Types"
- "Displaying the Pipeline Depth"
- "Updating HDL Import Blocks"
- "Analyzing the Hardware Resource Usage"
- "Loading Additional ModelSim Commands"
- "About Quartus II Assignments and Block Entity Names"

DSP Builder Naming Conventions

DSP Builder generates VHDL files for simulation and synthesis. When blocks or ports in your model share the same VHDL name, DSP Builder assigns them unique names in the VHDL to avoid name clashes. However, clock and reset ports are never renamed, and you see an error if they do not have unique names. Avoid name clashes on other ports, to avoid renaming of the top-level ports in the VHDL.

All DSP Builder port names must comply with the following naming conventions:

- Remember VHDL is not case sensitive. For example, the input port `MyInput` and `MYINPUT` is the same VHDL entity.

- Avoid using VHDL keywords for DSP Builder port names.

- Do not use illegal characters. VHDL identifier names can contain only a - z, 0 - 9, and underscore (_) characters.

■ Begin all port names with a letter (a - z). VHDL does not allow identifiers to begin with non-alphabetic characters or end with an underscore.

■ Do not use two underscores in succession (__) in port names because it is illegal in VHDL.

☞ White spaces in the names for the blocks, components, and signals are converted to an underscore when DSP Builder converts the Simulink model file (**.mdl**) into VHDL.

MATLAB Variables

You can specify many block parameters (such as bit widths and pipeline depth) by entering a MATLAB base workspace or masked subsystem variable. You can then set these variables on the MATLAB command line or from a script. DSP Builder evaluates the variable and passes its value to the simulation model files. DSP Builder ensures that the parameters are in the required range.

☞ Although DSP Builder no longer restricts parameters to 51 bits, MATLAB evaluates parameter values to doubles, which restricts the possible values to 51-bit numbers expressible by a double.

👣 For information about which values are parameterizable, refer to the *DSP Builder Standard Blockset Libraries* section in volume 2 of the *DSP Builder Handbook* or to the block descriptions, which you can access with the **Help** command in the right button pop-up menu for each block.

Fixed-Point Notation

Figure 3–1 describes the fixed-point notation that I/O formats use in the DSP Builder block descriptions.

Table 3–1. Fixed-Point Notation

Description	Notation		Simulink-to-HDL Translation *(1), (2)*
Signed binary: fractional (SBF) representation; a fractional number	[L].[R] where:	[L] is the number of bits to the left of the binary point and the MSB is the sign bit [R] is the number of bits to the right of the binary point	A Simulink SBF signal A[L].[R] maps in VHDL to STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)
Signed binary; integer (INT)	[L] where:	[L] is the number of bits of the signed bus and the MSB is the sign bit	A Simulink signed binary signal A[L] maps to STD_LOGIC_VECTOR({L - 1} DOWNTO 0)
Unsigned binary; integer (UINT)	[L] where:	[L] is the number of bits of the unsigned bus	A Simulink unsigned binary signal A[L] maps to STD_LOGIC_VECTOR({L - 1} DOWNTO 0)

Table 3–1. Fixed-Point Notation

Description	Notation	Simulink-to-HDL Translation [1], [2]
Single bit integer (BIT)	[1] where: the single bit can have values 1 or 0	A Simulink single bit integer signal maps to STD_LOGIC

Notes to Table 3–1:

(1) STD_LOGIC_VECTOR and STD_LOGIC are VHDL signal types defined in the (**ieee.std_logic_1164.all** and **ieee.std_logic_signed.all** IEEE library packages).

(2) For designs in which unsigned integer signals are used in Simulink, DSP Builder translates the Simulink unsigned bus type with width w into a VHDL signed bus of width $w + 1$ where the MSB bit is set to 0.

Figure 3–1 graphically compares the signed binary fractional, signed binary, and unsigned binary number formats.

Figure 3–1. Number Format Comparison
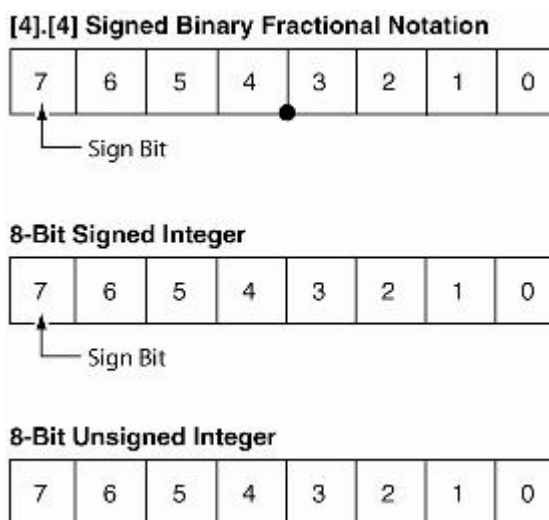
Binary Point Location in Signed Binary Fractional Format

For hardware implementation, you must cast Simulink signals into the desired hardware bus format. Therefore, convert floating-point values to fixed-point values.

This conversion is a critical step for hardware implementation because the number of bits required to represent a fixed-point value. The location of the binary point affects both the hardware resources and the system accuracy.

Choosing a large number of bits gives excellent accuracy—the fixed-point result is almost identical to the floating-point result—but consumes a large amount of hardware. You must design for the optimum size and accuracy trade-off. DSP Builder speeds up your design cycle by enabling simulation with fixed-point and floating-point signals in the same environment.

The `Input` block casts floating-point Simulink signals of type double into fixed-point signals. DSP Builder represents the fixed-point signals in the following signed binary fractional (SBF) format:

■ [*number of bits*].[]—represents the number of bits to the left of the binary point including the sign bit.

■ [].[*number of bits*]—represents the number of bits to the right of the binary point.

In VHDL, DSP Builder types the signals as STD_LOGIC_VECTOR.

For example, DSP Builder represents the 4-bit binary number 1101 as:

SimulinkThis signed integer is interpreted as –3

VHDLThis signed STD_LOGIC_VECTOR is interpreted as –3

If you change the location of the binary point to 11.01, that is, two bits on the left side of the binary point and two bits on the right side, DSP Builder represents the numbers as:

SimulinkThis signed fraction is interpreted as –0.75

VHDLThis signed STD_LOGIC_VECTOR is interpreted as –3

From a system-level analysis point of view, multiplying a number by –0.75 or –3 is very different, especially when looking at the bit width growth. In the first case, the multiplier output bus grows on the most significant bit (MSB), in the second case, the multiplier output bus grows on the least significant bit (LSB).

In both cases, the binary numbers are identical. However, the location of the binary point affects how a simulator formats the representation of the signal. For complex systems, you can adjust the binary point location to define the signal range and the area of interest.

For more information about number systems, refer to *AN 83: Binary Numbering Systems*.

Bit Width Design Rule

You must specify the bit width at the source of the datapath. DSP Builder propagates this bit width from the source to the destination through all intermediate blocks. Some intermediate DSP Builder blocks must have a bit width specified, while others have specific bit width growth rules which are described in the documentation for each block.

Some blocks which allow bit widths to be specified optionally, have an `Inferred` type setting that allows a growth rule to be used. For example, in the amplitude modulation tutorial design (Chapter 2, Getting Started) the `SinIn` and `SinDelay` blocks have a bit width of 16. Therefore, a bit width of 16 is automatically assigned to the intermediate `Delay` block.

Data Width Propagation

You can specify the bit width of many Altera blocks in the Simulink design. However, you do not need to specify the bit width for all blocks. If you do not specify explicitly the bit width, DSP Builder assigns a bit width during the Simulink-to-VHDL conversion by propagating the bit width from the source of a datapath to its destination.

Some intermediate DSP Builder blocks must have a specified bit width, while others have specific bit width growth rules that the documentation for each block describes. Some blocks, which allow bit widths to be specified optionally, allow use of a growth rule—the `Inferred` type setting.

Figure 3–2 illustrates bit-width propagation.

Figure 3–2. 3-Tap FIR Filter
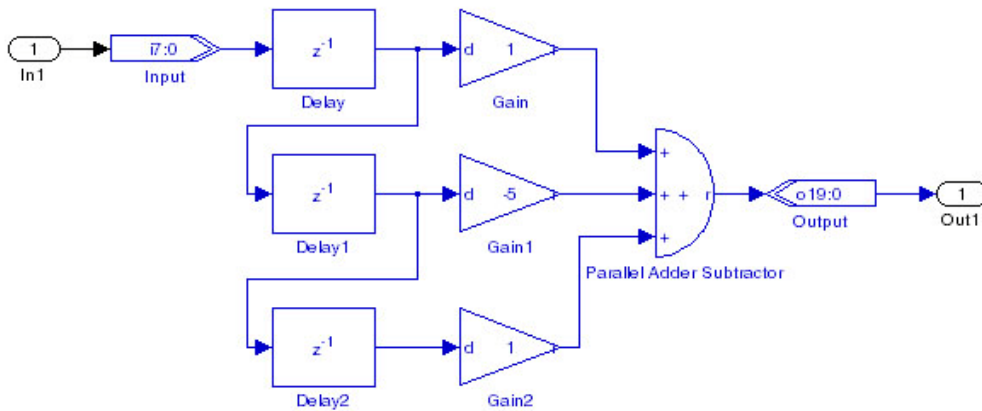
The **fir3tapsub.mdl** design is a 3-tap finite impulse response (FIR) filter and has the following attributes:

■ The input data signal is an 8-bit signed integer bus

■ The output data signal is a 20-bit signed integer bus

■ Three `Delay` blocks build the tapped delay line

■ The coefficient values are {1.0000, -5.0000, 1.0000}, a `Gain` block performs the coefficient multiplication

Figure 3–3 shows the RTL representation of **fir3tapsub.mdl** created by `Signal Compiler`.

Figure 3–3. 3-Tap FIR Filter in Quartus II RTL View

Tapped Delay Line

The bit width propagation mechanism starts at the source of the datapath, in this case at the `Input` block, which is an 8-bit input bus. This bus feeds the register `U0`, which feeds `U1`, which feeds `U2`. DSP Builder propagates the 8-bit bus in this register chain where each register is eight bits wide (Figure 3–4).

Figure 3–4. Tap Delay Line in Quartus II Version RTL Viewer

Arithmetic Operation

Figure 3–5 shows the arithmetic section of the filter, that computes the output `yout`:

$$yout[k] = \sum_{i=0}^{2} x[k-i]c[i]$$

where *c[i]* are the coefficients and *x[k - i]* are the data.

Figure 3–5. 3-Tap FIR Filter Arithmetic Operation in Quartus II Version RTL Viewer

This design requires three multipliers and one parallel adder. The arithmetic operations increase the bus width in the following ways:

■ Multiplying *a × b* in SBF format (where *l* is left and *r* is right) is equal to:

[*la*].[*ra*] × [*lb*].[*rb*]

The bus width of the resulting signal is:

([*la*] + [*lb*]).([*ra*] + [*rb*])

■ Adding *a* + *b* + *c* in SBF format (where *l* is left and *r* is right) is equal to:

[*la*].[*ra*] + [*lb*].[*rb*] + [*lc*].[*rc*]

The bus width of the resulting signal is:

(max([*la*], [*lb*], [*lc*]) + 2).(max([*ra*], [*rb*], [*rc*]))

The parallel adder has three input buses of 14, 16, and 14 bits. To perform this addition in binary, DSP Builder automatically sign extends the 14-bit busses to 16 bits. The output bit width of the parallel adder is 18 bits, which covers the full resolution.

The following options can change the internal bit width resolution and therefore change the size of the hardware required to perform the function that Simulink describes:

■ Change the bit width of the input data.

■ Change the bit width of the output data. The VHDL synthesis tool removes any unused logic.

■ Insert a `Bus Conversion` block to change the internal signal bit width.

Figure 3–6 shows how you can use `Bus Conversion` blocks to control internal bit widths.

Figure 3–6. 3-Tap Filter with BusConversion to Control Bit Widths
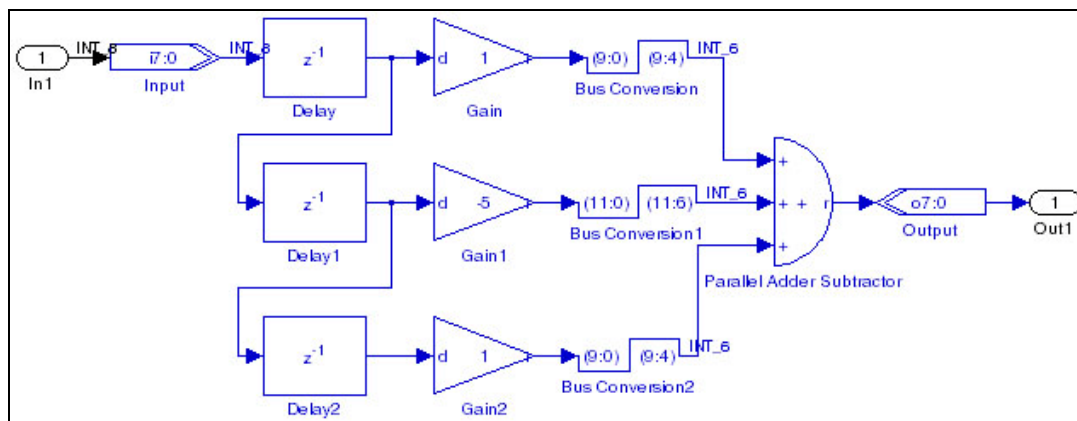
In Figure 3–6, the output of the `Gain` block has 4 bits removed. Port data type display is enabled in this example and shows that the inputs to the `Delay` blocks are of type `INT_8` but the outputs from the `Bus Conversion` blocks are of type `INT_6`.

☞ You can also achieve bus conversion by inserting an `AltBus`, `Round`, or `Saturate` block.

The RTL view illustrates the effect of this truncation. The parallel adder required has a smaller bit width and the synthesis tool reduces the size of the multiplier to have a 9-bit output (Figure 3–7).

Figure 3–7. 3-Tap Filter with BusConversion to Control Bit Widths in Quartus II RTL Viewer
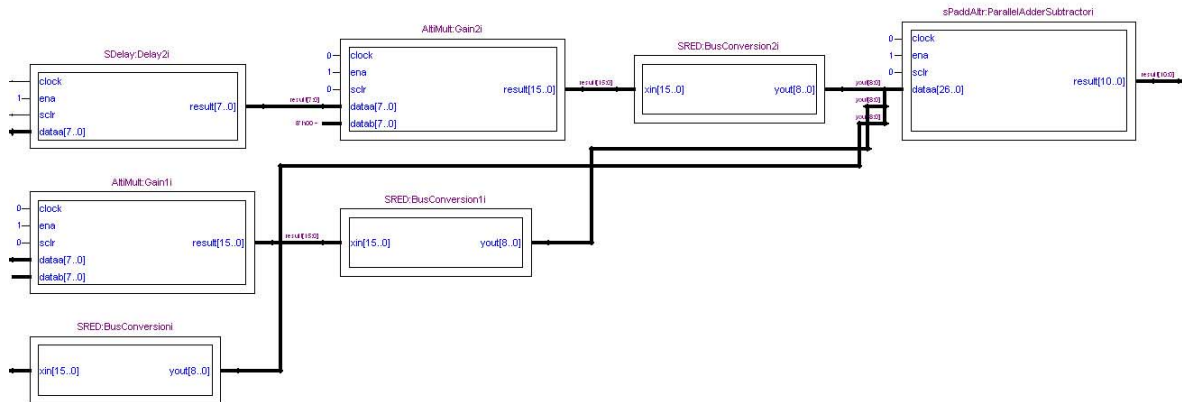
For more information, refer to "Fixed-Point Notation" on page 3–2.

Frequency Design Rules

This section describes the frequency design rules for single and multiple clock domains.

Single Clock Domain

If your design does not contain a `PLL` block or `Clock_Derived` block, DSP Builder uses synchronous design rules to convert a Simulink design into hardware. All DSP Builder registered blocks (such as the `Delay` block) operate on the positive edge of the single clock domain, which runs at the system sampling frequency.

The clock pin is not graphically displayed in Simulink unless you use the `Clock` block. However, when DSP Builder converts your design to VHDL it automatically connects the clock pin of the registered blocks (such as the `Delay` block) to the single clock domain of the system.

The default clock pin is named `clock` and there is also a default active-low reset pin named `aclr`.

By default, Simulink does not graphically display the clock enable and reset input pins of the DSP Builder registered blocks. When DSP Builder converts a design to VHDL, it automatically connects these pins. You can access and drive these optional ports by checking the appropriate option in the **Block Parameters** dialog box.

☞ Simulink issues a warning if you are using an inappropriate solver for your model. You should set the solver options to fixed-step discrete when you are using a single clock domain.

For Simulink simulation, all DSP Builder blocks (including registered DSP Builder blocks) use the sampling period specified in the `Clock` block. If there is no `Clock` block in your design, the DSP Builder blocks use a sampling frequency of 1. You can use the `Clock` block to change the Simulink sample period and the hardware clock period.

Multiple Clock Domains

A DSP Builder model can operate using multiple Simulink sampling periods. You can specify the clock domain in some DSP Builder block sources, such as the `Counter` block. You can also specify the clock domain in DSP Builder rate change blocks such as `Tsamp`.

When using multiple sampling periods, DSP Builder must associate each sampling period to a physical clock domain that can be available from an FPGA PLL or a clock input pin. Therefore, the top-level DSP Builder model must contain DSP Builder rate change blocks such as `PLL` or `Clock_Derived`.

You can use a `PLL` block to synthesize additional clock signals from a reference clock signal. These internal clock signals are multiples of the system clock frequency.

Refer to "Using the PLL Block" on page 3–14 for more information.

If your design contains the `PLL` block, `Clock` or `Clock_Derived` blocks, the DSP Builder registered blocks operate on the positive edge of one of the block's output clocks.

☞ You must set a variable-step discrete solver in Simulink when you are using multiple clock domains.

To ensure a proper hardware implementation of a DSP Builder design using multiple clock domains, consider the following points:

■ Do not use DSP Builder combinational blocks for rate transitions to ensure that the behavior of the DSP Builder Simulink model is identical to the generated RTL representation.

Figure 3–8 illustrates an incorrect use of the DSP Builder `Logical Bit Operator` (NOT) block.

Figure 3–8. Example of Incorrect Usage: Mixed Sampling Rate on a NOT Block
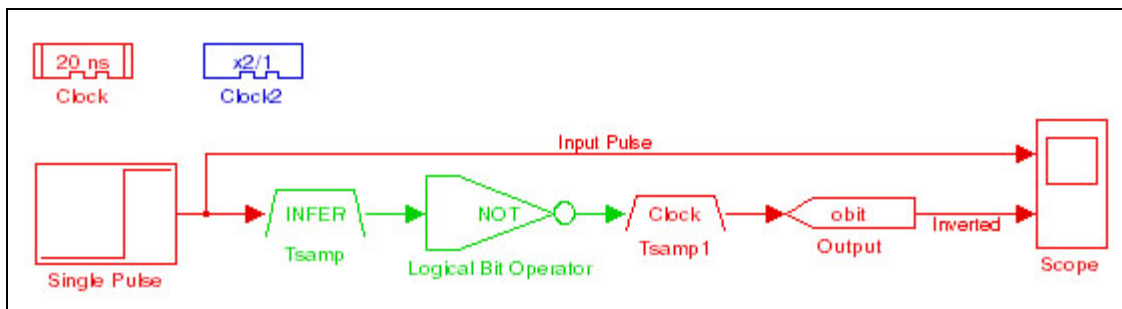
■ Two DSP Builder blocks can operate with two different sampling periods. However for most DSP Builder blocks, the sampling period of each input port and each output port must be identical.

Although this rule applies most of the DSP Builder blocks, there are some exceptions such as the `Dual-Clock FIFO` block where the sampling period of the read input port is expected to be different than the sampling period of the write input port.

■ For a datapath using mixed clock domains, the design may require additional register decoupling around the register that is between the domains.

This requirement is especially true when the source data rate is higher than the destination register, in other words, when the data of a register is toggling at the higher rate than the register's clock pin (Figure 3–9).

Figure 3–9. Data Toggling Faster than Clock
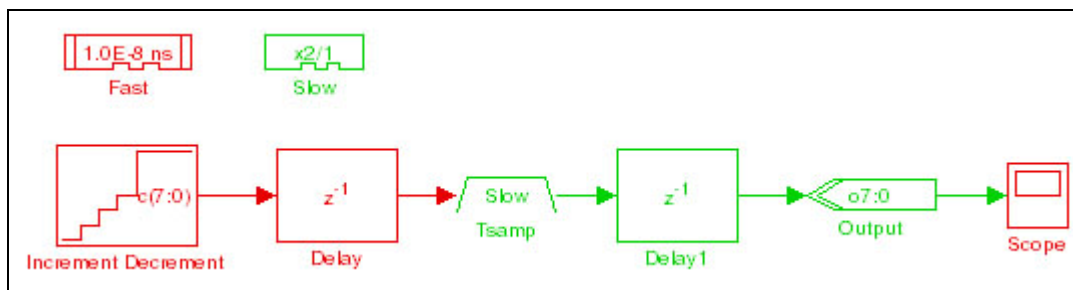
Figure 3–10 shows a stable hardware implementation.

Figure 3–10. Stable Hardware Implementation
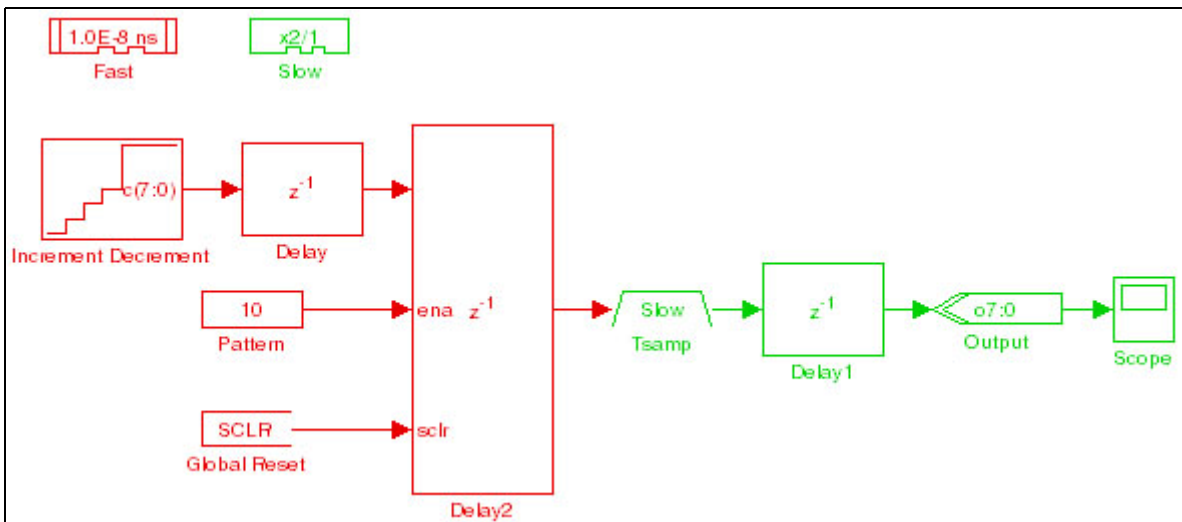
Using Clock and Clock_Derived Blocks

DSP Builder maps the `Clock` and `Clock_Derived` blocks to two hardware device input pins; one for the clock input, and one for the reset input for the clock domain. A design may contain zero or one `Clock` block and zero or more `Clock_Derived` blocks.

If you use `Clock_Derived` blocks, and there is only one system clock, you must generate an appropriate clock signal for connection to the hardware device input pins for the derived clocks.

The `Clock` block defines the base clock domain, and `Clock_Derived` blocks define other clock domains. DSP Builder specifies sample times in terms of the base clock sample time. If there is no `Clock` block, DSP Builder uses a default base clock, with a Simulink sample time of 1, and a hardware clock period of 20 μs.

This feature is available across all device families that DSP Builder supports. If no `Clock` block is present, the design uses a default clock pin named `clock` and a default active-low reset pin named `aclr`.

The `Signal Compiler` block assigns a clock buffer and a dedicated clock-tree to clock-signal input pin automatically to maintain minimum clock skew. If your design contains more `Clock` and `Clock_Derived` blocks than there are clock buffers available, non dedicated routing resources route the clock signals.

Clock Assignment

DSP Builder identifies registered DSP Builder blocks such as the `Delay` block and implicitly connects the clock, clock enable, and reset signals in the VHDL design for synthesis. When your design does not contain a `Clock` block, `Clock_Derived` block, or `PLL` block, all the registered DSP Builder block clock pins connect to a single clock domain (signal clock in VHDL).

Define clock domains by the clock source blocks: the `Clock` block, the `Clock_Derived` block and the `PLL` block.

The `Clock` block defines the base clock domain. You can specify its Simulink sample time and hardware clock period directly. If there is no `Clock` block, there is a default base clock with a Simulink sample time of 1. You can use the `Clock_Derived` block to define clock domains in terms of the base clock. DSP Builder specifies the sample time of a derived clock as a multiple and divisor of the base clock sample time.

The `PLL` block maps to a hardware PLL. You can use it to define multiple clock domains with sample times specified in terms of the PLL input clock. Use the PLL input clock either as the base clock or a derived clock.

Each clock domain has an associated reset pin. The `Clock` block and each of the `Clock_Derived` blocks have their own reset pin, the name of which is in the block's parameter dialog box. The clock domains of the `PLL` block share the reset pin of the `PLL` block's input clock.

When your design contains clock source blocks, DSP Builder implicitly connects the clock pins of all the registered blocks to the appropriate clock pin or PLL output. DSP Builder also connects the reset pins of the registered blocks to the top-level reset port for the block's clock domain.

DSP Builder blocks fall into the following clocking categories:

■ Combinational blocks—the output always changes at the same sample time slot as the input.

■ Registered blocks—the output changes after a variable number of sample time slots.

Figure 3–11 illustrates DSP Builder block combinational behavior.

Figure 3–11. Magnitude Block: Combinational Behavior
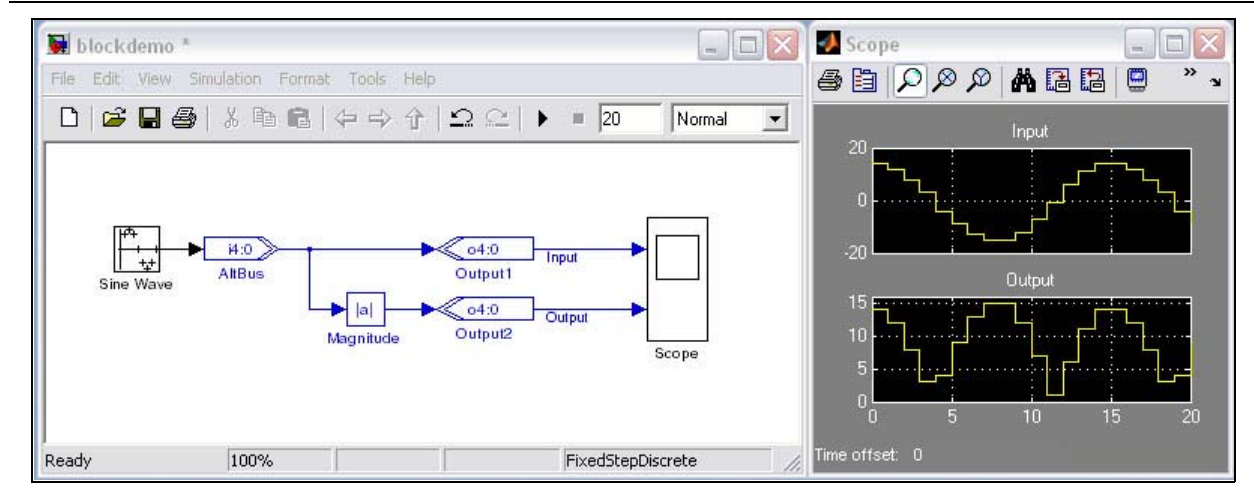
The Magnitude block translates as a combinational signal in VHDL. DSP Builder does not add clock pins to this function.

Figure 3–12 illustrates the behavior of a registered DSP block. In the VHDL netlist, DSP Builder adds clock pin inputs to this function. The Delay block, with the **Clock Phase Selection** parameter equal to 100, is converted into a VHDL shift register with a decimation of three and an initial value of zero.

Figure 3–12. Delay Block: Registered Behavior
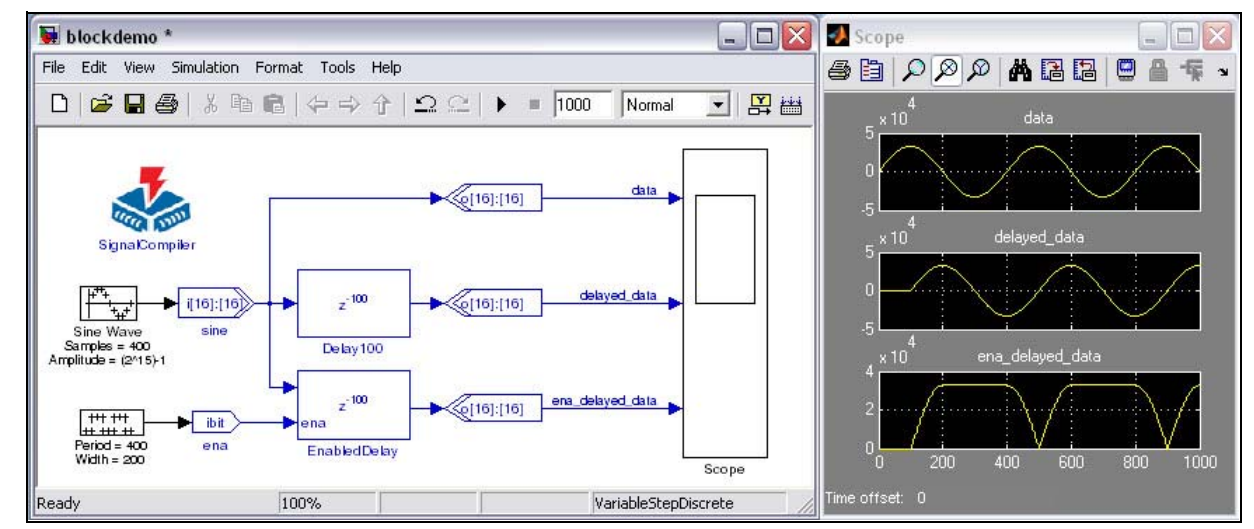
For feedback circuitry (the output of a block fed back into the input of a block), a registered block must be in the feedback loop. Otherwise, DSP Builder creates an unresolved combinational loop (Figure 3–13).

Figure 3–13. Feedback Loop
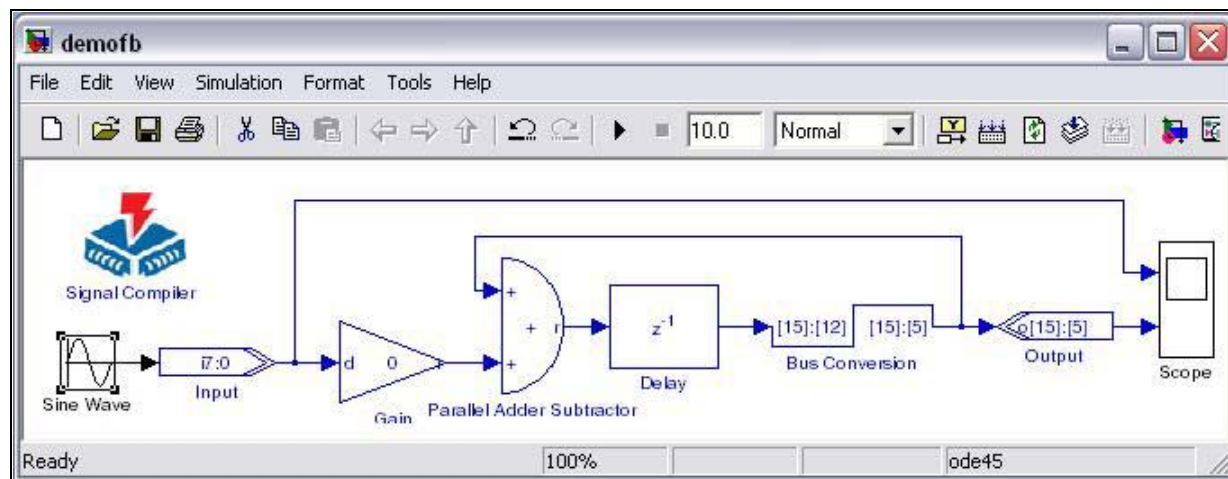
Use the PLL block and assign different sampling periods on registered DSP Builder blocks to design multirate designs.

Alternatively, use a single clock domain with clock enable and the following design rules to design multirate designs without the DSP Builder PLL block:

■ The fastest sample rate is an integer multiple of the slower sample rates. The **Clock Phase Selection** field in the **Block Parameters** dialog box specifies the values for the Delay block.

■ The **Clock Phase Selection** box accepts a binary pattern string to describe the clock phase selection. DSP Builder processes each digit or bit of this string sequentially on every cycle of the fastest clock. When a bit is equal to one, DSP Builder enables the block; when a bit is equal to zero, DSP Builder disables the block.

Table 3–2 shows some examples of typical clock phase selections.

Table 3–2. Clock Phase Selection Example

Phase	Description
1	The Delay block is always enabled and captures all data passing through the block (sampled at the rate 1).
10	The Delay block is enabled every other phase and every other data (sampled at the rate 1) passes through.
0100	The Delay block is enabled on the 2nd phase out of 4 and only the 2nd data out of 4 (sampled at the rate 1) passes through. The data on phases 1, 3, and 4 does not pass through the Delay block.

Figure 3–14 compares the scopes for the Delay block operating at a one quarter rate on the 1000 and 0100 phases, respectively.

Figure 3–14. 1000 as Opposed to 0100 Phase Delay
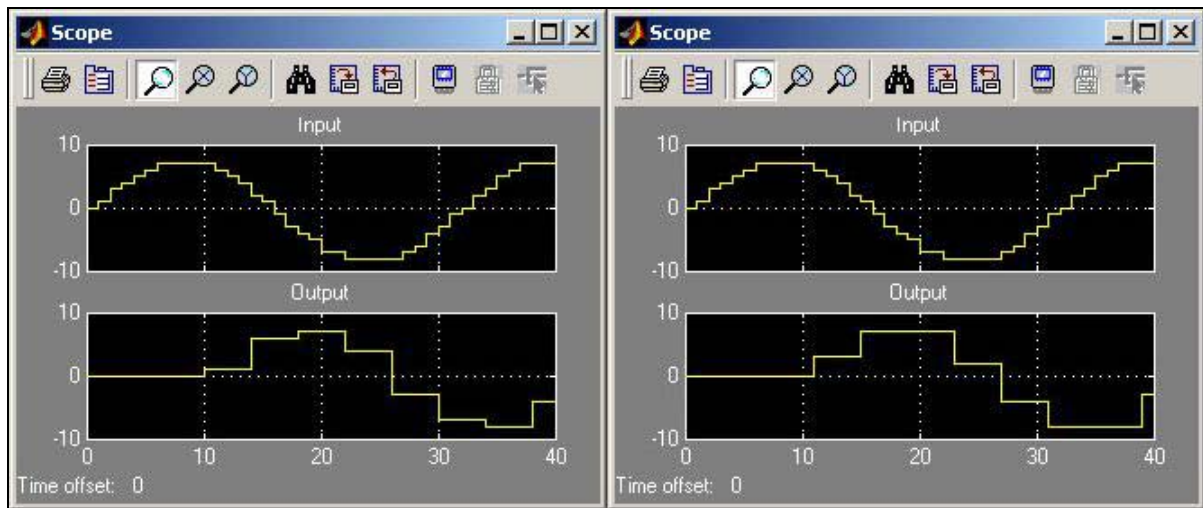
Using the PLL Block

DSP Builder maps the PLL block to the hardware device PLL. The number of PLL internal clock outputs that each device family supports depends on the specific device packaging.

For information about the built-in PLLs, refer to the device handbook for the device family you are targeting.

Figure 3–15 shows an example of multiple-clock domain support using the PLL block.

Figure 3–15. MultipleClockDelay.mdl
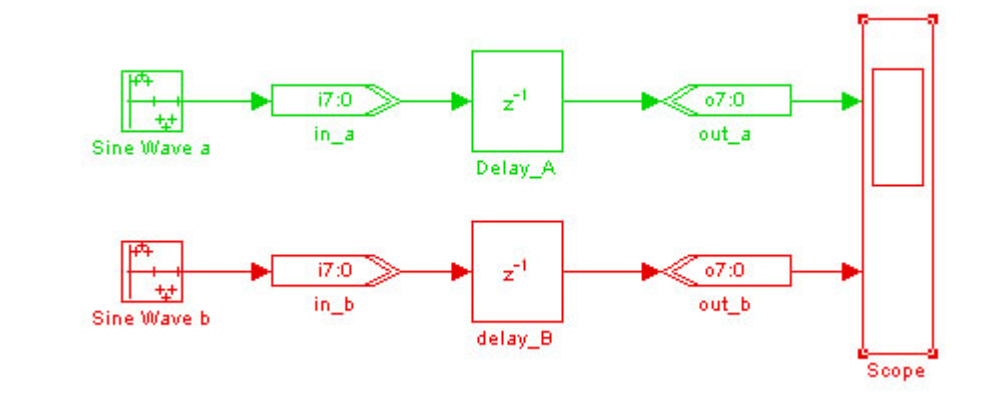
Figure 3–16 shows the clock setting configuration for the PLL block in the design example **MultipleClockDelay.mdl**. Output clock PLL_clk0 is set to 800 ns, and output clock PLL_clk1 is set to 100 ns.

Figure 3–16. PLL Setting
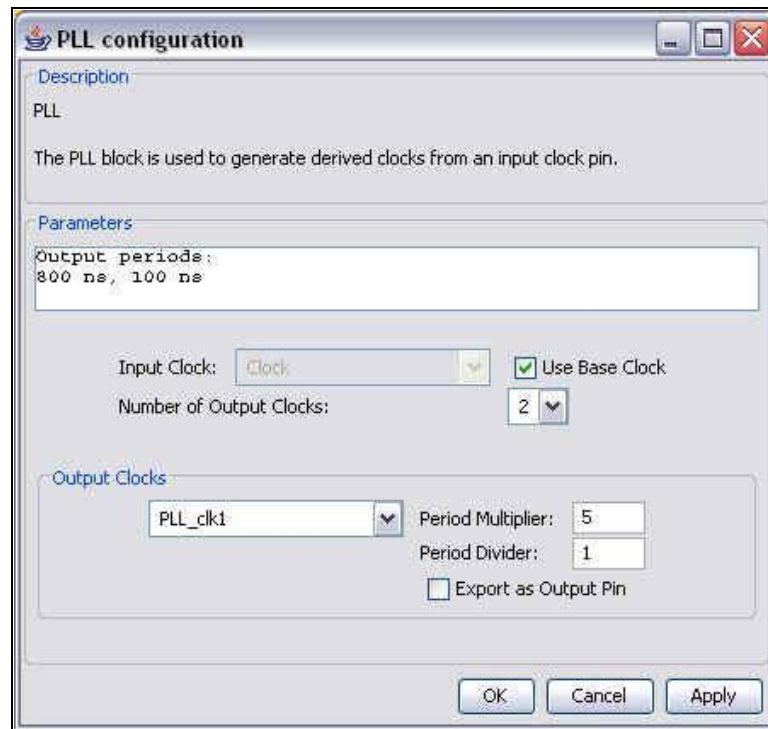
Datapath A (green in Figure 3–15) operates on output clock PLL_clk0 and datapath B (red in Figure 3–15) operates on output clock PLL_clk1. Specify these clocks by setting the **Specify Clock** option and enter the clock name in the **Block Parameter** dialog box for each input block.

In this design, the **Sample time** parameters for the Sine Wave a block and Sine Wave b block are set explicitly to 1e-006 and 1e-007, so that DSP Builder provides data to the input blocks at the rate at which they sample.

Using Advanced PLL Features

The DSP Builder PLL block supports the fundamental multiplication and division factor for the PLL. If you want to use other PLL features (such as phase shift, duty cycle), use a separate Quartus Prime project with the following method:

■ Create a new Quartus Prime project and use the MegaWizard™ Plug-In to configure the ALTPLL block.

■ Add the DSP Builder .**mdl** file to the Quartus Prime project as a source file.

■ Create a top-level design that instantiates your ALTPLL variation and your DSP Builder design.

About Timing Semantics Between Simulink and HDL Simulation

DSP Builder uses Simulink to simulate the behavior of hardware components. However, there are some fundamental differences between the step-based simulation in Simulink and the event-driven simulation that VHDL and Verilog HDL designs use.

This section describes the timing semantics that DSP Builder uses for translating between the Simulink and HDL environments.

Simulink Simulation Model

To ensure correlation between the HDL and Simulink simulation, you must use a discrete fixed or variable-step solver in Simulink.

☞ Use a fixed-step solver for a single clock domain design or a variable-step solver for multiple-clock domain designs.

Configure the solver timing mode in the **Configuration Parameters** dialog box from the Simulation menu in Simulink. Each step is a discrete unit of simulation. DSP Builder quantizes the clock in an idealized manner as a cycle counter.

At the beginning of each step, Simulink provides each block with inputs that you know. DSP Builder evaluates functions and propagates the resultant outputs in the current step. The outputs of your model are the results of all these computations.

For all steps, Simulink blocks produce output signals. Outputs varying based on inputs received in the same step are referred to as direct feedthrough. Some DSP Builder blocks may include direct feedthrough outputs, depending on the parameterization of each block.

HDL Simulation Models

DSP Builder drives hardware simulation with a clock signal and the available input stimuli. The TestBench block's testbench script feeds input signals to the HDL simulator that maintain correlation between the HDL and Simulink simulation.

Simulation models in the DSP Builder libraries evaluate their logic on positive clock edges. To avoid any timing conflicts, external inputs transition on negative clock edges. DSP Builder updates registered outputs on positive clock edges. The TestBench block-generated inputs arrive on negative clock edges, causing an apparent half-cycle delay in the arrival of output (Figure 3–17 on page 3–18).

☞ The HDL simulation in ModelSim should run over the same time as the Simulink simulation. Generally DSP Builder aligns the timing so that ModelSim simulation finishes at the end of the stimulus data. However, occasionally when using multiple clocks, the rounding calculation that aligns the clock signals may set ModelSim simulation to run for one additional clock cycle (on the fastest clock). You may receive an unexpected end of file error message because there is no stimulus data for this extra cycle.

Startup and Initial Conditions

The testbench includes a global reset for each clock domain. All blocks (except the `HDL Import` and MegaCore function blocks) automatically connect any reset on the hardware to the global asynchronous reset for the clock domain.

When a block explicitly declares an asynchronous reset, this reset is `OR`ed with the global reset.

A `Global Reset` block (`SCLR`), which corresponds to this hardware signal is in the Altera DSP Builder Blockset IO & Bus library.

The global reset signal is reset before meaningful simulation. When converting from the Simulink domain to the hardware domain, the reset period is before the Simulink simulation begins. Therefore, in Simulink simulation, the `Global Reset` block outputs only a constant zero and has no simulation behavior. Connect the hardware to reset, and thus reset at the start of a ModelSim testbench simulation.

☞ DSP blocks or MegaCore functions may have additional initial conditions or startup states that are not automatically reset by the global reset signal.

Initial Reset of HDL Import Blocks and MegaCore Functions in Simulink Simulations

The ModelSim testbenches have an initial reset cycle, which ModelSim performs, before simulation. The first 200 cycles are reset, then the testbench puts the test vectors through. The reset sets the intial state of registers, which may otherwise have 'X' (unknown) outputs. In Simulink simulations, there is no explicit reset signal—the Simulink simulation models for DSP Builder blocks assume there is a reset. HDL import blocks and MegaCore functions do not provide explicit models, but use a generic HDL simulator. Simulink does not have a way to represent 'X' in its numeric types— it writes an unknown 'X' as a 0. The HDL import block or MegaCore function may have registers that require a reset to avoid unknown outputs. Unknown states may be initially propagating through your imported HDL import block or MegaCore function. For some imported HDL import blocks or MegaCore functions, these initial unknown outputs may result in outputs that are different to the ModelSim simulation (which is reset).

Altera recommends that you must first explicitly reset HDL import blocks and MegaCore functions in Simulink simulation. If you have any such registers with unknown outputs in a feedback loop, the Simulink simulation always gives 'X' (zero in Simulink's numeric types) until reset and the unknown states continue to propagate.

☞ If a block in one clock domain drives a block in another clock domain with an asynchronous clear port, Simulink may not model the system. An asynchronous clear only takes full effect if you assert it at the end of a sample; if it is asserted then cleared, DSP Builder ignores it.

DSP Builder Global Reset Circuitry

By default, Simulink does not graphically display the clock enable and reset input pins on DSP Builder registered blocks. When DSP Builder converts a design to HDL, it automatically connects the implied clock enable and reset pins.

If you turn on the optional ports in the **Block Parameters** dialog box for each of the DSP Builder registered blocks, you can access and drive the clock enable and reset input pins graphically in the Simulink software.

In the HDL domain, the registered DSP Builder blocks uses an asynchronous reset, as this behavioral VHDL code example shows:

```
process(CLOCK, RESET)
begin
   if RESET = '1' then
      dout <= (others => '0');
   else if CLOCK'event and CLOCK = '1' then
      dout <= din;
   end if;
end
```

In addition, when targeting a development board, the **Block Parameters** dialog box for the DSP Board configuration block typically includes a **Global Reset Pin** selection box where you can choose from a list of pins that correspond to the DIP and push-button switches.

The reset logic polarity can be either active-high or active-low. When you select active-low, the value of the reset signal in Simulink simulation is still 0 for inactive and 1 for active. However, DSP Builder inserts a NOT gate on the input pin in the generated hardware. The value of the reset signal in simulation is therefore the value as it exists across the internal design, rather then the value at the input pin.

Quartus® II synthesis interprets this reset as an asynchronous reset, and uses an input of the logic element look-up table to instantiate the function. The HDL simulates correctly in this case because the testbench produces the reset input as required.

Reference Timing Diagram

Figure 3–17 shows the timing relationships in a hypothetical case where a register is fed by the output of a counter. The counter output begins at 10—the value is 10 during the first Simulink clock step.

Figure 3–17. Single-Clock Timing Relationships
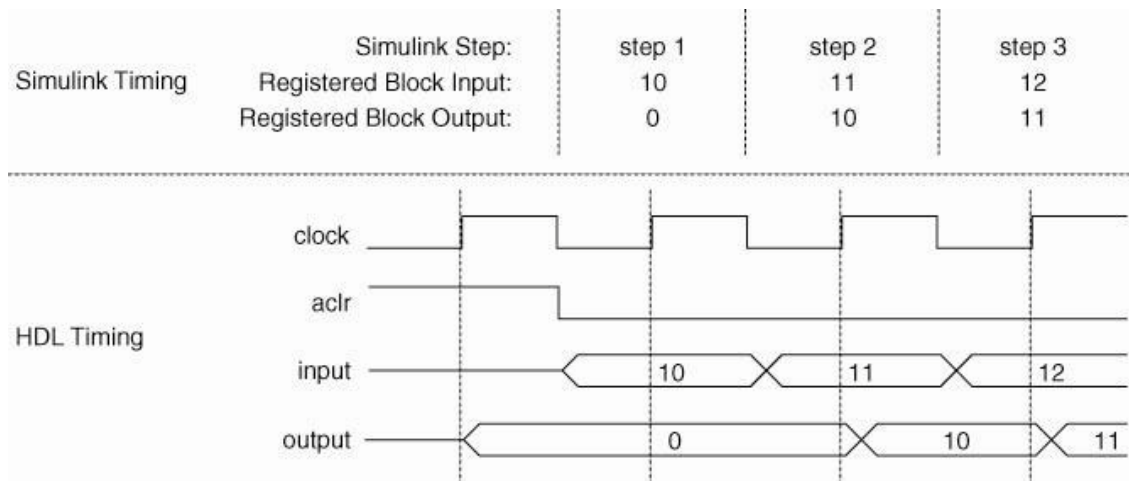
This timing is not true when crossing clock domains. For example, Figure 3–18 shows the timing delays in a design with a derived clock that has half the base clock period. In general, DSP Builder is not cycle-accurate when crossing clock domains.

Figure 3–18. Multiple-Clock Timing Relationships
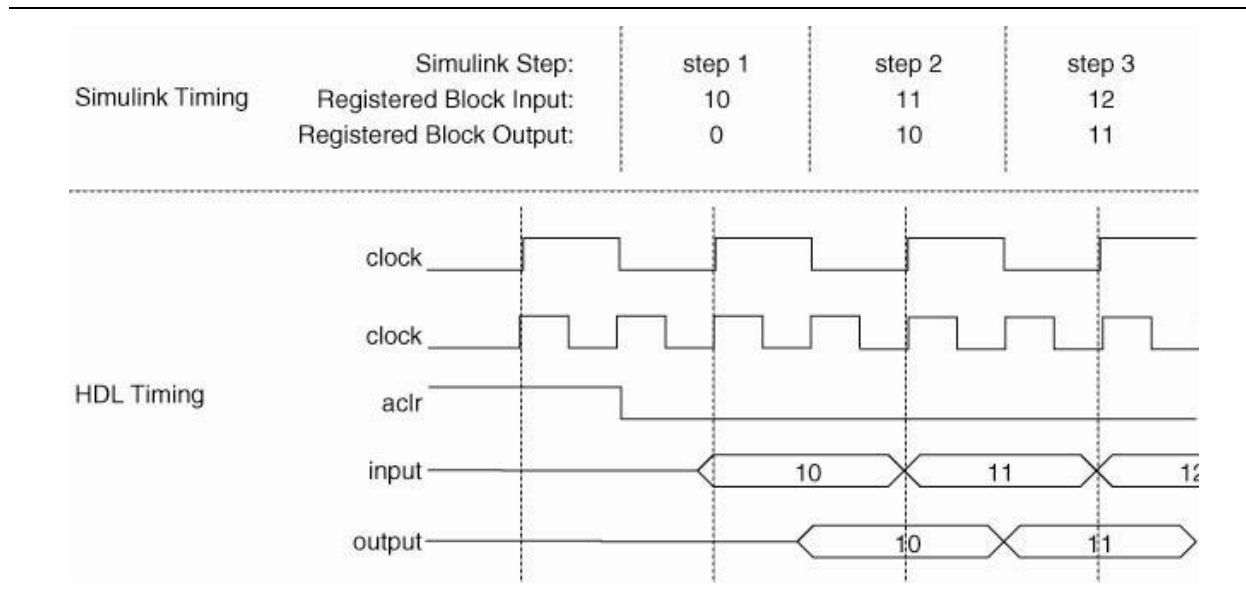
Signal Compiler and TestBench Blocks

The `Signal Compiler` block uses Quartus Prime synthesis to convert a Simulink design into synthesizable VHDL including generation of a VHDL testbench and other supporting files for simulation and synthesis.

`Signal Compiler` assumes that your design complies with the Simulink rules and that any variables and inherited variables propagate through the whole design.

You should always run a simulation in Simulink before running `Signal Compiler`. The simulation updates all variables in your design (including workspace variables and inherited parameters), sets up certain blocks (such as the memory blocks, and inputs from and outputs to workspace blocks), and also traps any design errors that do not comply with Simulink rules.

The `Input` and `Output` blocks map to input and output ports in VHDL and mark the edge of the generated system. Typically, you connect these blocks to the Simulink simulation blocks for your testbench. An `Output` block should not connect to another Altera block. If you connect more Altera blocks (that map to HDL), empty ports are created and the HDL does not compile for synthesis.

For more information about the `Input` and `Output` blocks, refer to the *IO & Bus Library* chapter of the *DSP Builder Reference Manual*.

Design Flows for Synthesis, Compilation and Simulation

You can use the `Signal Compiler` and `Testbench` blocks to control your design flow for synthesis, compilation, and simulation. DSP Builder supports the following flows:

■ Automatic flow—allows you to control the entire design process in the MATLAB or Simulink environment with the `Signal Compiler` block. With this flow, your design compiles inside a temporary Quartus Prime project. The results of the synthesis and compilation display in the Signal Compiler Messages box. You can also use the automatic flow to download your design into supported development boards.

■ Manual flow—you can also add the **.mdl** file to an existing Quartus Prime project using the *<model name>*_**add.tcl** script. This script is generated whenever the `Signal Compiler` or `TestBench` block is run. You can use the script to add the **.mdl** file and any imported HDL to your project. You can then instantiate your design in HDL.

■ Simulation flow—if the ModelSim executable (**vsim.exe**) is on your path, you can use the `TestBench` block to compile your design for ModelSim simulation. You can then automatically compare the Simulink and ModelSim simulation results.

For an example that uses the `Signal Compiler` blocker, refer to page 2–14 of the "Getting Started".

For information about the parameters for the `Signal Compiler` and `TestBench` blocks, refer to the *AltLab Library* chapter of the *DSP Builder Reference Manual*.

DSP Builder supports the Simulink Bus Creator, Bus Selector, and Bus Assignment blocks but you must only use them for routing.

Hierarchical Design

DSP Builder supports hierarchical design using the Simulink `Subsystem` block.

DSP Builder preserves the hierarchy structure in a VHDL design and each hierarchical level in a Simulink model file (**.mdl**) translates into one VHDL file.

For example, Figure 3–19 illustrates a hierarchy for a design **fir3tap.mdl**, which implements two FIR filters.

Figure 3–19. Hierarchical Design Example
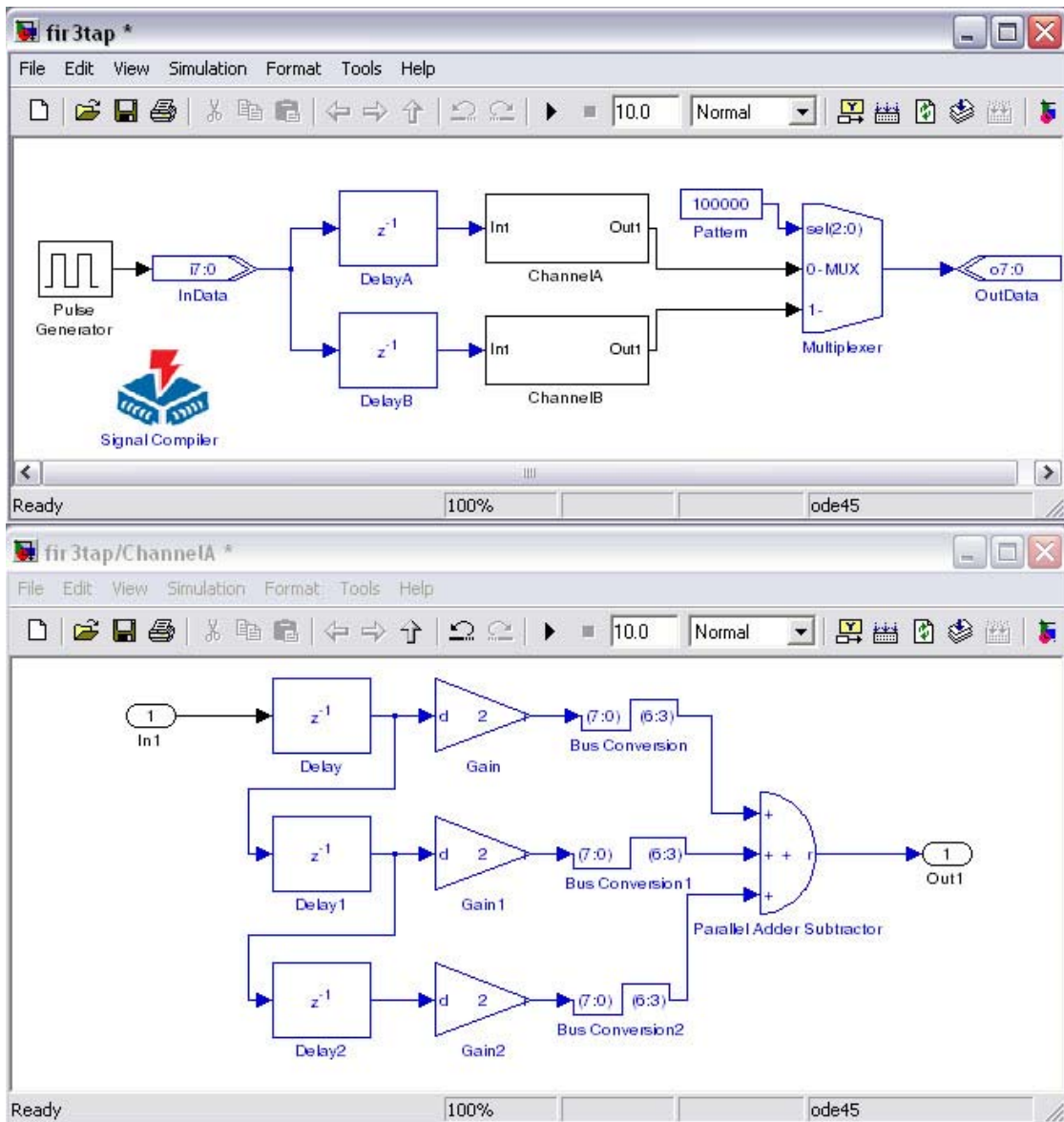
For information about naming the Subsystem block instances, refer to "DSP Builder Naming Conventions" on page 3–1.

Goto and From Block Support

DSP Builder supports the Goto and From blocks from the **Signal Routing** folder in the generic Simulink library.

You can use these blocks for large fan-out signals and to enhance the diagram clarity.

Figure 3–20 shows an example of the `Goto` and `From` blocks.

Figure 3–20. Goto & From Block Example
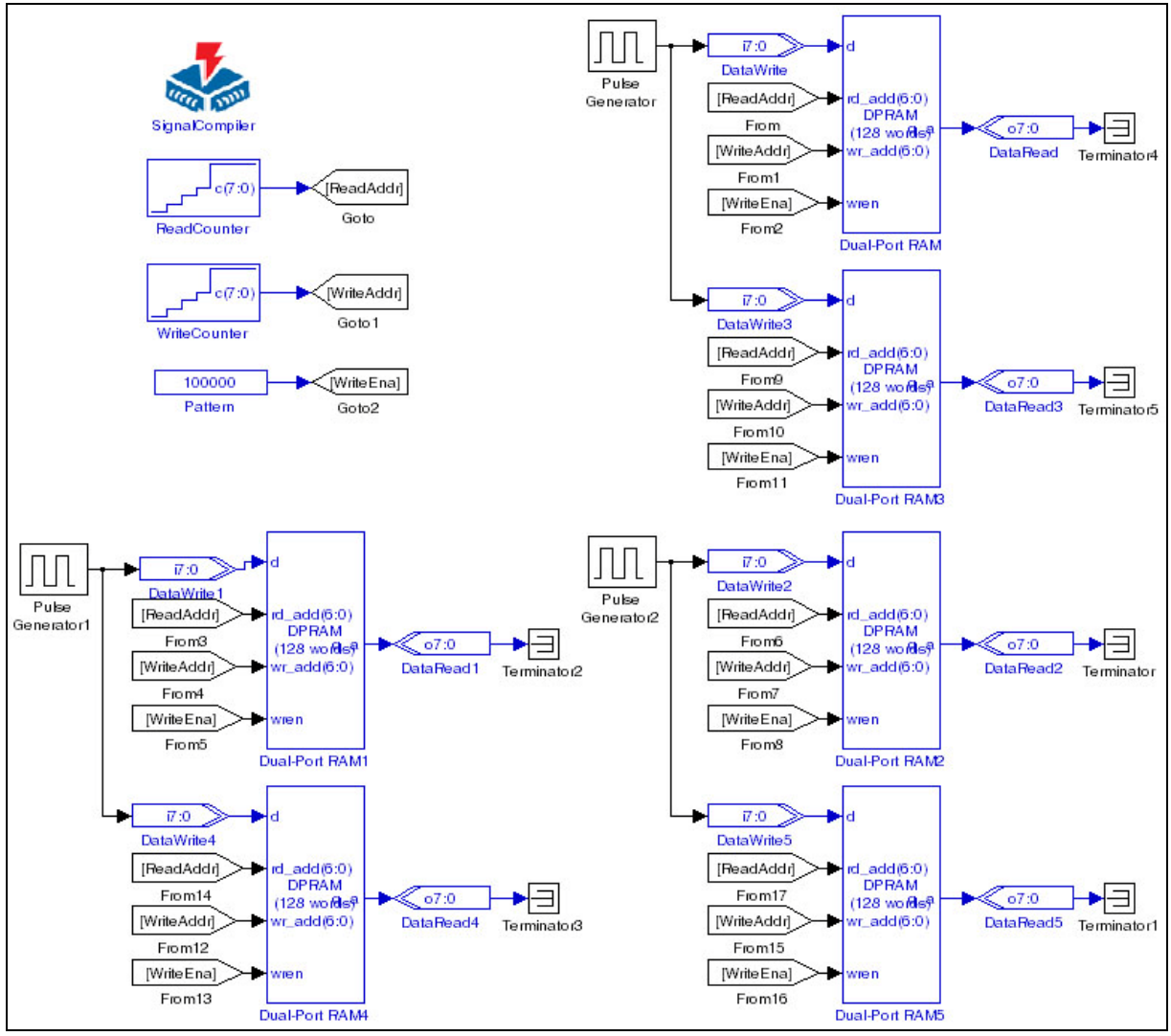
Use the Goto blocks ([ReadAddr], [WriteAddr], and [WriteEna] with the `From` blocks ([ReadAddr], [WriteAddr], and [WriteEna]), which connect to the dual-port RAM blocks.

Black Box and HDL Import

You can add your own VHDL or Verilog HDL code to your design and specify which subsystem block(s) DSP Builder should translate into VHDL. You can implement this process—creating a black box—implicitly or explicitly.

An explicit black box uses the `HDL Input`, `HDL Output`, `HDL Entity`, and `Subsystem Builder` blocks. For information about using these blocks to create an explicit black box, refer to "Subsystem Builder Design Example" in Chapter 8.

An implicit black box uses the `HDL Import` block to instantiate the black-box subsystem. For information about creating an implicit black box with your own HDL code, refer to the "HDL Import Design Example" in Chapter 8.

Using a MATLAB Array or .hex File to Initialize a Block

Use a MATLAB array to specify the values entered in the `LUT` block or to initialize the `Dual-Port RAM`, `Single-Port RAM`, `True Dual-Port RAM`, or `ROM` blocks. You can also use an Intel format hexadecimal format (.**hex**) file to initialize a `RAM` or `ROM` block.

If the MATLAB array data values or the values in the **.hex** file do not represent exactly in the selected data type, DSP Builder rounds them and issues a warning. DSP Builder rounds the values by expressing the number in binary format, then truncates to the specified width, which results in rounding towards minus infinity.

For example, if the input value is −0.25 (minimally expressed in signed binary fractional two's compliment format as `111`) and the selected target data format is signed fractional `[1].[1]`, DSP Builder truncates the value to `11` = −0.5. DSP Builder rounds the value towards minus infinity to the nearest representable number.

Similarly, if you select unsigned integer data type and the value is 1.9, DSP Builder rounds this value down to 1.

Comparison Utility

DSP Builder provides a simple utility that runs simulation comparison between Simulink and ModelSim from the command line:

```
alt_dspbuilder_verifymodel('modelname.mdl', 'logfile.txt')⏎
```

A testbench GUI displays messages as DSP Builder performs the comparison. The command returns true (1) or false (0) according to whether the simulation results match and the output is recorded in the specified log file.

For more information about running a comparison between Simulink and ModelSim, refer to "Performing RTL Simulation" in Chapter 2.

Adding Comments to Blocks

Add comments to any DSP Builder block by right-clicking on the block to display the **Block Properties** dialog box and entering text in the **Description** field of the dialog box (Figure 3–21 on page 3–24).

Figure 3–21. Adding Comments to a Block
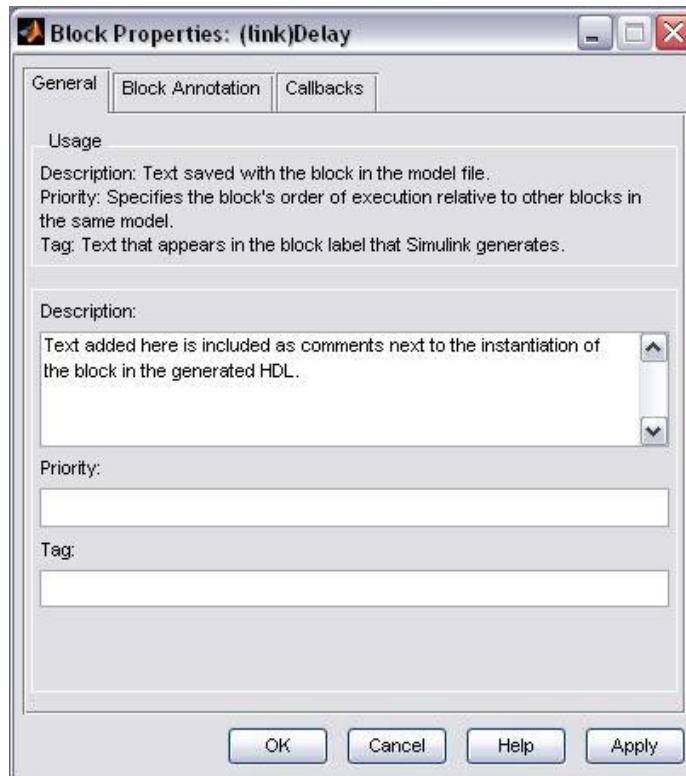
DSP Builder includes the comment text next to the instantiation of the block in the generated HDL.

Adding Quartus II Constraints

1. Set Quartus Prime global project assignments in your Simulink model by adding `Quartus Prime Global Project Assignment` blocks from the AltLab library. Each block sets a single global assignment but you can use multiple blocks for multiple assignments.

2. Use these assignments to set Quartus Prime compilation directives, such as target device or timing requirements.

For a description of the `Quartus Prime Global Project Assignment` block, refer to the *DSP Builder Reference Manual*.

3. Add additional Quartus Prime assignments or constraints that are not supported in DSP Builder by creating a Tcl script in your design directory. Any file named **<*model name*>_add_user.tcl** is automatically sourced when you run Signal Compiler.

The Tcl file can include any number of Quartus Prime assignments with the syntax:

```
set_global_assignment -name <assignment> <value>
```

For detailed information about Quartus Prime assignments, refer to the *Quartus Prime Settings File Reference Manual*.

Displaying Port Data Types

1. Optionally show the Simulink and DSP Builder port data types for each of the signals in your Simulink model by turning on **Port Data Types** in the **Port/Signal Displays** section of the Simulink Format menu.

When you set this option, the DSP Builder internal signal type (*SBF_L_R*, INT_L, *UINT_L*, or BIT where L, and R are the number of bit to the left and right of the binary point) displays. For example, SBF_8_4 for a 12-bit signed binary fractional data type with 4 fractional bits, or UINT_16 for a 16-bit unsigned integer.

Figure 3–22 shows the amplitude modulation example with port data type display enabled.

Figure 3–22. Tutorial Example Showing Port Data Types and Pipeline Depth
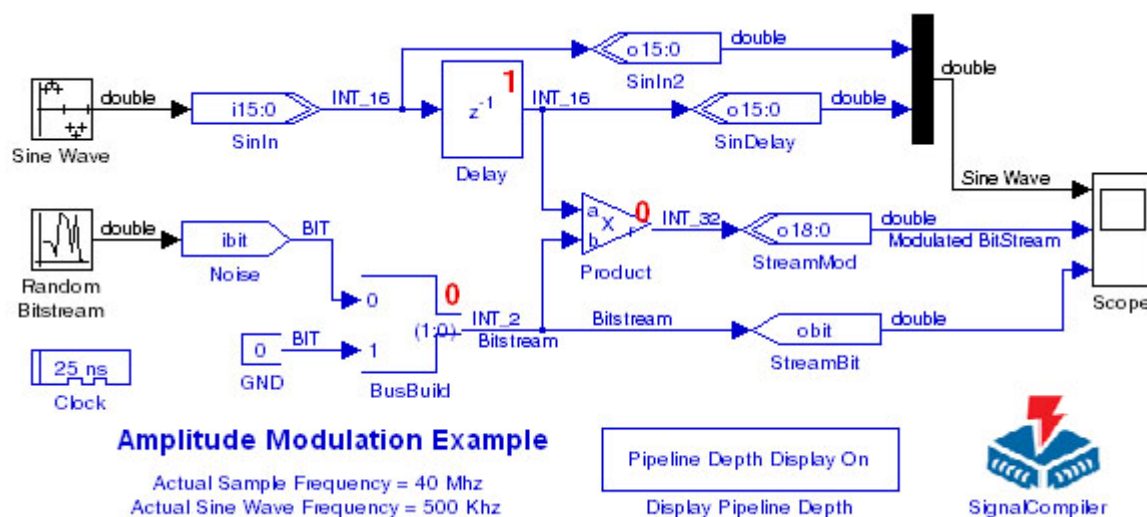
For more information about the DSP Builder internal signal types, refer to "Fixed-Point Notation" on page 3–2.

Displaying the Pipeline Depth

1. Optionally show the pipeline depth on the primitive blocks (such as the Arithmetic library blocks) in your Simulink model by adding a Display Pipeline Depth block from the AltLab library.

You can change the display mode by double-clicking on the block. When set, the current pipeline depth displays at the top right corner of each block that adds latency to your design (Figure 3–22). The selected mode shows on the `Display Pipeline Depth` block symbol.

Updating HDL Import Blocks

The `HDL Import` blocks in your design may need updating if you upgrade from a previous software version or move a design to a different workstation. Use the `alt_dspbuilder_refresh_hdlimport` command to update these blocks. This command checks that the referenced HDL files (or Quartus Prime project) exists. If it finds the references, the **HDL Import** dialog box opens and a compilation is automatically invokes to regenerate the Simulink model. If it finds neither, but there is an existing simulation netlist, it uses this netlist for simulation.

To run the command, follow these steps:

1. Start the MATLAB or Simulink software.

2. Open a Simulink model that contains imported HDL.

3. Run the command by typing the following at the MATLAB prompt:

    `alt_dspbuilder_refresh_hdlimport` ↵

You can optionally select a `HDL Import` block to run the command on only the selected subsystem.

Analyzing the Hardware Resource Usage

To analyze the hardware resources required for your design with a `Resource Usage` block, follow these steps:

1. Select the **AltLab** library from the **Altera DSP Builder BlockSet** folder in the Simulink Library Browser.

2. Drag and drop a `Resource Usage` block into your model and double-click on the block to open the **Resource Usage** dialog box.

3. Double-click on the `Signal Compiler` block and click **Compile** to recompile your design in the Quartus Prime software.

The `Resource Usage` block updates to show a summary of the estimated logic, RAM and DSP block usage (Figure 3–23).

Figure 3–23. Resource Usage Block
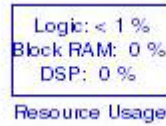
The **Resource Usage** dialog box updates to show a detailed report of the resources that each of the blocks require in your model that generate hardware.

For example, Figure 3–24 shows the hardware resources that the `Product` block requires in the amplitude modulation example.

Figure 3–24. Resource Usage Dialog Box

The information depends on the selected device family. Refer to the device documentation for more information.

You can also click the **Timing** tab and click **Highlight path** to highlight the critical paths on your design.

☞ When the source and destination in the dialog box are the same and a single block is highlights, the critical path is due to the internal function or a feedback loop. For a more complex example, the entire critical path through your design may highlight.

Loading Additional ModelSim Commands

When you import HDL as a black box, DSP Builder creates a subdirectory **DSPBuilder<**model name**>_import**. Any Tcl script ***_add_msim.tcl** in this subdirectory automatically sources when you launch ModelSim.

1. You should not modify the generated scripts, but create you own scripts such as **<**user name**>_add_msim.tcl**, which contain additional ModelSim commands that you want to load into ModelSim.

About Quartus II Assignments and Block Entity Names

The VHDL entity names of the blocks in a DSP Builder design are dependent on the block's parameter values. Blocks of the same type and same parameterization share a common VHDL entity.

The entity names have the following format:

```
<block type name>_GN<8 alphanumeric characters>
```

For example, a `Delay` block entity name:

```
alt_dspbuilder_delay_GNLVAGVO3B
```

Changing the parameterization of the block causes the entity name to change. If you want to make an assignment to a block in the Quartus Prime project, and for the assignment to remain when the block parameters change, you can use regular expressions in the assignments.

For example, you may want to make a **Preserve Registers** assignment to the `Delay` blocks in Figure 3–25 to prevent them from merging.

Figure 3–25. Entity Name Assignment Example
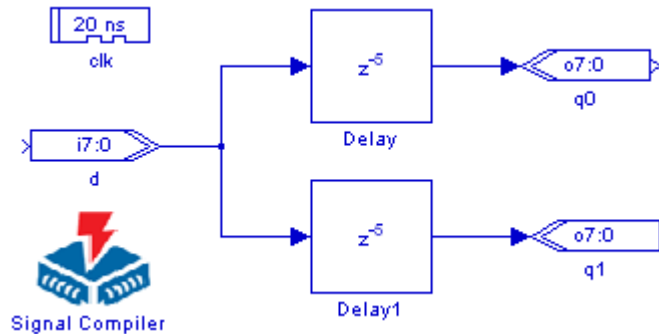
Using the Quartus Prime Assignment Editor and Node Finder tools, you can identify the names of the registers and make the assignments to them. For example, if your model is `my_model`, the names may be:

```
my_model_GN:auto_inst|alt_dspbuilder_delay_GNLVAGVO3B:Delay|alt_dsp
builder_SDelay:Delay1i|DelayLine
```

```
my_model_GN:auto_inst|alt_dspbuilder_delay_GNLVAGVO3B:Delay1|alt_ds
pbuilder_SDelay:Delay1i|DelayLine
```

These assignments prevent merging of the registers. If you change the length of the delay, the assignments are no longer valid. However, you can edit the **To** field of the assignment and use a regular expression that is still valid if the entity name changes due to a parameter change: Replace the eight alphanumeric characters following the `GN` in the block entity name with `.{8}`, which is a regular expression that matches any eight characters. The targets of the assignments then become:

```
my_model_GN:auto_inst|alt_dspbuilder_delay_GN.{8}:Delay|alt_dspbuil
der_SDelay:Delay1i|DelayLine
```

```
my_model_GN:auto_inst|alt_dspbuilder_delay_GN.{8}:Delay1|alt_dspbui
lder_SDelay:Delay1i|DelayLine
```

If you want the assignment to apply to the whole block, not just the specific nodes, you can use the following code:

```
my_model_GN:auto_inst|alt_dspbuilder_delay_GN.{8}:Delay
```

```
my_model_GN:auto_inst|alt_dspbuilder_delay_GN.{8}:Delay1
```

Figure 3–26 shows this example in the Quartus Prime Assignment Editor.

Figure 3–26. Preserve Registers Assignment in the Quartus Prime Assignment Editor

	From ▽	To	Assignment Name	Value	Enabled
1		◇ clk	Clock Settings	clk	Yes
2		◇ preserve_regs_GN:auto_inst|alt_dspbuilder_delay_GN.{8}:Delay	Preserve Registers	On	Yes
3		◇ preserve_regs_GN:auto_inst|alt_dspbuilder_delay_GN.{8}:Delay1	Preserve Registers	On	Yes
4	<<new>>	<<new>>	<<new>>		

This type of assignment can be useful for a complicated block that contains many registers when you want the assignment to apply to all of the registers.

☞ Note: DSP Builder standard blockset is a legacy product and Altera recommends you do not use it for new designs, except as a wrapper for advanced blockset designs.

Altera provides a number of parameterizable intellectual property (IP) MegaCore functions that you can integrate into the Simulink model of your DSP Builder designs.

☞ The OpenCore Plus evaluation feature allows you to download and evaluate these MegaCore functions in hardware and simulation prior to licensing.

Blocks represent these MegaCore functions in the MegaCore Functions library of the Altera DSP Builder Blockset in the Simulink Library Browser.

You must parameterize and generate these MegaCore functions after you add one of these blocks to your model.

Refer to "MegaCore Function Design Example" on page 4–3 for an example of the design flow using these MegaCore functions.

Installing MegaCore Functions

Altera DSP MegaCore functions install with the Quartus® II software.

Refer to the MegaCore function user guides for information about each MegaCore function.

You must run the DSP Builder MegaCore function setup command after installing new MegaCore functions to update DSP Builder.

To run this setup command, follow these steps:

1. Start DSP Builder in MATLAB, on Windows OS, click on **Start**, point to **All Programs**, click **Altera <version>**, click **DSP Builder**, and click **Start in MATLAB version XX**. If MATLAB is already running, ensure you close the Simulink library browser.

2. Use the cd command at the MATLAB prompt to change directory to the directory where DSP Builder is installed.

3. Run the setup command by typing the following at the MATLAB prompt:

```
alt_dspbuilder_setup_megacore ↵
```

☞ The process of building the MegaCore function blocks may take several minutes. Do not close MATLAB before the process completes. Expect and ignore any messages of the form "Cannot find the declaration of element 'entity'." when installing a new MegaCore library.

Running this command, creates a **MegaCore Functions** subfolder below the **Altera DSP Builder Blockset** in the Simulink Library Browser.

In this folder, there is a blue block with a version name for each of the installed MegaCore functions.

Updating MegaCore Function Variation Blocks

Although a DSP Builder design using MegaCore function blocks from the MegaCore Functions library can be translated by `Signal Compiler` into a VHDL or Verilog HDL model, a MegaCore function variation block always uses an intermediate VHDL file to record parameters.

These blocks may revert to their unconfigured appearance if the VHDL file that describes the function variation is available but the simulation database (**.simdb)** file is not.

Update a block if you change the version of the MegaCore function you are using. In these cases, you can update the MegaCore function variation blocks in your design using the `alt_dspbuilder_refresh_megacore` command. This command recreates the simulation files based on the VHDL file for each MegaCore function block in the current Simulink model.

☞ A Quartus II license must be available on the machine for the command to execute without errors.

Design Flow Using MegaCore Functions

Using MegaCore functions in the MATLAB or Simulink environment involves the following steps:

1. Add the MegaCore function to the Simulink model and give the block a unique name.

2. Parameterize the MegaCore function variation.

3. Generate the MegaCore function variation.

4. Connect your MegaCore function variation to the other blocks in your model.

5. Simulate the MegaCore function variation in your model.

👣 Refer to the appropriate MegaCore function user guide for information about the design flow used for each MegaCore function.

Adding the MegaCore Function in the Simulink Model

Add a MegaCore function to a Simulink model by dragging a copy of the block from the Simulink Library Browser to your design workspace like any other Simulink block.

The default name of a MegaCore function block includes its version number. If you add more than one copy of a block in the same model, this number is automatically incremented to make the name unique. The correct version number still shows on the body of the block. Altera recommends that you rename all blocks representing MegaCore functions with a name describing their use in your design. Using unique block names ensures that all the generated entities for the same MegaCore function in a hierarchical design also have unique names.

After adding the block and before parameterization, save your model file.

Parameterizing the MegaCore Function Variation

Double-click the MegaCore function block to open the IP Toolbench or MegaWizard interface.

☞   You can also double-click on a block to re-open and modify a previously parameterized MegaCore function variation.

Generating the MegaCore Function Variation

Before you can connect the block to your design, generate a MegaCore function variation after you have parameterized the MegaCore function.

Click **Generate** in IP Toolbench (or **Finish** in the MegaWizard interface) to generate the necessary files for your MegaCore function variation.

DSP Builder also performs an additional step of optimizing your model for use in Simulink.

Connecting the MegaCore Function Variation Block to the Design

The Simulink block now has the required input and output ports as parameterized in IP Toolbench or the MegaWizard interface. You can connect these ports to other Altera DSP Builder blocks in your Simulink design.

Simulating the MegaCore Function Variation in the Model

You can simulate the Simulink block representing the MegaCore function variation like any other block from the Simulink Library Browser.

☞   Ensure that the Simulink simulation engine is set to use the discrete solver by selecting **fixed-step** type under **Solver Options** in the **Configuration Parameters** dialog box.

You should reset the MegaCore function at the start of the simulation to avoid any functional discrepancy between RTL simulation and Simulink simulation ("Startup and Initial Conditions" on page 3–17).

MegaCore Functions Design Issues

This section describes some of the design issues to consider when using MegaCore functions in a DSP Builder design.

Simulink Files Associated with a MegaCore Function

DSP Builder stores the files that support the configuration and simulation of a MegaCore function variation in a subdirectory of the directory containing your Simulink MDL file **DSPBuilder_<*design name*>_import**. When copying a design from one location to another, make sure that you also copy this subdirectory.

DSP Builder needs the following specific files to simulate a MegaCore function variation:

■ If your MegaCore function variation is `my_function`, and generates in VHDL, your design variation is in a **my_function.vhd** file in your design directory.

■ If your design is `my_design`, the simulation information is in a **DSPBuilder_my_design_import/my_function.vo.simdb** file.

About Simulating MegaCore Functions That Have a Reset Port

MegaCores functions that have a reset port must have a reset cycle at the start of Simulink simulation to produce correct simulation results. The length of this reset cycle must be of sufficient length, and depends on the particular MegaCore function and parameterization.

For example, in Figure 4–1, DSP Builder cannot tie the reset to a constant because the simulation does not match hardware.

Figure 4–1. MegaCore Function Design With a Reset Port
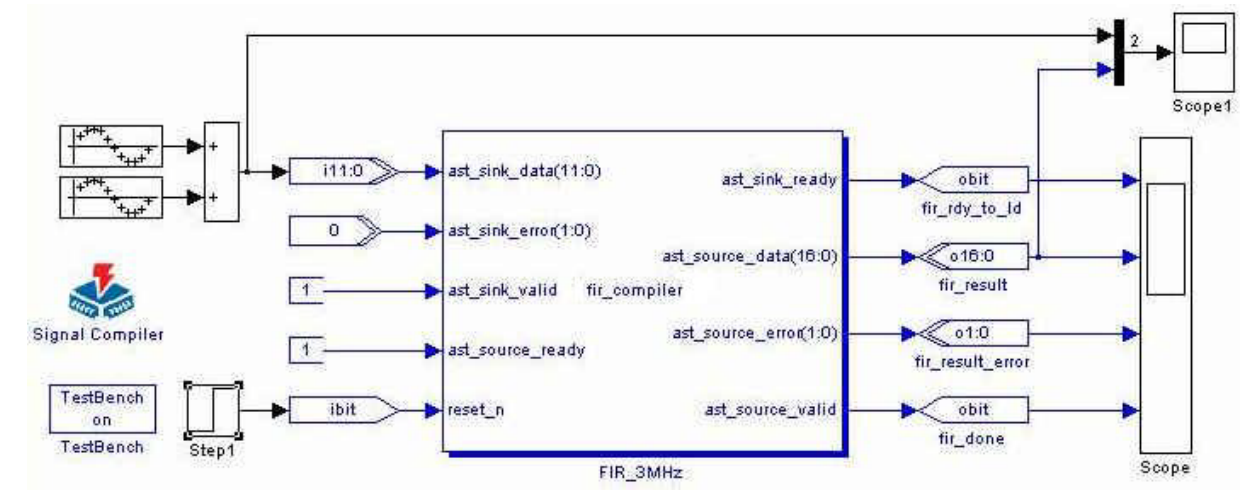
You must simulate an initial reset cycle (with the step input) to replicate hardware behavior. As in hardware, this reset cycle must be sufficiently long to propagate through the core, which may be 50 clock cycles or more for some MegaCore functions such as the FIR Compiler.

Additional adjustment of the reset cycles may be necessary when a MegaCore function receives data from other MegaCore functions, to ensure that the blocks leave the reset state in the correct order and DSP Builder delays them by the appropriate number of cycles.

About Setting the Device Family for MegaCore Functions

Most of the MegaCore functions available in DSP Builder use the IP Toolbench interface.

The CIC MegaCore function uses a MegaWizard user interface. This interface always inherits the device family setting from the `Signal Compiler` block. If there is no `Signal Compiler` block in your design, DSP Builder uses the Stratix device family by default.

MegaCore functions that use IP Toolbench allow you to modify the device family setting in the IP Toolbench interface.

☞ The FFT, FIR Compiler, NCO, Reed Solomon Compiler, and Viterbi Compiler MegaCore functions use IP Toolbench.

If you change the device family in `Signal Compiler`, you must check that any IP Toolbench MegaCore functions have the correct device family set to ensure that the simulation models and generated hardware are consistent.

Adding the HIL block to your Simulink model allows you to cosimulate a Quartus Prime software design with a physical FPGA board implementing a portion of that design. You define the contents and function of the FPGA by creating and compiling a Quartus II project. A simple JTAG interface between Simulink and the FPGA board links the two.

The main benefits of using the HIL block are faster simulation and richer instrumentation. The Quartus Prime project you embed in an FPGA runs faster than a software-only simulation. To further increase simulation speed, the HIL block offers frame and burst modes of data transfer that are significantly faster than single-step mode when you use it with suitable designs.

The HIL block also makes available to the hardware a large Simulink library of sinks and sources, such as channel models and spectrum analyzers, which can give you greater control and observability.

This chapter explains the HIL block design flow, walks through an example using the HIL block, and discusses the optional burst and frame data transfer modes.

HIL Design Flow

The HIL block in AltLab library of the **Altera DSP Builder Blockset** enables the HIL functionality. It represents the functions implemented on your FPGA, and works smoothly with the normal DSP Builder work flow.

The HIL design flow comprises the following steps:

1. Create a Quartus Prime project that defines the functions you want to co-simulate in hardware and use Signal Compiler block to compile the Quartus Prime project through the Quartus Prime Fitter.

2. Add the HIL block to your Simulink model and import the compiled Quartus Prime project into the HIL block. You can also connect instrumentation to your HIL block by adding additional blocks from the Simulink Sinks and Sources libraries.

    ☞ If the original design contains a Clock block that defines a period and sample time that is different from the default values, you must add a Clock block with the same values as the HIL block.

3. Specify parameters for the HIL block, including the following options:

    ■ The Quartus Prime project to define its functionality

    ■ The clock and reset pins

    ■ The reset active level

    ■ The input and output pin characteristics

    ■ The use of single-step versus burst mode

4. Compile the HIL block to create a programming object file (.**pof**) for hardware cosimulation.

5. Scan for JTAG cables and hardware devices connected to the local host or any remotely enabled hosts.

6. Program the board that contains your target FPGA.

7. Simulate the combined software and hardware system in Simulink.

☞ When using a `HIL` block in a Simulink model, set a fixed-step, single tasking solver.

Figure 5–1 shows this system-level design flow using DSP Builder.

Figure 5–1. System-Level Design Flow
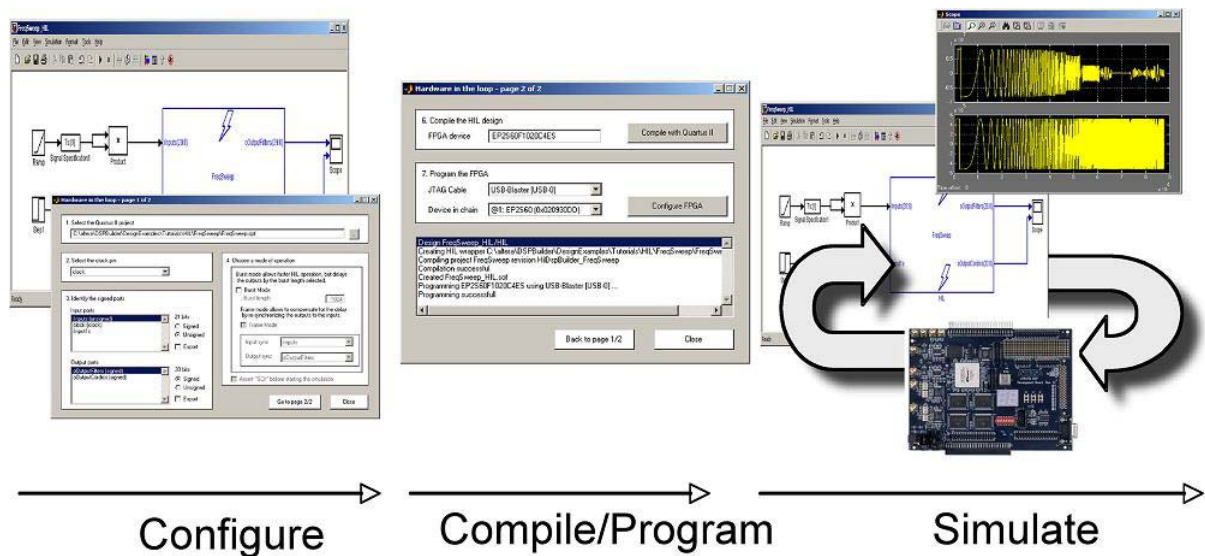
HIL Requirements

The `HIL` block has the following requirements:

■ An FPGA board with a JTAG interface

■ A valid Quartus Prime project that contains a single clock domain from Simulink. DSP Builder creates an internal Quartus Prime project when you run `Signal Compiler`.

■ A JTAG download cable (for example, a ByteBlasterMV™, ByteBlaster™ II, ByteBlaster, MasterBlaster™, or USB-Blaster™ cable).

■ A maximum of one `HIL` block for each JTAG download cable.

HIL Design Example

DSP Builder includes the following design examples in the *<DSP Builder install path>***DesignExamples\Tutorials\HIL** directory that demonstrate the use and effectiveness of HIL:

■ Imaging edge detection

■ Export example

■ Fast Fourier Transform (FFT)

■ Frequency sweep

This section shows the frequency sweep design.

☞ This tutorial uses the Stratix IV device on an Stratix IV EPSGX530 Development Board. However, you can also use any other supported device and development board.

To create a frequency sweep design, follow these steps:

1. Run MATLAB, and open the model **FreqSweep.mdl** in the *<DSP Builder install path>***DesignExamples\Tutorials\HIL\FreqSweep** directory. Figure 5–2 shows the model.

Figure 5–2. Frequency Sweep Model
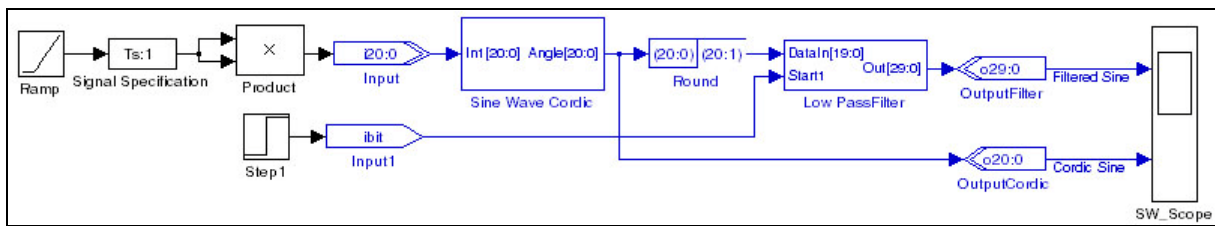
2. Double-click the `Signal Compiler` block. In the dialog box that appears
   (Figure 5–3 on page 5–4), click **Compile**.

   This action creates a Quartus Prime project, **FreqSweep.qpf**, compiles your model
   for synthesis, and runs the Quartus Prime Fitter.

   Progress is indicated by status messages and a scrolling bar at the bottom of the
   dialog box.

Figure 5–3. Signal Compiler Dialog Box, Simple Tab
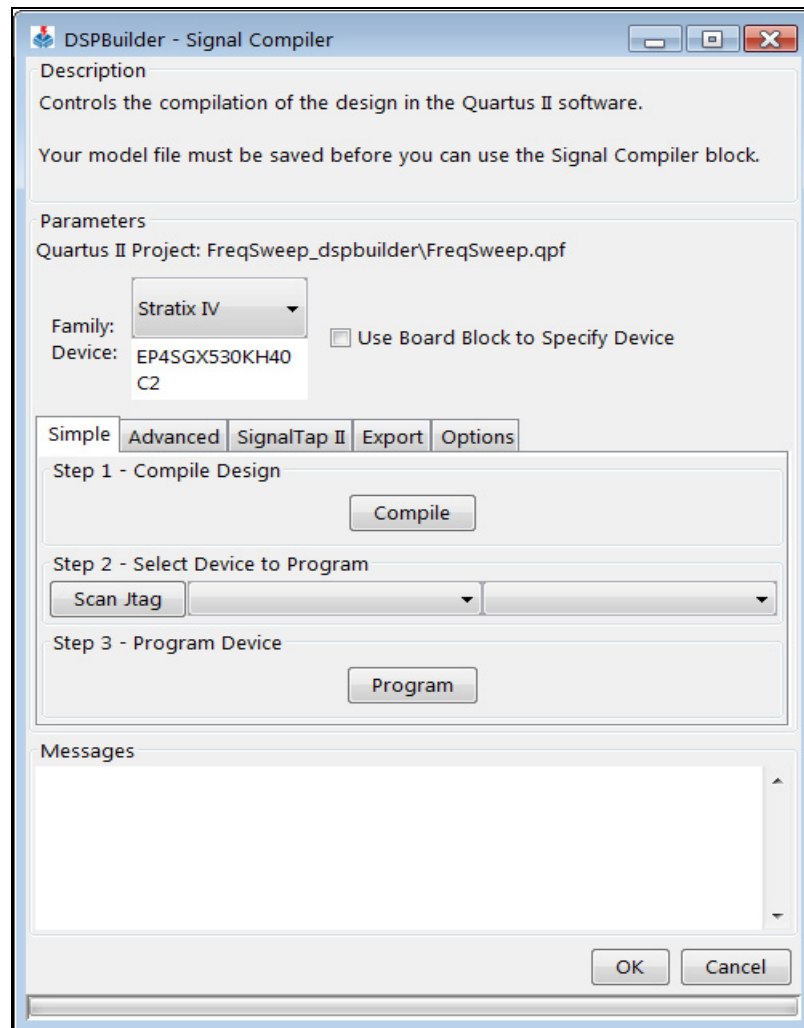
3. Review the Messages, then click **OK** to close the **Signal Compiler** dialog box.

4. Replace the internal functions of the frequency sweep model with an HIL block. Open the model **FreqSweep_HIL.mdl** from the **FreqSweep** directory (step 1).

   Figure 5–4 shows this model, with the HIL block in place.

Figure 5–4. Frequency Sweep Design Model Using the HIL Block
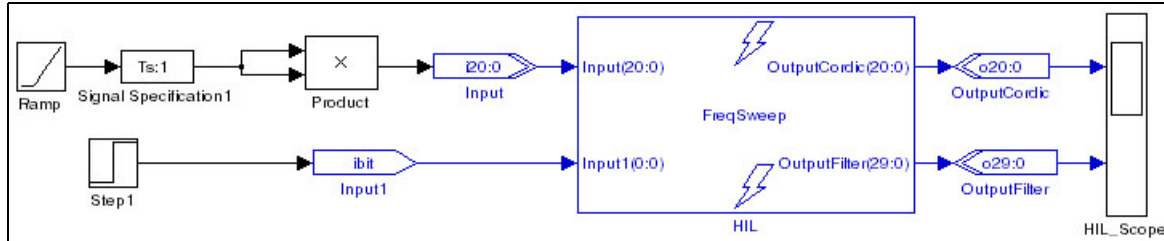
5. Double-click the frequency sweep HIL block to display the **Hardware in the loop** dialog box.

6. Select the Quartus Prime project by browsing into the **FreqSweep_dspbuilder** directory to locate the **FreqSweep.qpf** file.

   ☞ The full path to this file is visible in the dialog box when you select this file.

7. Select **Clock** from the list of available clock pins.

   ☞ HIL does not support multiple clock domains and only the specified signal is the HIL clock signal. The HIL treats any other clocks in your design as input signals.

8. Select **aclr** from the list of available reset pins.

9. Identify the signed ports:

   ■ Select the **Input** port and click **Unsigned**.

   ■ Select each output port (**OutputCordic** and **OutputFilter**) and click **Signed**.

10. Select the reset level to be **Active_High**.

11. Select the mode of operation by turning off **Burst Mode**.

12. Click **Next page**. to display the second page of the **Hardware in the loop** dialog box.

13. Specify a value for the **FPGA device** and click **Compile with Quartus Prime** to compile the HIL design.

    ☞ If no output writes to the MATLAB command window, check that the original Quartus Prime project is up-to-date and compiles with he same version of the Quartus Prime software that compiles your Simulink model.

14. Click **Scan Jtag** to find available cables and hardware devices in the chain.

15. Select the JTAG download cable that references the required FPGA device and click **Configure FPGA** to program the FPGA on the board.

16. Click **Close**.

17. Simulate your design in Simulink. Figure 5–5 shows the scope display from the finished design.

Figure 5–5. Scope Output from the FrequencySweep Model with HIL Block
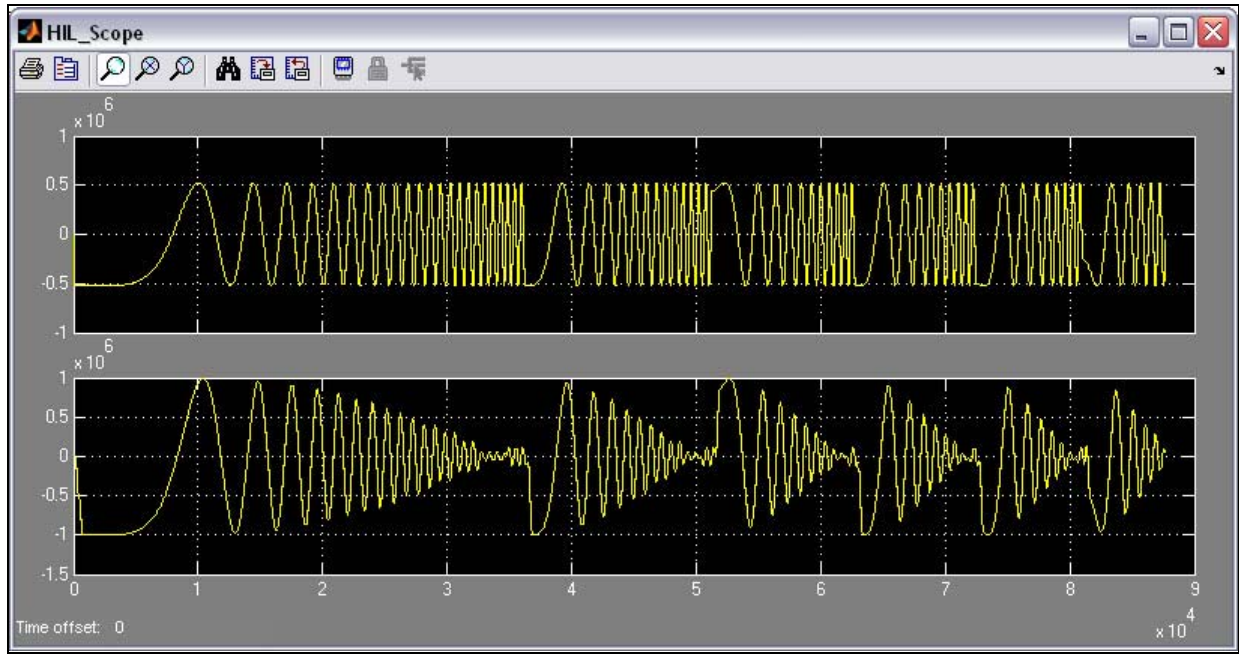
Burst Mode

The Quartus Prime software infrastructure that communicates with the FPGA through JTAG—system-level debugging (SLD)—uses a serial data transfer protocol.

To maximize the throughput of this data transfer, the HIL block offers a burst mode that buffers the stimulus data and presents it in bursts to the hardware.

Table 5–1 shows the advantages and disadvantages of using burst mode compared with the normal single-step mode.

Table 5–1. Comparing Single-Step and Burst Modes

Mode	Advantages	Disadvantages
Single step	Cycle accurate simulation. Feedback is possible outside of the HIL block.	High SLD overhead.
Burst	■ Low SLD overhead. ■ Fast HIL results.	A latency is introduced on the output signals of the HIL block making feedback loop difficult outside the FPGA device.

Using Burst Mode

To activate burst mode turn on the **Burst Mode** option in the **Hardware in the loop** dialog box.

When you set this option, you can specify the required number of data packets as the **Burst length**. The HIL block sends data to the hardware in bursts of the size you specify.

☞ DSP Builder determines the size of the packet by the larger of the total input data width or the total output data width. If the packet size multiplied by the **Burst length** exceeds the preset data array, DSP Builder sets the **Burst length** to 1.

Simulation using burst mode works the same as single clock mode, but DSP Builder introduces a latency of the specific packet size on the output signals of the HIL blocks. As a consequence, feedback-loops may not work properly unless you enclose them in the HIL block, and some intervention may be necessary when comparing or visualizing HIL simulation results.

The HIL block uses software buffers to send and receive from the hardware, so you can change these buffer sizes without recompiling the HIL function.

Troubleshooting HIL Designs

This section describes various problems that you may encounter when you are using HIL designs.

☞ If the top-level of your design changes, compile and reload the Quartus Prime project into HIL to ensure that all information is up-to-date.

Fails to Load the Specified Quartus Prime Project

HIL reads design information, such as clock, reset, and input and output ports, from the specified Quartus Prime project. However, it can fail to load your project if the project is not compiled with the Quartus Prime Fitter, there is a Quartus Prime version mismatch, or the Quartus Prime project file is not up-to-date.

No Inputs Found From the Quartus Prime Project

This problem occurs if the DSP Builder model file contains only the internally induced signals, such as from a counter, and also does not produce any outputs. However, HIL simulation works correctly.

No Outputs Found From the Quartus Prime Project

This problem occurs if your design does not have any outputs and makes the HIL simulation meaningless.

HIL Design Stays in Reset During Simulation

An asynchronous reset is permanently asserted for a HIL design.

Check that the reset active level matches the setting in the original design. Recompile the HIL design after you have changed the reset level.

HIL Compilation Appears to Hang

After clicking **Compile with Quartus Prime** in the HIL **Block Parameters** dialog box, no output writes to the MATLAB command window. This problem occurs if the original Quartus Prime project is out-of-date or compiled by a different version of the Quartus Prime software.

Scan JTAG Fails to Find Correct Cable or Device

This problem occurs if you connect the target DSP development board switch it on after you open the HIL dialog box.

This chapter describes how to set up and run the SignalTap® II logic analyzer. In this chapter, you analyze three internal nodes in a simple switch controller design named **switch_control.mdl**. This design flow works for any of the Altera development boards that DSP Builder supports.

For detailed information about the supported development boards, refer to the *Boards Library* chapter in the *DSP Builder Standard Blockset Libraries* section in volume 2 of the *DSP Builder Handbook*.

In this design, an LED on the development board turns on or off depending on the state of user-controlled switches and the value of the incrementer. The design consists of an incrementer function feeding a comparator, and four switches that feed into two AND gates. The comparator and AND gate outputs feed an OR gate, which feeds an LED on the DSP development board.

The SignalTap II logic analyzer captures the signal activity at the output of the two AND gates and the incrementer of the design loads into the Altera device on the development board. The logic analyzer retrieves the values and displays them in the MATLAB work space.

For more information about using the SignalTap II logic analyzer with the Quartus II software, refer to the Quartus Prime Help.

A `SignalTap II Logic Analyzer` block in DSP Builder includes the following characteristics:

- Has a simple, easy-to-use interface
- Analyzes signals in the top-level design file
- Uses a single clock source
- Captures data around a trigger point. 88% of the data is pre-trigger and 12% of the data is post-trigger

☞ Alternatively, you can use the Quartus II software to instantiate of the SignalTap II logic analyzer in your design. The Quartus II software supports additional features, such as using multiple clock domains, and adjusting the percentage of data captured around the trigger point.

SignalTap II Design Flow

Working with the SignalTap II logic analyzer in DSP Builder involves the following flow:

1. Add a `SignalTap II Logic Analyzer` block to your design.
2. Specify the signals (nodes) that you want to analyze by inserting `SignalTap II Node` blocks.

3. Turn on the **Enable SignalTap** option in the **Signal Compiler** dialog box.

4. Choose one of the JTAG cable ports in the **Signal Compiler** dialog box or the **SignalTap II Logic Analyzer** dialog box.

5. Using `Signal Compiler`, synthesize your model, perform compilation in the Quartus II software, and download your design into the DSP development board (starter or professional).

6. Specify the required trigger conditions in the `SignalTap II Logic Analyzer` block.

For details of the `SignalTap II Logic Analyzer` and `SignalTap II Node` blocks, refer to the descriptions of these blocks in the *AltLab Library* chapter in the *DSP Builder Standard Blockset Libraries* section in volume 2 of the *DSP Builder Handbook*.

SignalTap II Nodes

A node represents a wire carrying a signal that travels between different logical components of a design file. The SignalTap II logic analyzer can capture signals from any internal device node in a design file, including I/O pins.

The SignalTap II logic analyzer can analyze up to 128 internal nodes or I/O elements. As more it capture more signals, it uses more logic elements (LEs) or embedded system blocks (ESBs).

Before capturing signals, assign each node to analyze to a SignalTap II logic analyzer input channel. To assign a node to an input channel, you must connect it to a `SignalTap II Node` block.

SignalTap II Trigger Conditions

The trigger pattern describes a logic event in terms of logic levels or edges. The SignalTap II logic analyzer uses a comparison register to recognize the moment when the input signals match the data specified in the trigger pattern.

The trigger pattern comprises a logic condition for each input signal. By default, all signal conditions for the trigger pattern are set to **Don't Care**, masking them from trigger recognition. You can select one of the following logic conditions for each input signal in the trigger pattern:

- Don't care

- Low

- High

- Rising edge

- Falling edge

- Either edge

The SignalTap II logic analyzer triggers when it detects the trigger pattern on the input signals.

SignalTap II Design Examples

Altera provides several examples (Figure 6–1).

Figure 6–1. SignalTap II Design Example Directory Structure
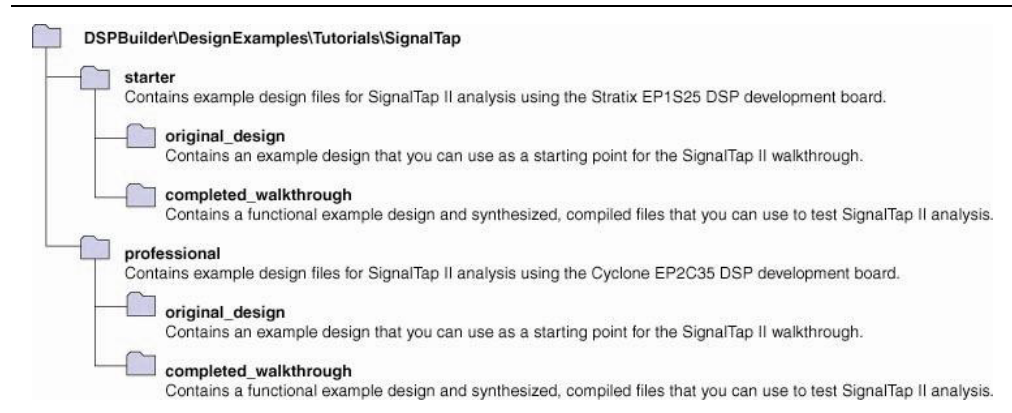
You can start from the design example in the **original_design** directory.

Alternatively, you can use the design example in the **completed_walkthrough** directory and go directly to .

Opening the Design Example

Open the template **switch_control.mdl** design in the *<DSP Builder install path>*\
DesignExamples\Tutorials\SignalTap\professional\original_design directory.
(Figure 6–2).

Figure 6–2. Starting Point for the SignalTap II Design Example
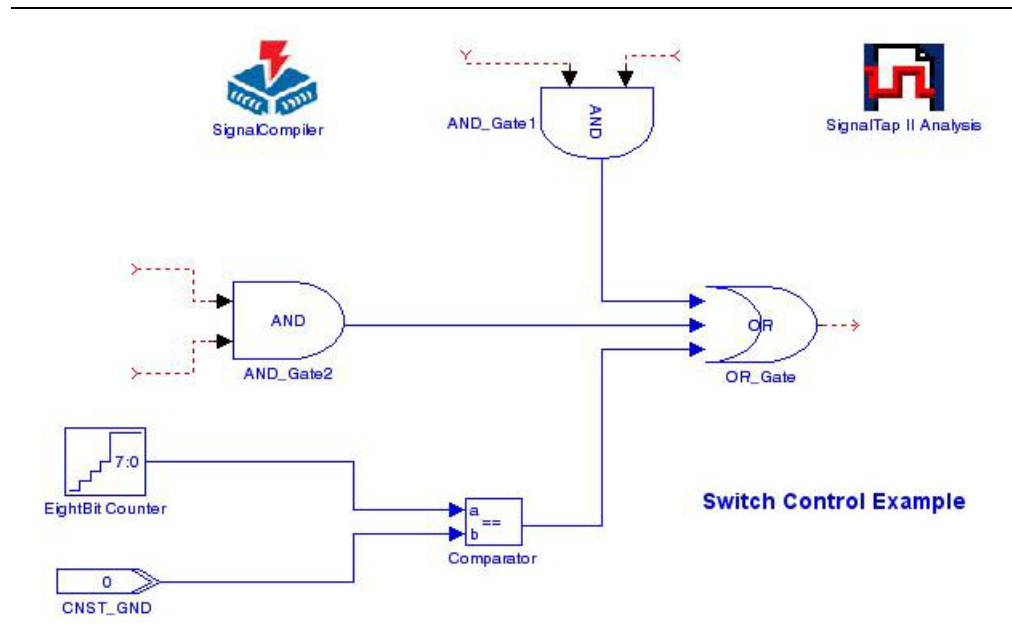
Adding the Configuration and Connector Blocks

Add the board configuration block and connector blocks for the board that you want to use. This tutorial uses the Stratix IV GX EP4SGX530 development board.

1. Select the **Boards** library from the **Altera DSP Builder Blockset** folder in the Simulink library browser.

2. Open the **StratixIVEP4SGX530** folder. Drag and drop the `Stratix IV GX EP4SGX530 FPGA Development Kit` configuration block into your model.

3. Drag and drop the `Dip switch` block close to the `AND_Gate2` block in your model. .

4. Drag and drop the `LED0` block close to the `OR_Gate` block in your model. Connect this block to the `OR_Gate` output.

5. Select the Simulink **Sources** library. Drag and drop a `Pulse Generator` block near to the `DIP Switch`.

6. Connect the new blocks as in Figure 6–3.

Figure 6–3. Switch Control Example with Board, Pulse Generator and Terminator Blocks
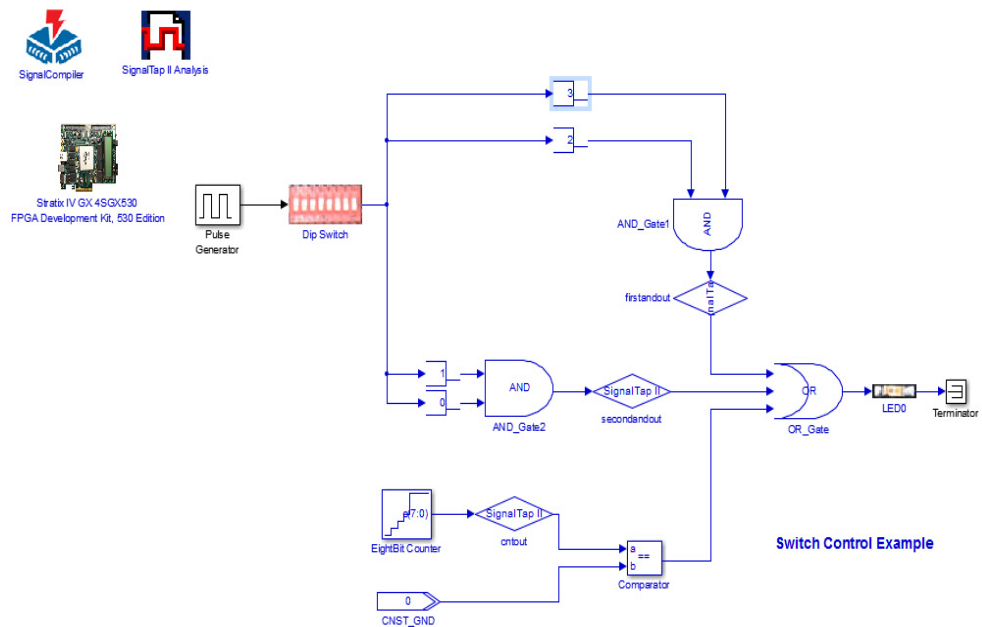
7. Use the **Block Parameters** dialog box to set the parameters (Table 6–1) for both pulse generator blocks.

Table 6–1. Parameters for the Pulse Generator Blocks

Parameter	Value
Pulse type	Time based
Time	Use Simulation time
Amplitude	1
Period	2

Table 6–1. Parameters for the Pulse Generator Blocks

Parameter	Value
Pulse Width	50
Phase delay	0
Interpret vector parameters as 1-D	On

8. Select the Simulink **Sinks** library. Drag and drop a `Terminator` block near to the `OR_Gate` block and connect it to this block.

Specifying the Nodes to Analyze

To add `SignalTap II Node` blocks to the signals (also called nodes) that you want to analyze (in this tutorial they are the output of each `AND` gate and the output of the incrementer), follow these steps:

1. Open the **AltLab** library in the Simulink Library Browser. Drag a `SignalTap II Node` block into your design. Position the block so that it is on top of the connection line between the `AND_Gate1` block and the `OR_Gate` block.

    ☞   If you position the block with this method, the Simulink software inserts the block and joins connection lines on both sides.

2. Click the text under the block icon in your model and change the block instance name by deleting the text and typing the new text `firstandout`.

3. Add a `SignalTap II Node` block between the `AND_Gate2` block and the `OR_Gate` block and name it `secondandout`.

4. Add a `SignalTap II Node` block between the `Eightbit Counter` block and the `Comparator` block and name it `cntout`.

5. Click **Save** on the File menu.

Turning On the SignalTap II Option in Signal Compiler

When you add node blocks to signals, each block implicitly connects to the SignalTap II logic analyzer. This connection is a functional change to your model and you must recompile your design before you can use the SignalTap II logic analyzer.

To compile your design, follow these steps:

1. Double-click the `Signal Compiler` block and click the **SignalTap II** tab in the **Signal Compiler** dialog box.

2. Verify that the **Enable SignalTap II** option is on.

    When this option is on, `Signal Compiler` inserts an instance of the SignalTap II logic analyzer into your design.

3. Select a depth of **128** for the SignalTap II sample buffer (that is, the number of samples stored for each input signal) in the **SignalTap II depth** list.

4. Verify that the **Use Base Clock** option is on.

5. Click the **Simple** tab and verify that the **Use Board Block to Specify Device** option is on.

6. Click the **Compile** button.

   When the conversion is complete, information messages in the dialog box display the memory allocated during processing.

   ☞ You must compile your design before you open the `SignalTap II Analyzer` block because the block relies on data files that create during compilation.

7. Click **Scan Jtag** and select the appropriate download cable and device (for example, **USB-Blaster** cable and **EP2C35** device).

8. Click **Program** to download your design to the development board.

9. Click **OK**.

Specifying the Trigger Levels

To specify the trigger levels, follow these steps:

1. Double-click the `SignalTap II Logic Analyzer` block. The dialog box displays all the nodes connected to `SignalTap II Node` blocks as signals to be analyzed.

2. Specify the following trigger condition settings for the `firstandout` block:

   a. Click **firstandout** under `Signal Tap II Nodes`.

   b. Select **Falling Edge** in the **Set Trigger Level** list.

   c. Click **Change**. The condition is updated.

3. Repeat these steps to specify the trigger condition **High** for the `secondandout` block.

The SignalTap II logic analyzer captures data for analysis when it detects all trigger patterns simultaneously on the input signals. For example, because you specify **Falling Edge** for `firstandout` and **High** for `secondandout`, the SignalTap II logic analyzer only triggers when it detects a falling edge on `firstandout` and a logic level high on `secondandout`.

Performing SignalTap II Analysis

You are now ready to run the analyzer and display the results in a MATLAB plot. After you click **Acquire**, the SignalTap II logic analyzer begins analyzing the data and waits for the trigger conditions to occur. To perform analysis, follow these steps:

1. Click **Scan Jtag** in the **SignalTap II Logic Analyzer** dialog box and select the appropriate download cable and device.

2. Click **Acquire**.

3. Press switch SW4 on the DSP development board to trigger the SignalTap II logic analyzer.

   ☞ If you press and hold switch SW2 or SW3 while pressing switch SW4, the trigger condition is not met and acquisition does not occur.

4. Click **OK** in the **SignalTap II Logic Analyzer** dialog box when you finish.

DSP Builder interprets the captured data as unsigned values and displays them in MATLAB plots. It stores the values in MATLAB **.mat** files in the working directory.

Figure 6–4 shows the MATLAB plot for the SignalTap II node firstandout.

Figure 6–4. MATLAB Plot for SignalTap II Node firstandout
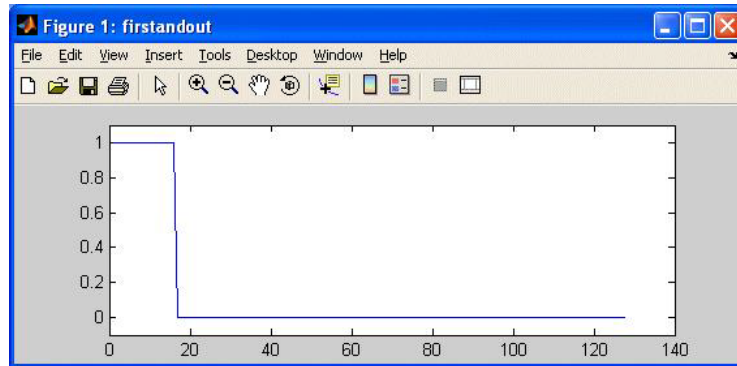
Figure 6–5 shows the MATLAB plot for the SignalTap II node secondandout.

Figure 6–5. MATLAB Plot for SignalTap II Node secondandout
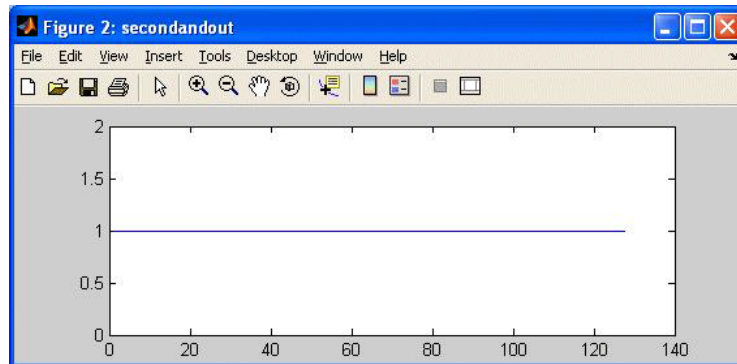
Figure 6–6 shows the MATLAB plot for the SignalTap II node cntout.

Figure 6–6. MATLAB Plot for SignalTap II Node cntout
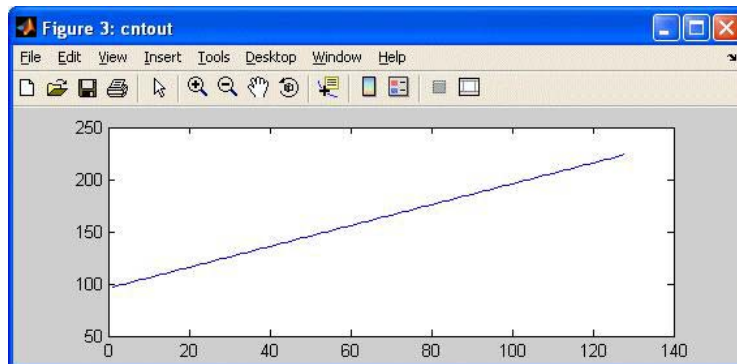
For more information about the `SignalTap II Logic Analyzer` block, refer to the *SignalTap II Logic Analyzer* block description in the AltLab Library chapter in the *DSP Builder Standard Blockset Libraries* section in volume 2 of the *DSP Builder Handbook*.

This chapter describes how to use the Avalon-MM blocks in the Interfaces library to create a design that functions as a custom peripheral to Qsys.

Qsys is a system development tool for creating systems that can contain processors, peripherals, and memories. Qsys automates the task of integrating hardware components into a larger system.

To integrate a DSP Builder design into your Qsys system, your peripheral must meet the Avalon-MM interface or Avalon-ST interface specification.

The Interfaces library supports peripherals that use the Avalon-MM and Avalon-ST interface specifications.

☞ The correct version of MATLAB with DSP Builder must be available on your system path to integrate DSP Builder .**mdl** files in Qsys.

Avalon-MM Interface

The *Avalon Interface Specifications* provide peripheral designers with a basis for describing the address-based read and write interfaces on master (for example, a microprocessor or DMA controller) and slave peripherals (for example, a memory, UART, or timer).

The `Avalon-MM Master` and `Avalon-MM 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 Qsys system. These blocks provide you the following benefits:

■ Automates the process of specifying Avalon-MM ports that are compatible with the Avalon-MM bus

■ Supports multiple Avalon-MM master and Avalon-MM slave instantiations

■ Saves time spent hand coding glue logic that connects Avalon-MM ports to DSP blocks

For more information about the Avalon-MM Interface, refer to the *Avalon Interface Specifications*.

Avalon-MM Interface Blocks

A Qsys component is a design module that Qsys recognizes and can automatically integrate into a system.

Qsys can recognize a DSP Builder design model if it is in the same working directory as the Qsys project. With the Avalon-MM blocks in the Interfaces library, you can design the DSP function and add an Avalon-MM block that makes it a custom peripheral in the Simulink environment.

You can instantiate each Avalon-MM block multiple times in a design to implement an SOPC component with multiple master or slave ports.

Avalon-MM Slave Block

Figure 7–1 shows a block that describes an Avalon-MM slave interface where all the Avalon-MM signals are enabled.

Figure 7–1. Avalon-MM Slave Block Signals
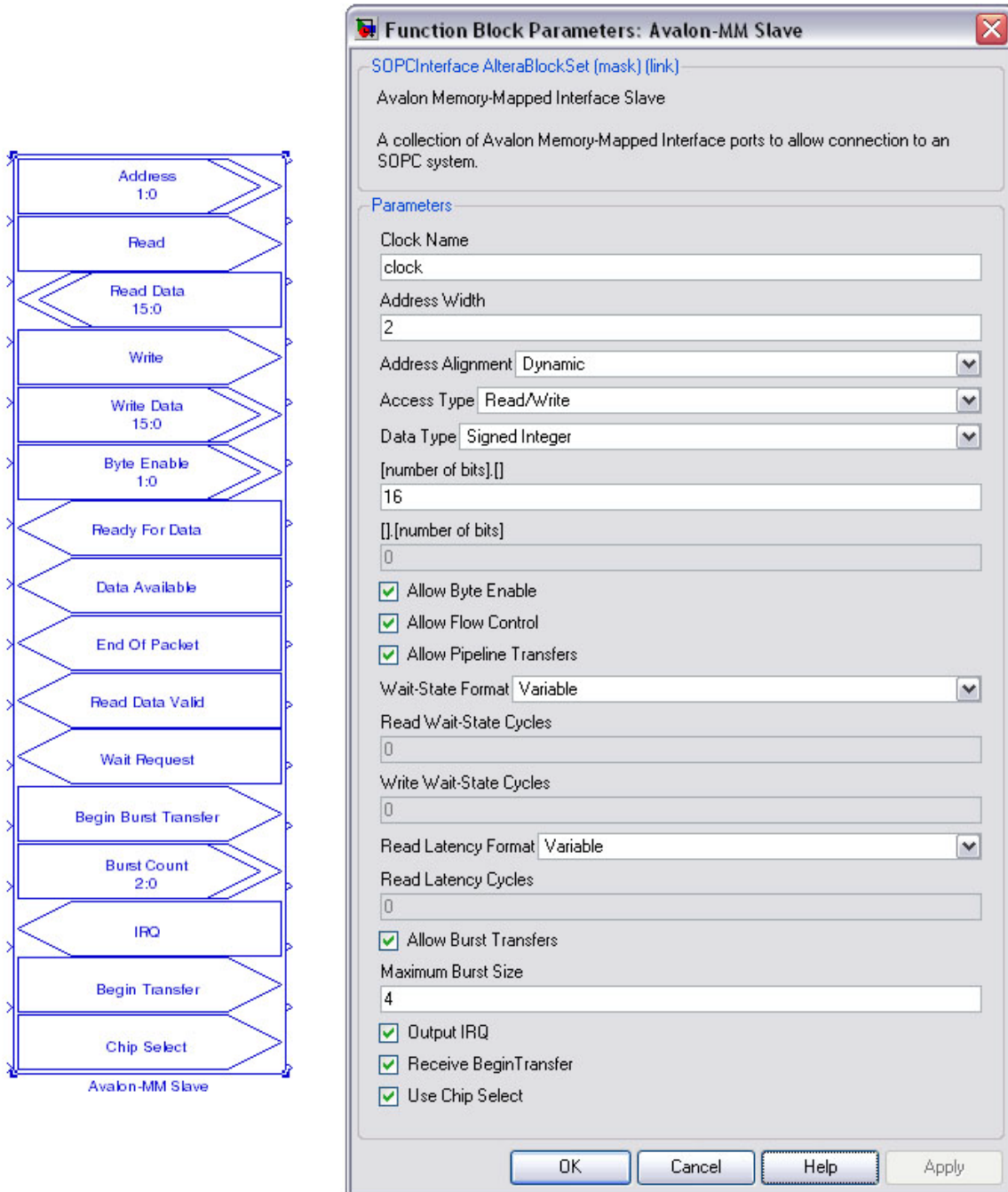
Each of the input and output ports of the block correspond to the input and output ports of the pin or bus that Figure 7–1 shows between the ports.

Inputs to the DSP Builder core display as right pointing bus or pins; outputs from the core display as left pointing pins or busses.

You can use the opposite end of any pins to provide pass-through test data from the Simulink domain.

Avalon-MM Master Block

You may want to use an `Avalon-MM Master` block (for example, to design a DMA controller) in a design that functions as an Avalon-MM Master in your Qsys system.

For more information about these signals, refer to the *DSP Builder Standard Blockset Libraries* section in volume 2 of the *DSP Builder Handbook*.

Figure 7–2 on page 7–3 shows a block that describes an Avalon-MM master interface where all the Avalon-MM signals are enabled.

Figure 7–2. Avalon-MM Master Block Signals
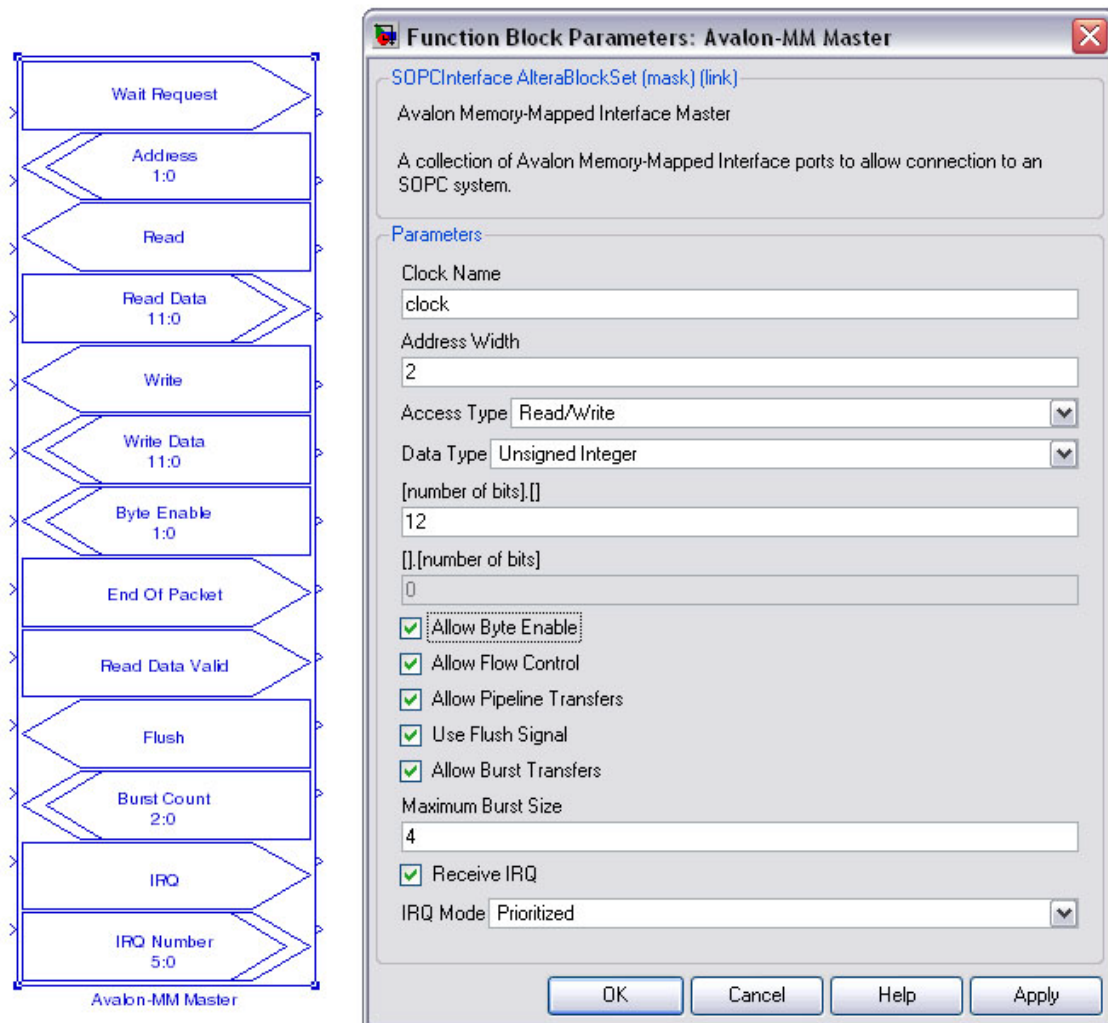
Wrapped Blocks

The `Avalon-MM Master` and `Avalon-MM Slave` interface blocks allow you to generate a Qsys component in DSP Builder, but they do little to mask the complexities of the interface. The Avalon-MM read and write FIFO blocks in the Interfaces library provide a higher level of abstraction.

You can implement a typical DSP core that handles data in a streaming manner, with the signals `Data`, `Valid`, and `Ready`. To provide a high level view, DSP Builder provides you with configurable `Avalon-MM Write FIFO` and `Avalon-MM Read FIFO` blocks for you to map Avalon-MM interface signals to this protocol.

Figure 7–3 shows an example system with `Avalon-MM Write FIFO` and `Avalon-MM Read FIFO` blocks.

Figure 7–3. Example System with Avalon-MM Write FIFO and Avalon-MM Read FIFO Blocks
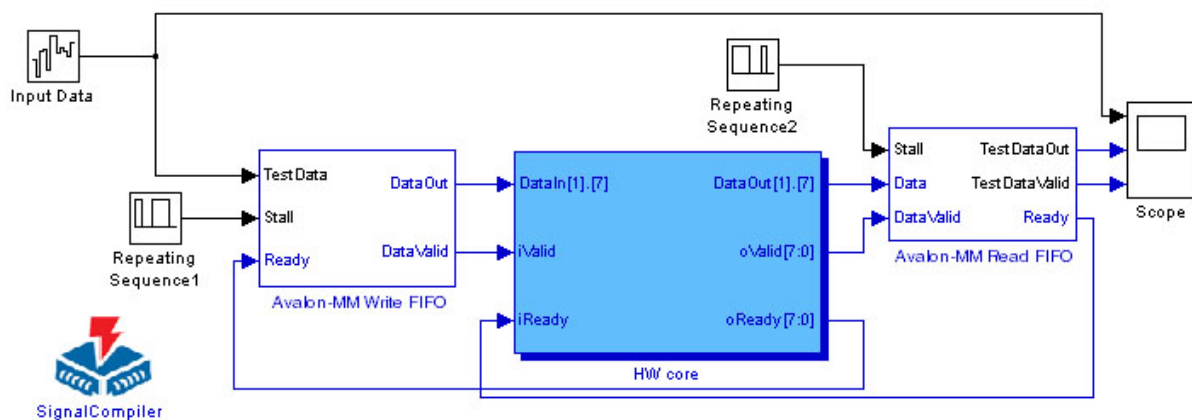
Looking at the Avalon-MM Write FIFO

1. Double-click on an `Avalon-MM Write FIFO` block to open the **Block Parameters** dialog box so that you can set parameters for the data type, data width and FIFO depth.

2. Open the hierarchy below the `Avalon-MM Write FIFO` block, by right-clicking the block and click **Look Under Mask** on the pop-up menu.

You can use this design as a template to design new functionality (for example, when you use an Avalon-MM address input to split incoming streams).

Figure 7–4 shows the internal content of an Avalon-MM Write FIFO buffer.

Figure 7–4. Avalon-MM Write FIFO Content
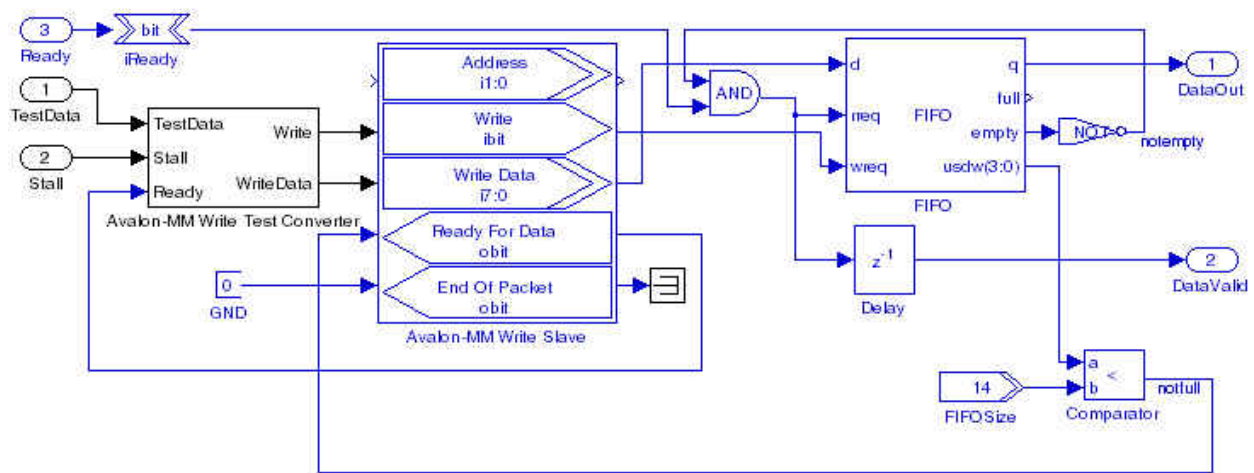
The `Avalon-MM Write Test Converter` block handles caching and conversion of Simulink or MATLAB data into accesses over the Avalon-MM interface. You can use this block to test the functionality of your design. The `Avalon-MM Write Test Converter` is simulation only and does not synthesize to HDL.

Looking at the Avalon-MM Read FIFO Buffer

1. Double-click on an `Avalon-MM Write FIFO` block to open the **Block Parameters** dialog box that you can use to set parameters for the data type, data width and FIFO depth.

2. Open the hierarchy below the `Avalon-MM Read FIFO` block by right-clicking on the block and choosing **Look Under Mask** from the pop-up menu.

Figure 7–5 shows the internal content of an Avalon-MM Read FIFO.

Figure 7–5. Avalon-MM Read FIFO Content
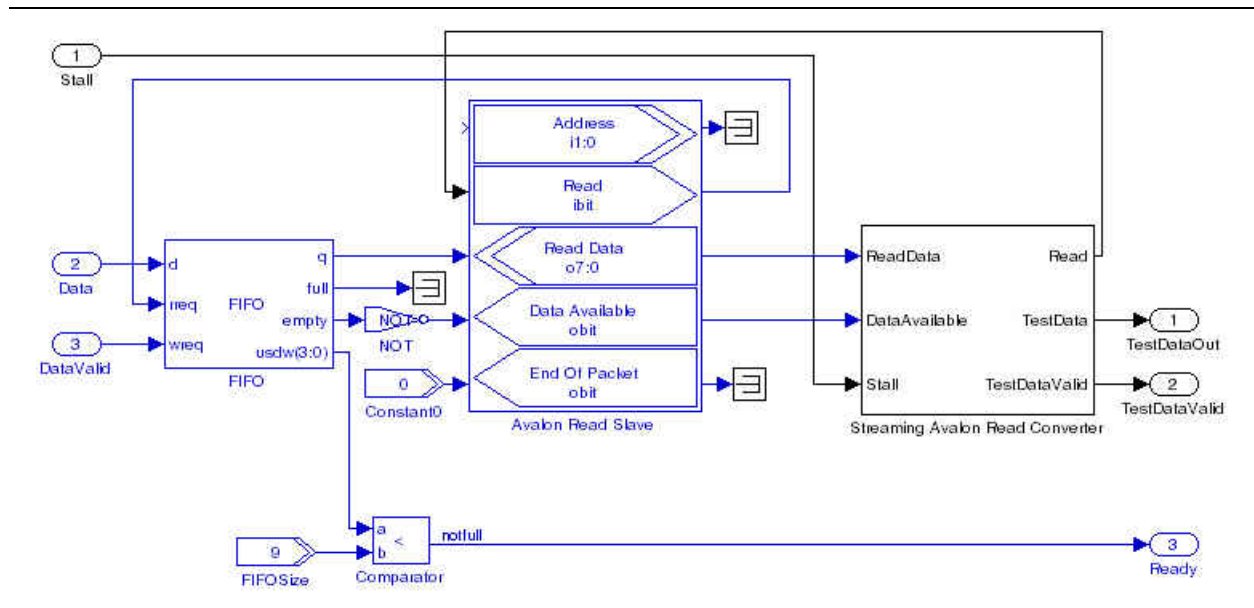
The `Avalon-MM Read Data Converter` block handles caching and conversion of Simulink or MATLAB data into accesses over the Avalon-MM interface. You can use this block to test the functionality of your design. The `Avalon-MM Read Data Converter` is simulation only and does not synthesize to HDL.

Avalon-MM Interface Blocks Design Example

This tutorial describes how to interface a design using the Avalon-MM blocks as a custom peripheral to the Nios II embedded processor in Qsys.

The design consists of a 4-tap FIR filter with variable coefficients. You load the coefficients with the Nios II embedded processor while an off-chip source supplies the input data through an analog-to-digital converter. The design sends filtered output data off-chip through a digital-to-analog converter.

Adding Avalon-MM Blocks to the Design Example

To complete the design example, follow these steps:

1. Click **Open** in the MATLAB software.

2. Browse to the **<*DSP Builder install path*>\DesignExamples\Tutorials\ SOPCBuilder\SOPCBlock** directory.

3. Select the **new_topavalon.mdl** file and click **Open**.

Figure 7–6 shows **new_topavalon.mdl**.

Figure 7–6. new_topavalon.mdl Design Example
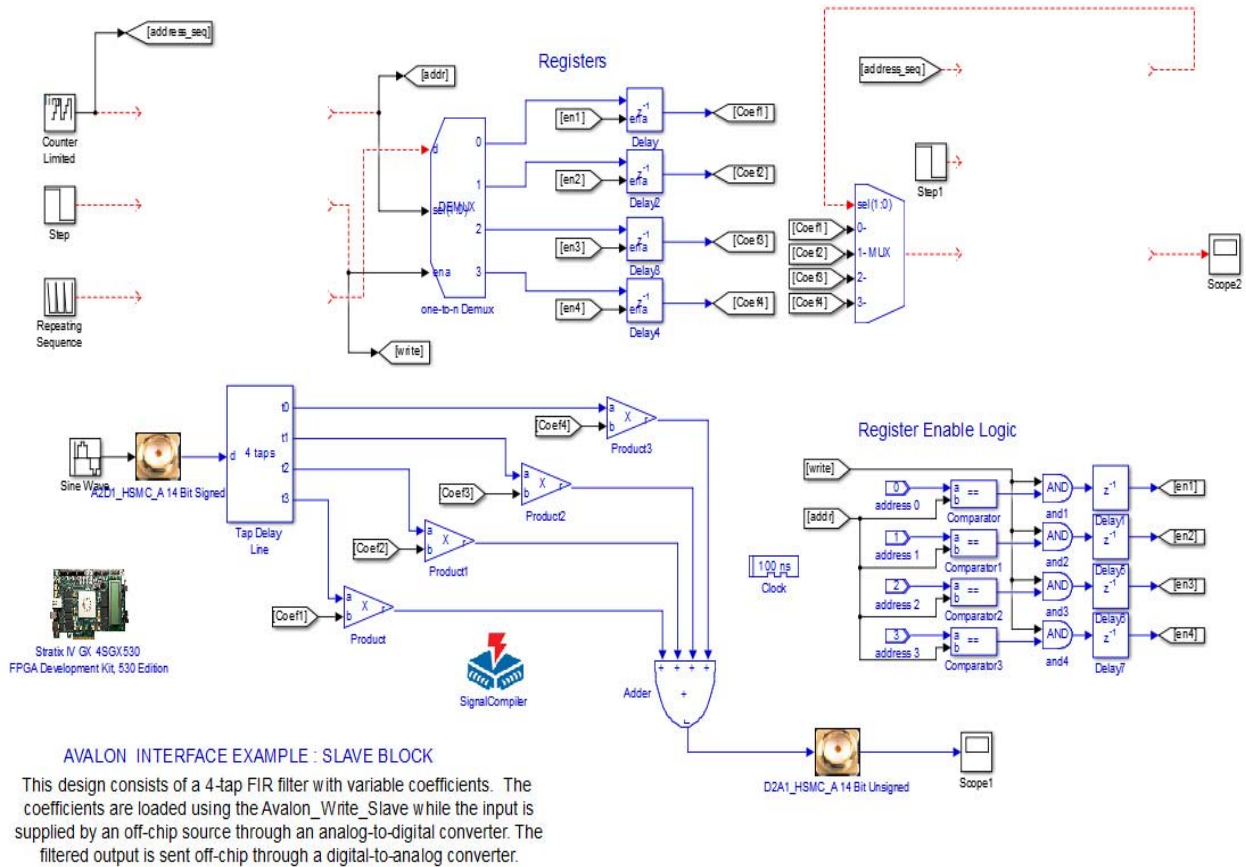
4. Rename the file by clicking **Save As** on the File menu. Create a new folder **MySystem** and save your new MDL file as **topavalon.mdl** in this folder.

5. Open the Simulink Library Browser. Expand the **Altera DSP Builder Standard Blockset** and select **Avalon Memory-Mapped** in the **Interfaces** library.

6. Drag and drop an `Avalon-MM Slave` block into the top left of your model. Change the block name to `Avalon_MM_Write_Slave`.

7. Double-click on the `Avalon_MM_Write_Slave` block to bring up the **Block Parameters** dialog box.

8. Select **Write** for the address type, **Signed Integer** for the data type, and specify `8` bits for the data width. Turn off the **Allow Byte Enable** option.

9. Click **OK**.

The `Avalon_MM_Write_Slave` block redraws with three ports: `Address i1:0`, `Write ibit`, and `Write Data i7:0`.

10. Connect the ports (Figure 7–7).

☞ You can re-size a block by dragging the resize handles at each corner.

Figure 7–7. topavalon.mdl Design Example
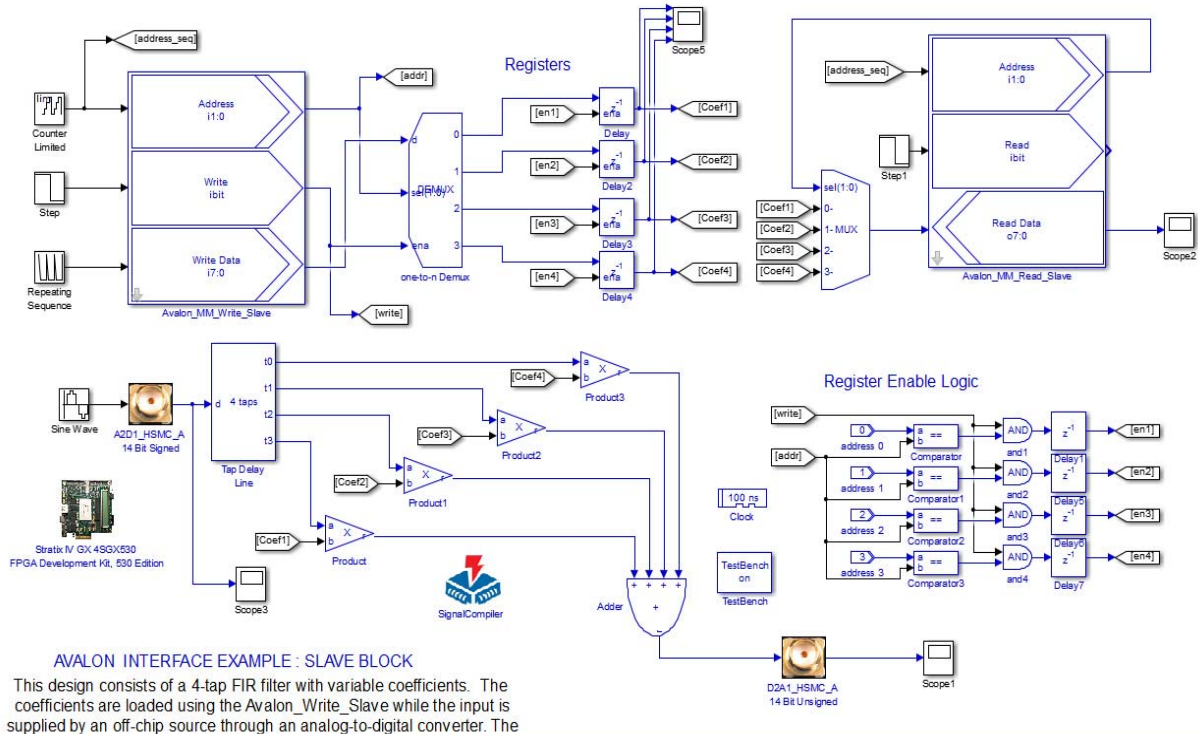
11. Drag and drop another `Avalon-MM Slave` block into the top right of your model and change the name of this block instance to `Avalon_MM_Read_Slave`.

12. Double-click on the `Avalon_MM_Read_Slave` block to bring up the **Block Parameters** dialog box.

13. Select **Read** for the address type, `Signed Integer` for the data type, and specify `8` bits for the data width.

14. Click **OK** and notice that the `Avalon_MM_Read_Slave` block redraws with three ports: `Address i1:0`, `Read ibit`, and `Read Data o7:0`.

15. Complete your design by connecting the `Avalon_MM_Read_Slave` ports (Figure 7–7).

    🌑 The default design example uses the Stratix IV Development Board. If you have a different board, you must replace the board block and analog-to-digital converter blocks by corresponding blocks for the appropriate board.

16. Add oscilloscope probes to monitor the signals on the development board.

17. Click **Save** on the File menu in your model window to save your model.

18. Run a simulation and observe the results on the oscilloscope probes. Coefficient values 1 0 0 0 load into the filter.

Verifying the Design

Before using your design in Qsys, use the `TestBench` block to verify your design. To verify your design, follow these steps:

1. Double-click the `TestBench` block to display the **TestBench Generator** dialog box (Figure 7–8).

Figure 7–8. TestBench Dialog Box
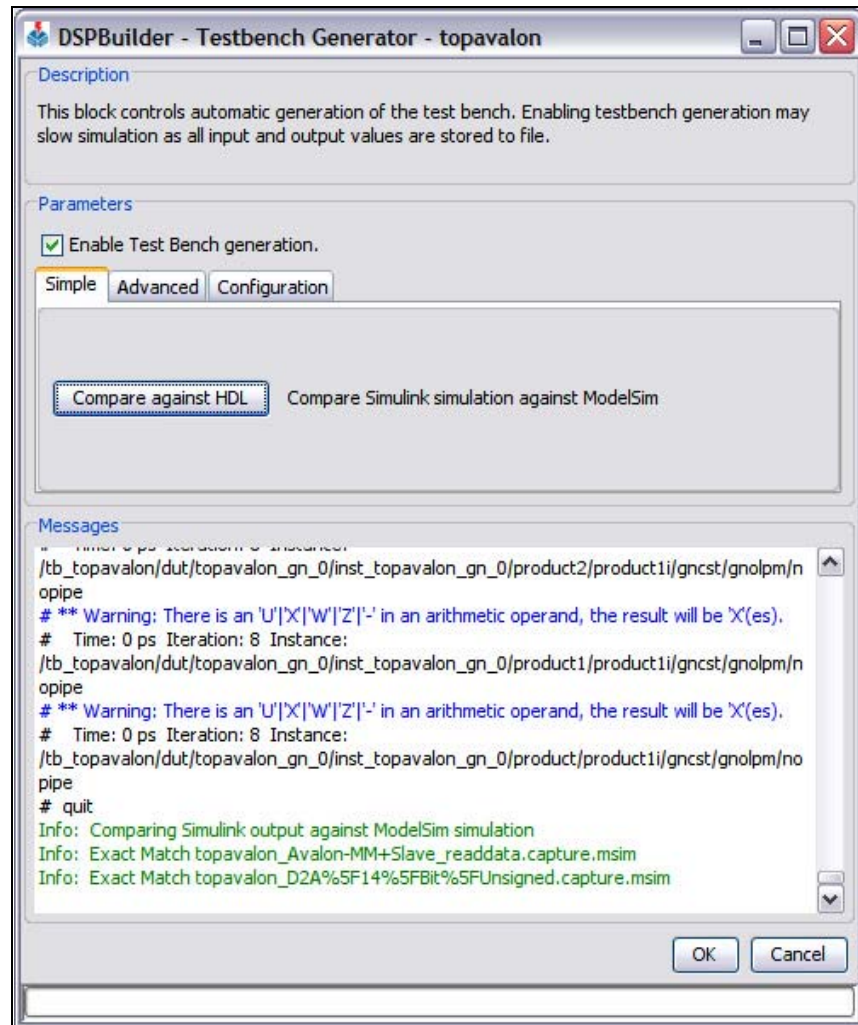
2. Click **Compare against HDL**.

   This process generates HDL, runs Simulink and ModelSim, and then compares the simulation results. Progress messages issue in the dialog box and completes with a message "`Exact Match`".

3. Click **OK**.

Running Signal Compiler

To generate all the hardware and files required by the Quartus Prime software, follow these steps:

1. Double-click on the `Signal Compiler` block to display the **Signal Compiler** dialog box.

2. Verify that the **Device** is set to match your target development kit and click **Compile**.

3. When the compilation has completed successfully, click **OK**.

Instantiating the Design in Qsys

To instantiate your design as a custom peripheral to the Nios II embedded processor in Qsys, follow these steps:

1. Start the Quartus® II software.

2. On the File menu in the Quartus II software, click **New Project Wizard**.

   a. Specify the working directory for your project by browsing to **<*DSP Builder install path*>\DesignExamples\Tutorials\SOPCBuilder\SOPCBlock\ MySystem**.

   b. Specify a name for your project. This tutorial uses `SOPC` for the project name.

   ☞ The Quartus Prime software automatically specifies a top-level design entity that has the same name as the project. This tutorial assumes that the names are the same.

   c. Click **Next** in the **New Project Wizard**, until you get to the **Family and Device Settings** page. Verify that the selected device matches the FPGA on your DSP development board (if applicable).

   d. Click **Finish** to create the Quartus Prime project.

3. On the Tools menu, click **Tcl Scripts** and follow these steps:

   a. Select **topavalon_add.tcl** in the Project folder.

   b. Click **Run** to load your .**mdl** file and other required files into the Quartus Prime project.

4. On the Tools menu, click **Qsys** and save as `SOPC`.

5. On the **Libraries** tab in Qsys, set the following options:

   a. Expand **Memories and Memory Controllers**.

   b. Expand **On-Chip** and double-click **On-Chip Memory (RAM or ROM)**.

   c. Specify **30 KBytes** for the **Total Memory size**.

   d. Click **Finish** to add an on-chip RAM device to the system.

   e. Double-click **Nios II Processor** in the **System Contents** tab to display the MegaWizard interface.

   f. Set the reset and exception vectors to use **onchip_mem** and click **Finish** to add the processor to your system with all other parameters set to their default values.

   g. Expand **Peripherals** and **Debug and Performance**. Double-click on **System ID Peripheral** and click **Finish** to accept the default settings.

   h. Expand **Interface Protocols** and **Serial**. Double-click on **JTAG UART** and click **Finish** to accept the default settings.

   i. Expand **DSPBuilder Systems** and double-click the **topavalon_interface** module to include it in your Nios II system (Figure 7–9).

Figure 7–9. Including Your DSP Builder Design Module in Qsys
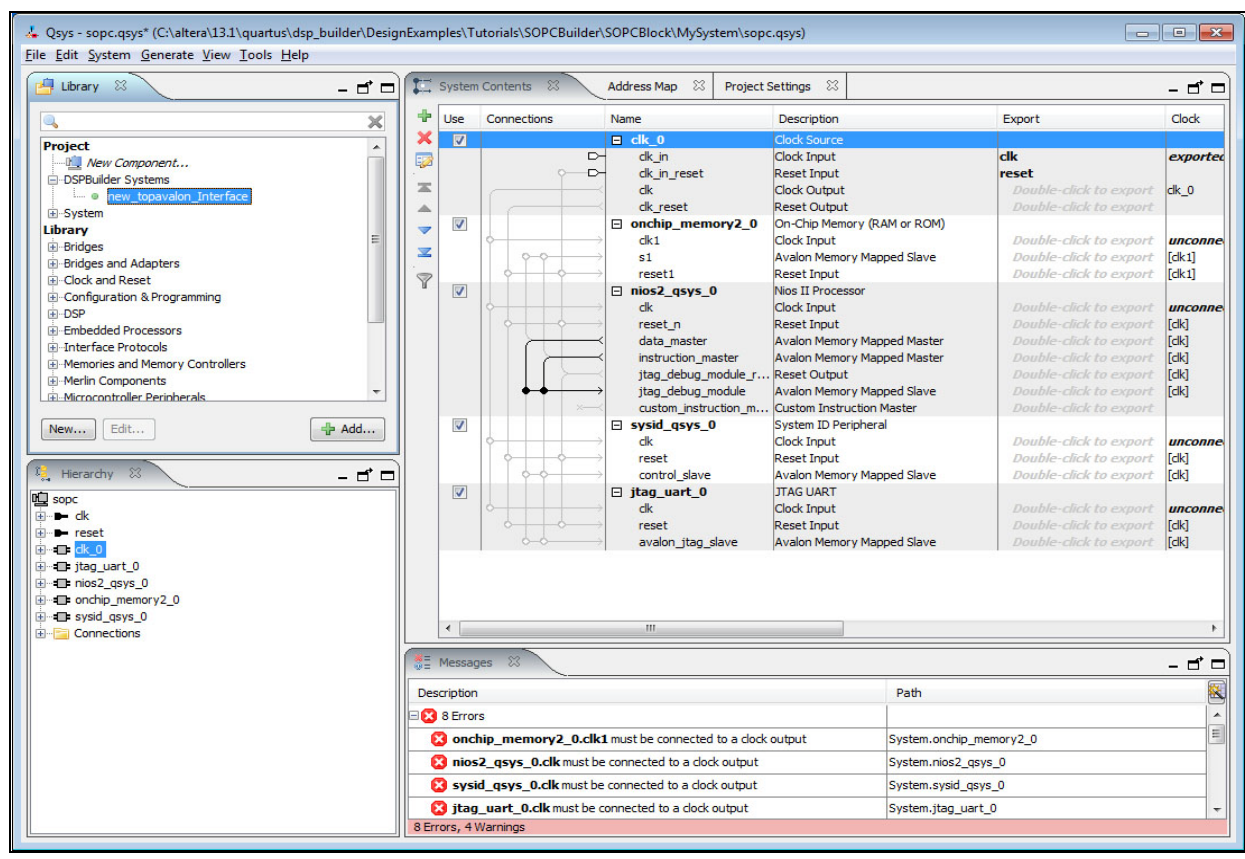
☞ If the memory device, Nios II processor, debug peripheral, interface protocol, and DSP Builder system add in this order, you should not need to set a base address. However, you can click **Auto-Assign Base Addresses** on the System menu to automatically add a base address if necessary.

6. Click **Generate** to generate the Qsys system. The system generation may take several minutes.

After the system generation in Qsys completes, you can design the rest of your Nios II embedded processor system using the standard Nios II embedded processor design flow. Continue with this tutorial to run the system from software using the Nios II processor.

Compiling the Quartus Prime Project

To compile the Quartus Prime project, follow these steps:

1. On the Assignments menu in the Quartus Prime software, click **Device** to display the Device page of the Settings dialog box and create the basic pin settings as follows:

   a. In the **Settings** dialog box, click **Device and Pin Options**.

   b. In the **Device and Pin Options** dialog box, click the **Unused Pins** tab, select **As input tri-stated** and click **OK**.

   c. Click **OK** to close the **Settings** dialog box.

2. On the Assignments menu, click **Pins** to open the **Pin Planner** and make pin assignments for clk and reset_n (depending on which development board you are using).

   ☞ If the **Location** column does not display, right-click in the pin assignments table and click **Customize Columns** to change the table display.

   You can ignore all other pin assignments for this tutorial.

3. Close the **Pin Planner**.

4. On the Processing menu, click **Start Compilation** to compile the Quartus Prime project.

5. When the compilation completes, click **Programmer** on the Tools menu and click **Start** in the **Quartus Prime Programmer** to program the FPGA device on your development board.

6. Close the **Quartus Prime Programmer** window.

Testing the DSP Builder Block from Software

Altera provides a C program that loads a set of four coefficient into the filter, reads them back, and then repeats the process. To use this program, follow these steps:

1. On the Nios II menu in Qsys, click **Nios II IDE**.

2. Create a new Nios II C/C++ application as follows:

   a. On the File menu in the Nios II IDE, point to **New** and click **Project**.

   b. **Nios II C/C++ Application**.

   c. In the **New Project** wizard, select **Nios II C/C++ Application** and click **Next**.

   d. Type `test_DSP_Block` for the **Name** of the project and select the **Blank Project** template.

   e. Turn on **Specify Location** and browse to the directory *<DSP Builder install path>***\DesignExamples\Tutorials\SOPCBuilder\SOPCBlock\MySystem**.

   f. Click on **Make a New Folder** and create a **software** subdirectory.

   g. Click **OK**.

   h. Browse to the System PTF file *<DSP Builder install path>***\DesignExamples\ Tutorials\SOPCBuilder\SOPCBlock\MySystem\SOPC.ptf**

   i. Click **Finish** in the **New Project** wizard.

   j. Verify that the application project `test_DSP_Block` appears in the **Nios II C/C++ Projects** list.

3. Add the test software to the new project as follows:

   a. Locate the file **test_DSP_Block.c** in your file system. (*<DSP Builder install path>***\DesignExamples\Tutorials\SOPCBuilder\SOPCBlock**)

   b. Right-click on the **test_DSP_Block.c** file and click **Copy**.

   c. Select the **test_DSP_Block** project folder in the **Nios II IDE** and paste the **test_DSP_Block.c** file into the project.

4. Set some of the reduced code footprint options in the Nios II IDE as follows:

   a. Right-click on the Nios II IDE application project, **test_DSP_Block**, and click **Properties**.

   b. In the **Properties** dialog box click **System Library**.

   c. Turn on **Reduced device drivers** and **Small C library**.

   d. Turn off **Support C++**.

   e. Click **OK**.

5. Run the test_DSP_Block software project in the Nios II IDE by right-clicking on **test_DSP_Block** and clicking **Run As Nios II Hardware**.

   The project compiles and the application code runs on the development board. Observe the following results in the **Nios II IDE Console**:

```
LOADING...

Coefficient 1 = 1

Coefficient 2 = 0

Coefficient 3 = 0

Coefficient 4 = 0

RELOADING...

Coefficient 1 = 0
```

```
Coefficient 2 = 0

Coefficient 3 = 1

Coefficient 4 = 0
```

☞ Completed versions of the **topavalon.mdl** design for the Stratix IV development boards are available in the **<***DSP Builder install path***>\DesignExamples\Tutorials\SOPCBuilder\SOPCBlock\Finished Examples** directory.

Avalon-MM FIFO Design Example

This tutorial describes how to interface a design built using the `Avalon-MM FIFO` block as a custom peripheral to the Nios® II embedded processor in Qsys.

The design consists of a Prewitt edge detector with one Avalon-MM write FIFO buffer and one Avalon-MM read FIFO buffer. The design uses an additional slave port as a control port.

For a full description of the Prewitt edge detector design, refer to *AN364: Edge Detection Reference Design*.

For this hardware implementation, DSP Builder stores the image in the compact flash and loads it in DMA with a Nios II embedded processor. DSP Builder outputs the edge detected image through a VGA controller. The DSP Builder model uses Simulink to read in the original image and to capture the edge detected result.

Opening the Design Example

To open the design example, follow these steps:

1. Click **Open** on the File menu in the MATLAB software.

2. Browse to the **<***DSP Builder install path***>\DesignExamples\Tutorials\ SOPCBuilder\AvalonFIFO** directory.

3. Select the **sopc_edge_detector.mdl** file and click **Open**.

Figure 7–10 shows **sopc_edge_detector.mdl**.

Figure 7–10. sopc_edge_detector.mdl Design Example
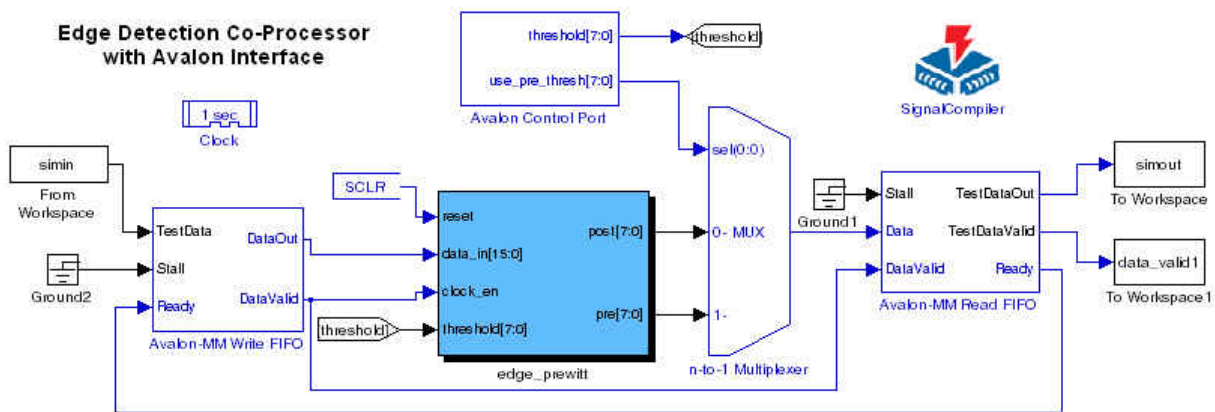
Compiling the Design

In this example, you use the `Signal Compiler` block to verify that your design generates valid HDL.

☞ Alternatively, use the `TestBench` block ("Avalon-MM Interface Blocks Design Example" in "Verifying the Design" on page 7–9).

To verify your design, follow these steps:

1. Double-click the `Signal Compiler` block.

2. Select the family and device for the DSP Development board you are using. The design example is configured for a Stratix 1S25 board (Figure 7–11).

3. Click **Compile**.

Figure 7–11. Signal Compiler Dialog Box
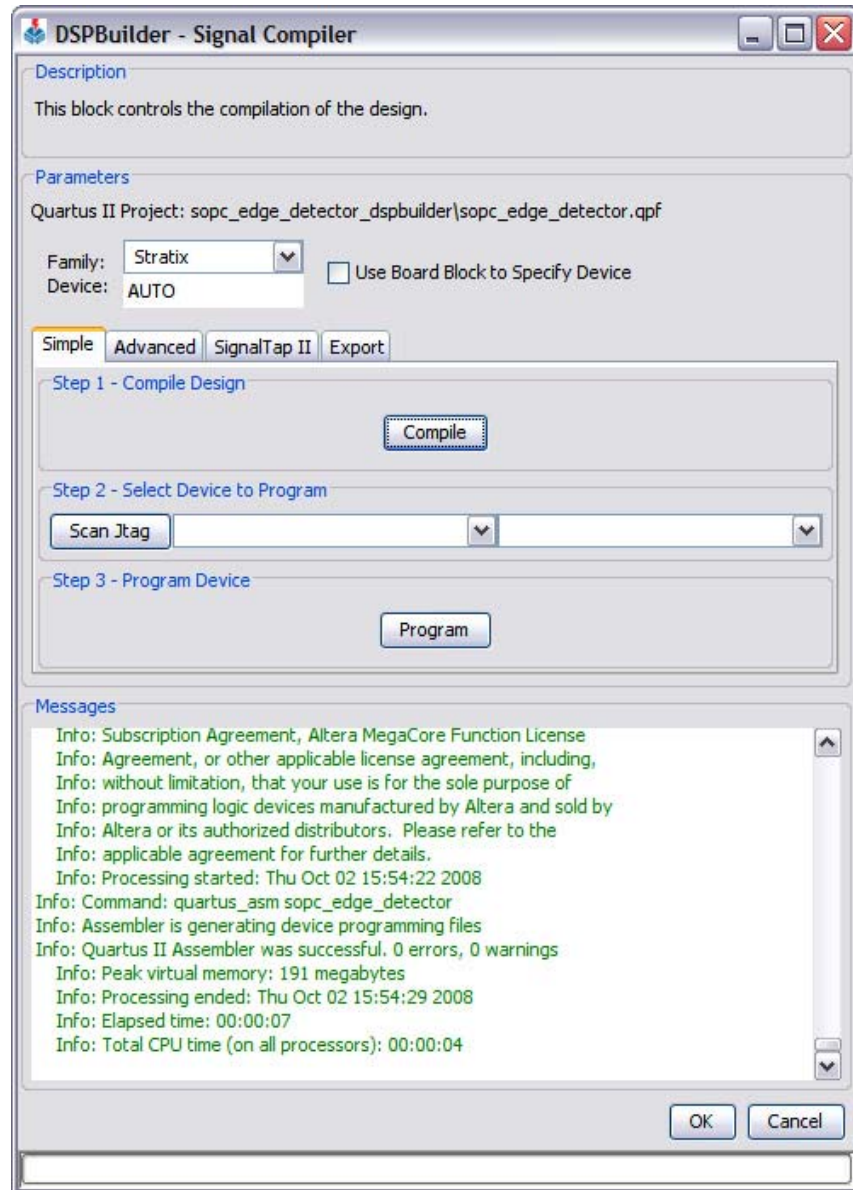
4. When the compilation completes successfully, click **OK**.

☞ The Avalon-MM read and write converter is simulation only and does not synthesize to HDL.

Instantiating the Design in Qsys

To instantiate your design as a custom peripheral to the Nios II embedded processor in Qsys, follow these steps:

1. Start the Quartus II software.

2. On the File menu in the Quartus II software, click **New Project Wizard** and set the following options:

    a. Specify the working directory for your project by browsing to **<*DSP Builder install path*>\DesignExamples\Tutorials\SOPCBuilder\AvalonFIFO**.

    b. Specify a name for your project. This tutorial uses FIFO for the project name.

    ☞ The Quartus Prime software automatically specifies a top-level design entity that has the same name as the project. This tutorial assumes that the names are the same.

    c. Click **Finish** to create the Quartus Prime project.

3. On the Tools menu, click **Tcl Scripts** and set the following options:

    a. Load your design by selecting **sopc_edge_detector_add.tcl** in the Project folder.

    b. Click **Run**.

4. On the Tools menu, click **Qsys** to display the **Create New System** dialog box.

    a. Specify AvalonFIFO as the system name.

    b. Select **VHDL** for the target HDL.

    c. Click **OK**.

5. Click the **System Contents** tab in Qsys and set the following options:

    a. Expand **Memories and Memory Controllers**.

    b. Expand **On-Chip** and double-click **On Chip Memory (RAM or ROM)**.

    c. Click **Finish** to add an on-chip RAM device with default parameters.

6. Double-click the **Nios II Processor** module in the **System Contents** tab to display the MegaWizard interface.

7. Set the reset and exception vectors to use **onchip_memory2_0** and click **Finish** to add the processor to your system with all other parameters set to their default values.

8. Expand **DSPBuilder Systems** in the **System Contents** tab and double-click the **sopc_edge_detector_interface** module to include it in your Nios II system.

You can now design the rest of your NIOS embedded processor with the standard Qsys design flow.

👣 For more detailed instructions, refer to "Instantiating the Design in Qsys" on page 7–10 in the "Avalon-MM Interface Blocks Design Example".

Avalon-ST Interface

All DSP MegaCore functions in the DSP Builder MegaCore Functions library have interfaces that comply with the *Avalon Interface Specifications*. You can combine multiple MegaCore functions easily because they use a common interface. This section summarizes the features of the Avalon-ST interface.

The *Avalon Interface Specifications* define how to convey data between a source interface and a sink interface. The interface indicates the integrity of the data by a feed forward signal, `valid`. The specification also defines how the MegaCore functions may stall other blocks (backpressure) or regulate the rate at which you provide data with a feedback sideband signal, `ready`.

You can configure the DSP Builder `Avalon-ST Source` and `Avalon-ST Sink` blocks with a ready latency of 0 or 1. The ready latency is the number of cycles that a source must wait after a sink asserts ready so that a data transfer is possible. The source interface provides valid data at the earliest time possible, and it holds that data until sink asserts ready. The `ready` signal notifies the source interface that it has sampled the data on that clock cycle.

For the *ready_latency* = 0 mode, Figure 7–12 shows the interaction that occurs between the source interface `valid` signal and the sink interface `ready` signal.

Figure 7–12. Avalon-ST Interface Timing for ready-latency=0
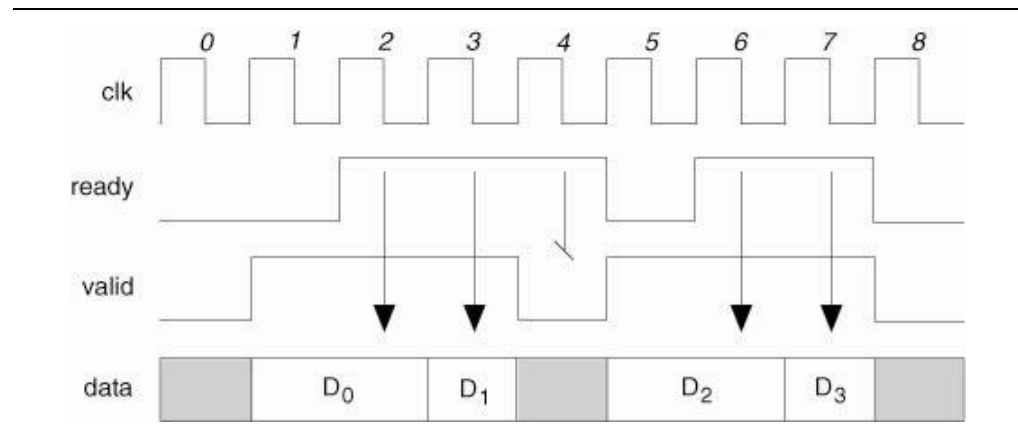
On cycle one, the source provides data and asserts `valid` even though the sink is not `ready`. The source waits until cycle two and the sink acknowledges that it samples the data by asserting `ready`. On cycle three, the source happens to provide data on the same cycle that the sink is ready to receive it and so the transfer occurs immediately. On the fourth cycle, the sink is ready but because the source does not provide any valid data, the data bus is not sampled.

A beat is the transfer of one unit of data between a source and sink interface. This unit of data may consist of one or more symbols, so it can support modules that convey more than one piece of information on each valid cycle. Some modules have parallel input interfaces and other instances require serial input interfaces. For example, when conveying an in-phase and quadrature component on the same clock cycle. The choice depends on the algorithm, optimization technique, and throughput requirements.

Figure 7–13 gives an example of a data transfer where two symbols are conveyed on each beat—an in phase symbol I and a quadrature symbol Q. In this example, each symbol is eight bits wide.

Figure 7–13. Packetized Data Transfer
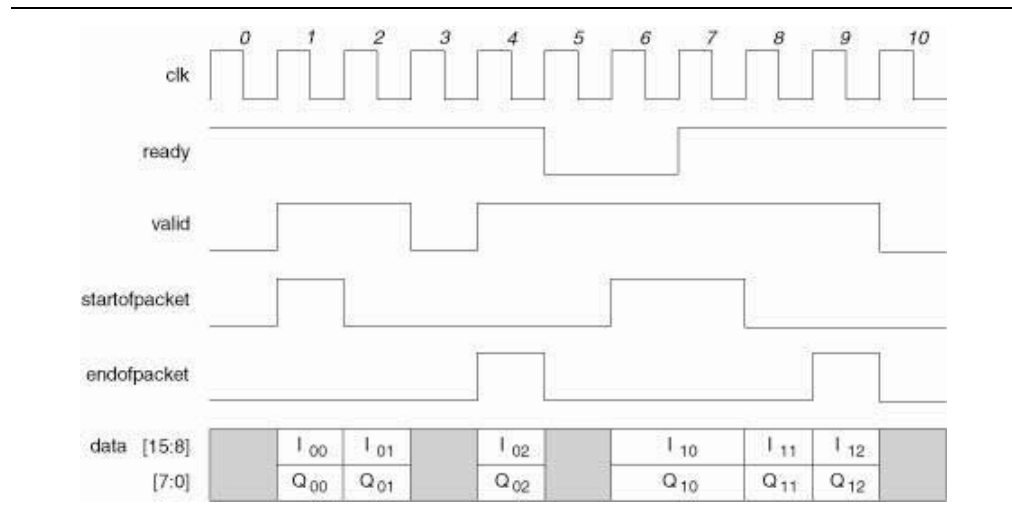
The *Avalon Interface Specifications* also describe several mechanisms to support the transfer of data associated with multiple channels. Altera recommends that you achieve this mechanism with packet based transfers where each packet has a deterministic format and each channel is allocated a specific field (time slot in a packet).

Packet transfers require two additional signals that mark the start and the end of the packet. The MegaCore functions have internal counters that count the samples in a packet so they know which channel a particular sample is associated with and synchronize appropriately with the start and end of packet signals. In Figure 7–13, the in phase and quadrature components associated with three different channels convey between two MegaCore functions.

Avalon-ST Packet Formats

You can allocate the data associated with each channel a field in a packet. To describe the relationship between the input and the output interfaces of a MegaCore function, you must define the packets associated with each interface.

Two parameters describe the basic format of a packet: `SymbolsPerBeat`, and `PacketDescription`. The `SymbolsPerBeat` parameter defines the number of symbols that DSP Builder presents in parallel on every valid cycle. The `PacketDescription` is a string description of the fields in the packet.

A basic `PacketDescription` is a comma-separated list of field names, where a field name starts with a letter and may include the characters `a-zA-Z0-9_`. Typical field names include `Channel1`, `Channel2`, and `Q`. Field names are case sensitive and white space is not permitted.

Figure 7–14 shows an example of a generic function that has two input interfaces and performs a transformation on the two input streams.

Figure 7–14. Generic Function
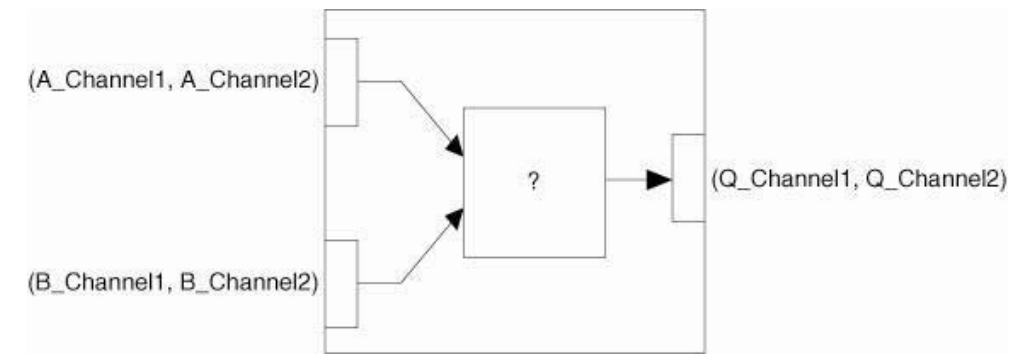
For a complete description of the Avalon-ST interface, refer to the *Avalon Interface Specifications*. For an example of a design that uses Avalon-ST interfaces and the Packet Format Converter blocks, refer to *AN442: Tool Flow for Design of Digital IF for Wireless Systems*.

Black Box Interfaces

The `Signal Compiler` block converts subsystems with blocks from the DSP Builder block libraries into HDL code. Non-DSP Builder blocks, such as encapsulations of your own pre-existing HDL code, require the `Signal Compiler` block to recognize them as black boxes so that the conversion process does not alter them.

There are two types of black-box interface in DSP Builder: implicit and explicit.

Implicit Black Box Interface

Use the `HDL Import` block to infer the implicit black-box interface.

The `Signal Compiler` block recognizes the `HDL Import` block as a black box and bypasses this block during the HDL translation.

For information about the `HDL Import` block, refer to the block description in the *AltLab Library* chapter of the *DSP Builder Standard Blockset Libraries* section in volume 2 of the *DSP Builder Handbook*.

Explicit Black-Box Interface

Use the `HDL Input`, `HDL Output`, `HDL Entity`, and `Subsystem Builder` blocks to specify the explicit black-box interface.

Using the `HDL Input`, `HDL Output`, and `HDL Entity` blocks prevents `Signal Compiler` from translating the subsystem into HDL. You can also use a `Subsystem Builder` block to create a new subsystem and then automatically populate its ports using the specified HDL.

Typically use the explicit black-box interface to encapsulate non-DSP Builder blocks from the main Simulink blocksets.

For information about the `HDL Input`, `HDL Output`, `HDL Entity`, and `Subsystem Builder` blocks, refer to the block descriptions in the *AltLab Library* chapter of the *DSP Builder Standard Blockset Libraries* section in volume 2 of the *DSP Builder Handbook*.

HDL Import Design Example

The `HDL Import` block provides an interface to import a HDL module into your DSP Builder design.

☞ To define imported VHDL use *std_logic_1164* types. If your design uses any other VHDL type definitions (such as arithmetic or numeric types), write a wrapper that converts them to *std_logic* or *std_logic_vector*.

The following sections show an example of importing an existing VHDL design into the DSP Builder environment with the `HDL Import` block.

Importing Existing HDL Files

To import existing HDL files into a DSP Builder design, follow these steps:

1. In MATLAB, change the current directory setting to: **<*DSP Builder install path*>\DesignExamples\Tutorials\BlackBox\HDLImport**

2. On the File menu, click **Open** and select **empty_MyFilter.mdl**.

   This design file has some of the peripheral blocks instantiated including the input and output ports and source blocks that provide appropriate stimulus for simulation. It is missing the main filter function, which you can import as HDL.

3. Rename the file by clicking **Save As** on the File menu. Name your new MDL file **MyFilter.mdl**.

4. Open the Simulink Library Browser. Expand the **Altera DSP Builder Blockset** and select the **AltLab** library.

5. Drag and drop a `HDL Import` block into your model.

6. Double-click on the `HDL Import` block to bring up the DSP Builder **HDL Import** dialog box (Figure 8–1 on page 8–3).

7. In the **HDL Import** dialog box, enable the **Import HDL** radio button and click on the **Add** button to select the HDL input files.

8. From the **VHDL Black Box File** dialog box, select the files **fir_vhdl.vhd**, **four_mult_add.vhd**, and **final_add.vhd,** then click on **Open**.

9. Ensure that `fir_vhdl` is specified as the name of the top-level design entity. The **fir_vhdl.vhd** file describes the top-level entity, which implements an 8-tap low-pass FIR filter design.

10. Turn on the option to **Sort top-level ports by name**.

11. Under **Generate Simulink Model**, click **Compile** to generate a Simulink simulation model for the imported HDL design.

Figure 8–1. HDL Import Dialog Box
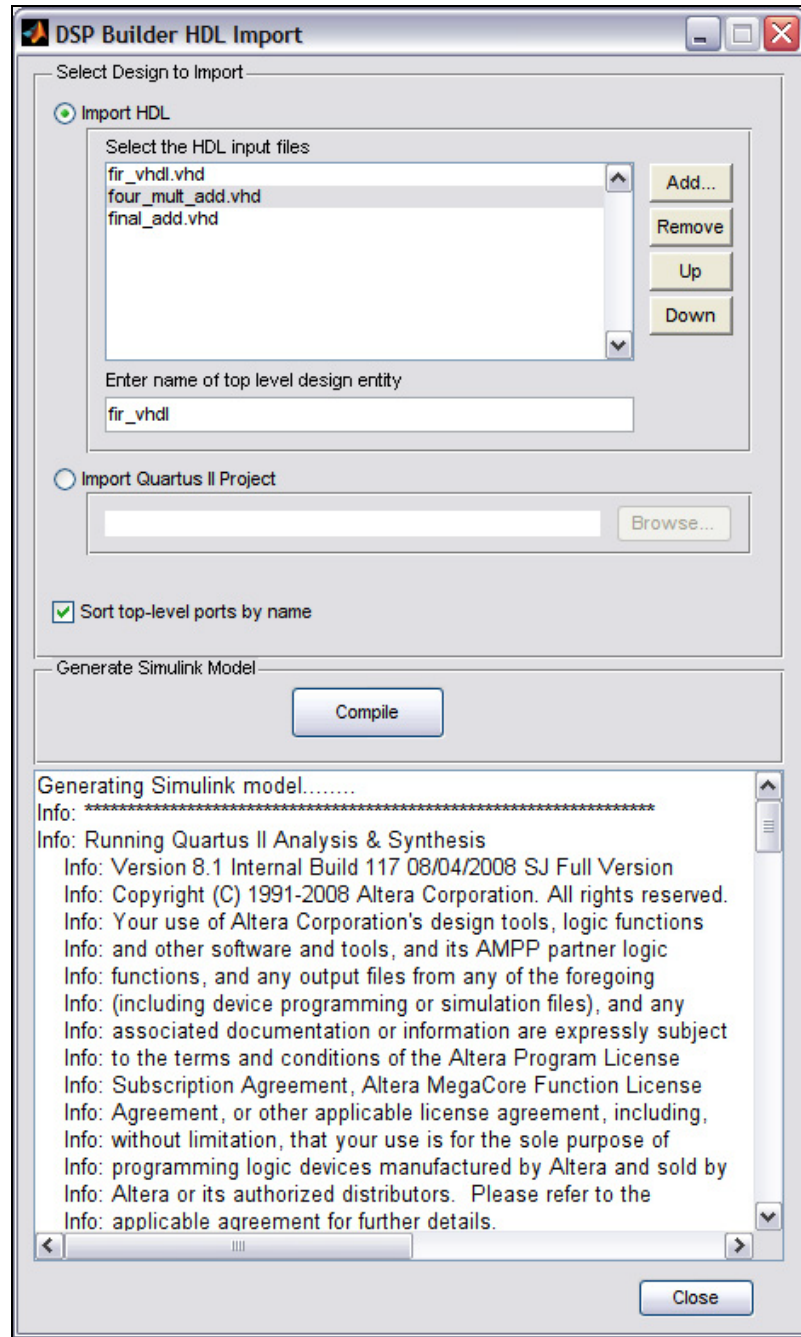
12. Progress messages issue in the **HDL Import** dialog box ending with the message:

    `Quartus Prime Analysis & Synthesis was successful.`

13. The `HDL Import` block in the **MyFilter.mdl** model updates to show the ports defined in the imported HDL.

14. Click **OK** to close the **HDL Import** dialog box.

15. Connect the input and output ports to the symbol (Figure 8–2). The code generated for the `HDL Import` block automatically converts to a black box.

Figure 8–2. Completed Design
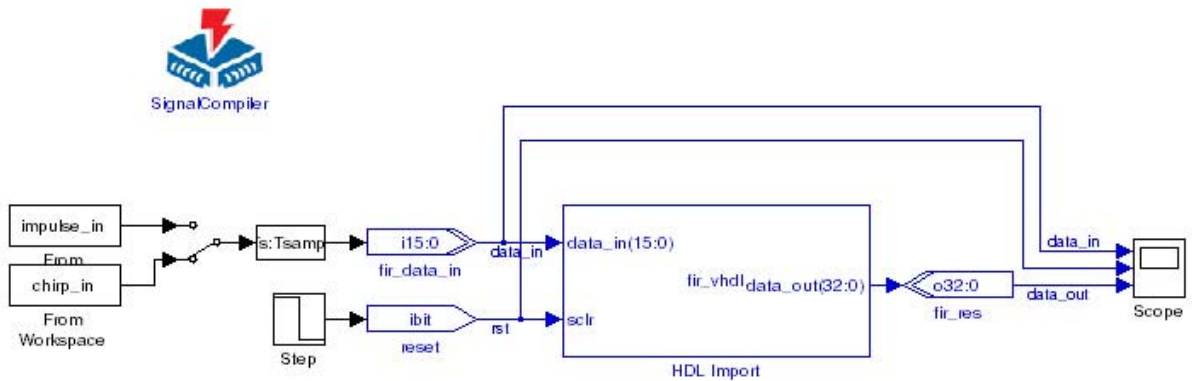
16. Click **Save** on the File menu to save the **MyFilter.mdl** file.

Simulating the HDL Import Model using Simulink

Follow these steps to run simulation in Simulink:

1. Double-click on the manual switch connected to the `Tsamp` block which feeds into the `fir_data_in` input port.

   This toggles the switch and sets the `impulse_in` stimulus, which verifies the impulse response of the low-pass filter.

2. Click **Start** on the Simulation menu in your model window.

3. Double-click on the `Scope` block to view the simulation results.

4. Click the **Autoscale** icon to resize the scope. This scales both axes to display all stored simulation data until the end of the simulation (which is set to 500*Tsamp for this model).

5. Click the **Zoom X-axis** icon and drag the cursor to zoom in on the first 70 X-axis time units.

   Figure 8–3 on page 8–5 shows the simulation results.

Figure 8–3. Simulink Simulation Results for the Impulse Stimulus
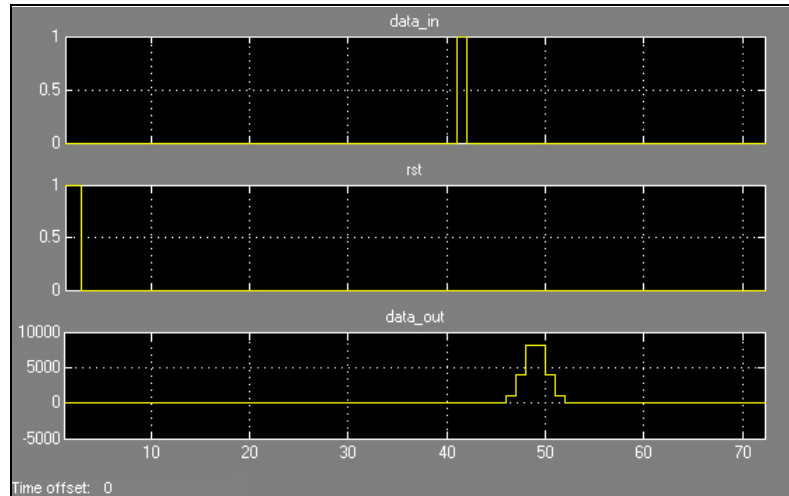
6. Double-click on the manual switch connected to the Tsamp block to select the **chirp_in** stimulus—a sinusoidal signal the frequency of which increases at a linear rate with time.

7. Click **Start** on the Simulation menu in your model window.

8. Double-click on the Scope block to view the simulation results.

9. Press the **Autoscale** icon to resize the scope.

   Figure 8–4 shows the simulation results.

Figure 8–4. Simulink Simulation Results for the Chirp Stimulus
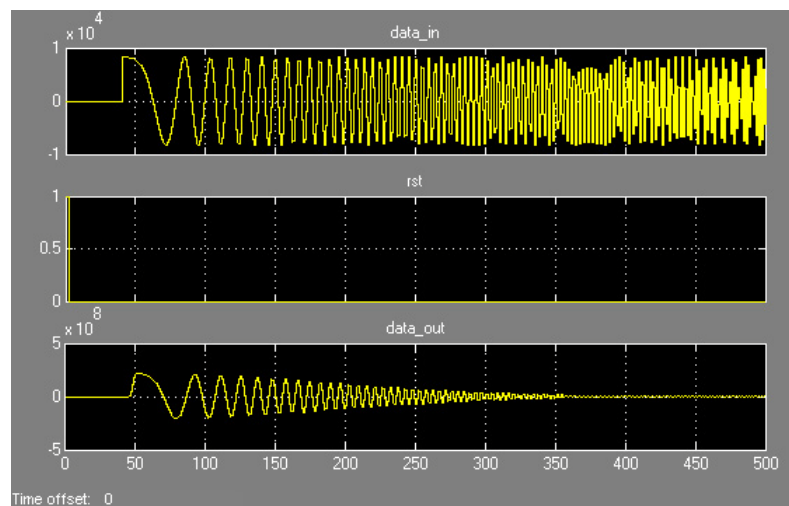
The HDL import tutorial is complete. You can optionally compile your model for synthesis or perform RTL simulation on your design by following similar procedures to those described in the "Getting Started".

Subsystem Builder Design Example

The `Subsystem Builder` block makes it easy for you to import the input and output signals for a VHDL or Verilog HDL design into a Simulink subsystem.

If your HDL design contains any LPM or megafunctions that the `HDL Import` block does not support, use the `Subsystem Builder` block. The `Subsystem Builder` block also allows you to create your own Simulink simulation model from non-DSP Builder blocks for faster simulation speed.

Unlike the `HDL Import` block, the `Subsystem Builder` block does not create a Simulink simulation model for the imported HDL design.

For more information about the `Subsystem Builder` block, refer to the block description in the *AltLab Library* chapter in the *DSP Builder Standard Blockset Libraries* section in volume 2 of the *DSP Builder Handbook*.

In addition to porting the HDL design to a Simulink subsystem, you must create the Simulink simulation model for the block. The simulation models describes the functionality of the particular HDL subsystem. The following options are available to create Simulink simulation models:

- Simulink generic library

- Simulink blocksets (such as the DSP and Communications blocksets)

- DSP Builder blockset

- MATLAB functions

- S-functions

☞ You must add a `Non-synthesizable Input` block and a `Non-synthesizable Output` block around any DSP Builder blocks in the subsystem.

The following section shows an example that uses an S-function to describe the simulation models of the HDL code.

Creating a Black Box System

To create a black-box system, follow these steps:

1. In MATLAB, change the current directory to: **<*DSP Builder install path*> \DesignExamples\Tutorials\BlackBox\SubSystemBuilder**

2. Click **Open** on the File menu. Select the **filter8tap.mdl** file and click **OK**.

3. Open the Simulink Library Browser and expand the **AltLab** library under the **Altera DSP Builder blockset**.

4. Drag a `Subsystem Builder` block into your model.

5. Double-click the `Subsystem Builder` block.

   The **Subsystem Builder** dialog box displays (Figure 8–5).

Figure 8–5. Subsystem Builder Dialog Box
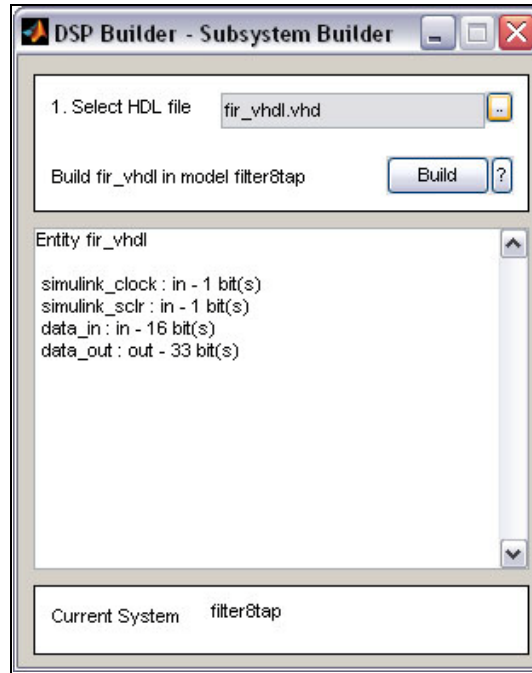
6. In the dialog box, browse for the **fir_vhdl.vhd** file and click **Build**.

   This action builds the subsystem and adds the signals for the `fir_vhdl` subsystem to the symbol in your **filter8tap.mdl** model. The **Subsystem Builder** dialog box automatically closes.

7. Connect the ports (Figure 8–6).

Figure 8–6. filter8tap Design
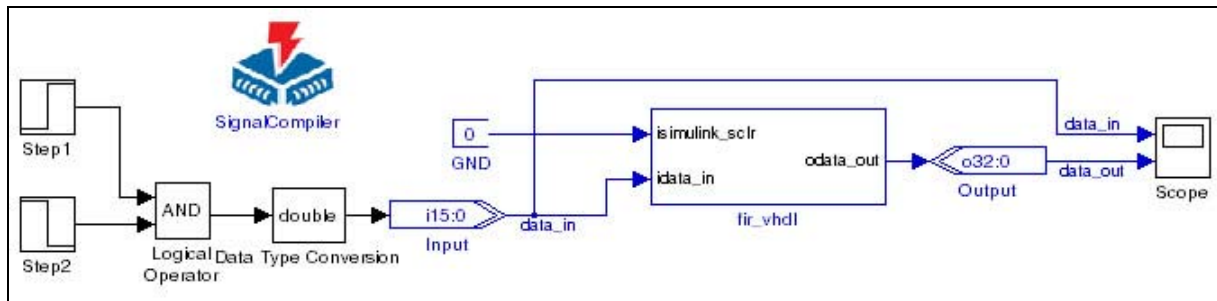
8. Double-click on the `fir_vhdl` symbol. The **filter8tap/fir_vhdl** subsystem opens (Figure 8–7).

Figure 8–7. Library: filter8tap/fir_vhdl Window
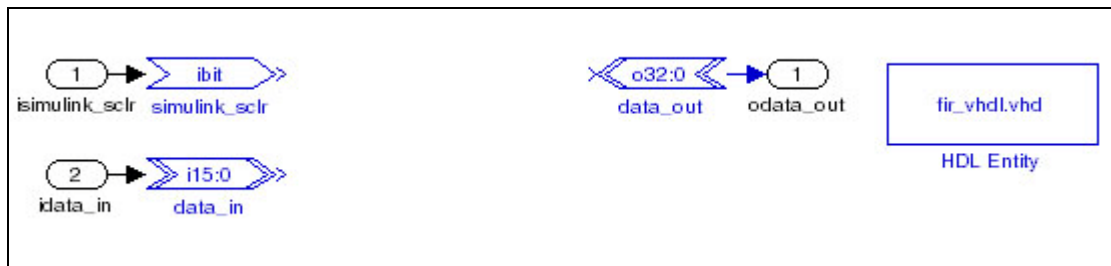
The subsystem contains two `HDL Input` blocks (`simulink_sclr` and `data_in`) and a `HDL Output` block (`data_out`). Each of these blocks in turn connects to a subsystem input or output. DSP Builder also creates a `HDL Entity` block to store the name of the HDL file and the names of the clock and reset ports.

☞ The clock is handled implicitly and no port is explicitly created in the subsystem.

9. Leave your model window open for use in the next section.

In the next section, you build the simulation model that represents the functionality of this block in your Simulink simulations.

Building the Black-Box SubSystem Simulation Model

For this example, you use a S-function C++ simulation model to represent the 8-tap FIR filter block. To create your model, follow these steps:

1. In the Simulink Library Browser, expand the **Simulink** folder.

2. From the **User-Defined Functions** library, drag and drop a `S-Function` block into your model window.

3. Double-click the `S-Function` block to display the **Function Block Parameters: S-Function** dialog box.

4. In the **Block Parameters** dialog box, change the **S-Function name** to `Sfir8tap` and enter the parameters `-1 3962 4817 5420 5733 5733 5420 4817 3962`.

The `Sfir8tap` function is a C++ Simulink S-Function simulation model for the 8-tap Fir filter block.

The first parameter refers to the sampling rate (-1 indicates it inherits the sampling rate from the preceding block) and the rest of the parameters represent the eight filter coefficients.

☞ Leave the **S-function modules** parameter with its default value.

5. Click the **Edit** button to view the code that describes the S-Function.

☞ If the code does not appear automatically, click **Browse** and select the **Sfir8tap.CPP** file.

6. Scroll down in the **Sfir8tap.CPP** file to the S-function methods section.

The following code shows the Simulink C++ S-Mex function code that designs a Simulink filter simulation model:

```
/*===================*
 * S-function methods *
 *===================*/
/* Function: mdlInitializeSizes======================================
 * Abstract:
 * The sizes information is used by Simulink to determine the S-function
 * block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    /* See sfuntmpl.doc for more details on the macros below */
    ssSetNumSFcnParams(S, 9);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
    /* Return if number of expected != number of actual parameters */
        return;
    }
    // Set DialogParameters not tunable
    const int iMaxssGetSFcnParamsCount = ssGetSFcnParamsCount(S);
    for (int p=0;p<iMaxssGetSFcnParamsCount;p++)
    {
        ssSetSFcnParamTunable(S, p,  0);
    }
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 1);
    ssSetInputPortDataType(S,  0, SS_DOUBLE);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 1);
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumDWork(S, DYNAMICALLY_SIZED); // reserve element in the
    ssSetNumModes(S, 0);       // pointers vector to store a C++ object
```

```
        ssSetNumNonsampledZCs(S, 0);

        ssSetOptions(S, 0);

}
```

During simulation, Simulink invokes certain callback methods from the S-function. The callback methods are subfunctions that initialize, update discrete states, and calculate output. Table 8–1 shows the design example callback methods.

Table 8–1. S-Function Callback Methods

Callback Method	Description
mdlInitializeSizes	Specify the number of inputs, outputs, states, parameters, and other characteristics of the S-function.
mdlInitializeSampleTimes	Specify the sample rates at which this S-function operates.
mdlStart	Initialize the vectors of this S-function.
mdlOutputs	Compute the signals that this block emits.
mdlUpdate	Update the states of the block.
mdlTerminate	Perform any actions required at termination of simulation.

1. At the MATLAB command prompt, type:

   mex Sfir8tap.CPP ↵

   The mex command compiles and links the source file into a shared library executable in MATLAB, **Sfir8tap.mexw32**. The extension is specific to 32-bit version of MATLAB run in Windows.

2. Close the editor window and click on **OK** to close the **Function Block Parameters** dialog box.

3. In the **filter8tap/fir_vhdl** window, connect the input port of the S-function block to the data_in block, and connect the output port of the **S-function** block to the data_out block (Figure 8–8).

Figure 8–8. S-Function Block Connection
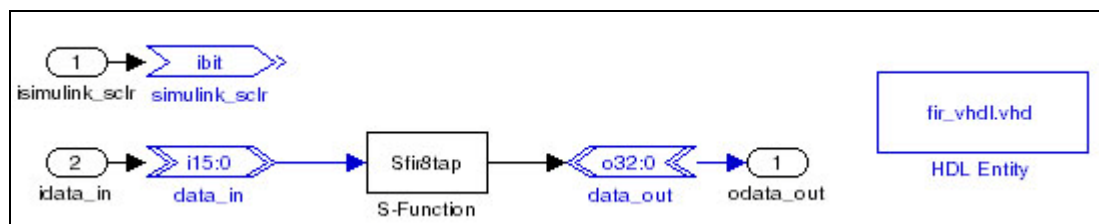
📝 You do not need to connect the simulink_sclr block. The HDL Entity block automatically maps any input ports named simulink_clock in the VHDL entity to the global clock signal, and any input ports named simulink_sclr to the global synchronous clear signal.

4. Click **Save** on the File menu to save the **filter8tap.mdl** file.

Simulating the Subsystem Builder Model

To run the Simulink simulation, follow these steps:

1. Click **Start** on the Simulation menu in the **filter8tap.mdl** window to begin the simulation.

2. Double-click the `Scope` block to view the simulation results. Click **Autoscale** to resize the scope.

3. Click the **Zoom X-axis** icon and use the cursor to zoom in on the first 22 x-axis time units.

   Figure 8–9 shows the simulation results.

Figure 8–9. Simulink Simulation Results of 8-Tap FIR Filter, Scope Window
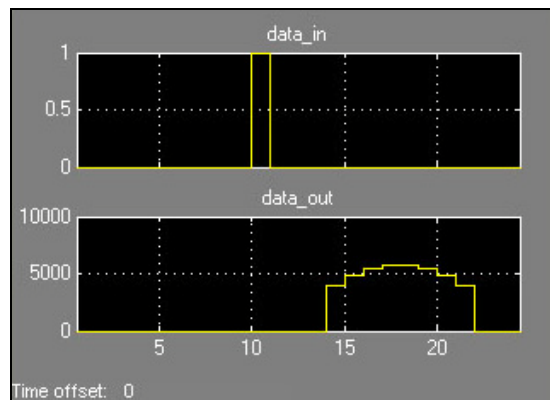
☞ Because the input is a pulse, the simulation results show the impulse response of the 8-tap FIR filter, which translates to the eight coefficient values. You can change the input stimulus to verify the step and random response of the filter.

Adding VHDL Dependencies to the Quartus Prime Project and ModelSim

The VHDL file is dependent on two other VHDL files. The Quartus Prime software or ModelSim do not examine these two files, and compilation either fails or gives unexpected results. To resolve this issue, follow these steps:

1. Double-click on the `Signal Compiler` block and click **Compile**. Ignore the result for now. This action creates a **DSPBuilder_filter8tap_import** directory in the directory containing your design.

   ☞ Alternatively, you can create the directory **DSPBuilder_filter8tap_import** directly.

2. Copy the **extra_add.tcl** and **extra_add_msim.tcl** files from the original design directory to the **DSPBuilder_filter8tap_import** directory.

The **extra_add.tcl** file adds **final_add.vhd** and **four_mult_add.vhd** to the Quartus Prime project, while **extra_add_msim.tcl** compiles them in ModelSim when your design is run using the `TestBench` block. The Quartus Prime software executes any files ending with **_add.tcl** when it creates the project. ModelSim executes files ending with **_add_msim.tcl** when it compiles your design testbench.

Simulate the Design in ModelSim

To test the simulation model against the HDL in ModelSim, follow these steps:

1. In the Simulink Library Browser, expand **AltLab** library under **Altera DSP Builder Blockset**.

2. Drag a `TestBench` block into your model.

3. Double-click on the `TestBench` block and click **Compare against HDL**.

   When the comparison completes successfully an `Exact Match` message issues in the **TestBench Generator** dialog box.

☞ If you want to use ModelSim directly, click on the **Advanced** tab, turn on the **Launch GUI** option, and then click **Run ModelSim**.

This completes the Subsystem Builder tutorial. You can optionally compile your model for synthesis ("Getting Started").

A parameterizable custom library block is a Simulink subsystem in which DSP Builder primitives describe the block functionality. This design flow also supports parameterizable hierarchical subsystem structures.

Altera provides an example of a custom library block, **<*DSP Builder install path*>\ DesignExamples\Tutorials\BuildingCustomLibrary\top.mdl**. (Figure 9–1).

Figure 9–1. top.mdl Example
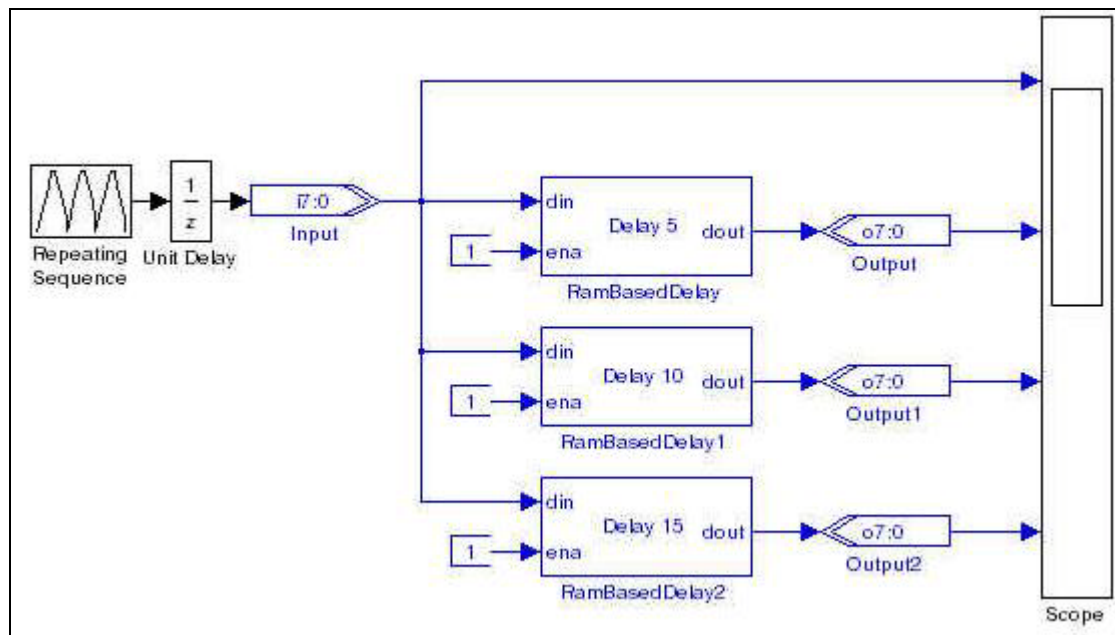
The `RamBasedDelay` block that **top.mdl** uses, is an example of a custom parameterizable Simulink block. The library file **MyLib.mdl** defines it. The `RamBasedDelay` block has one parameter, **Delay**.

Creating a Custom Library Block

To create your own custom block, follow these steps:

1. Creating a Library Model File

2. Building the HDL Subsystem Functionality

3. Defining Parameters Using the Mask Editor

4. Linking the Mask Parameters to the Block Parameters

5. Making the Library Block Read Only

6. Adding the Library to the Simulink Library Browser

Creating a Library Model File

To create a Library Model File for your custom block, follow these steps:

1. In MATLAB, change the current directory setting to: **<***DSP Builder install path***>\DesignExamples\Tutorials\BuildingCustomLibrary**.

2. Open the Simulink Library Browser. On the File menu in the Simulink Library Browser, point to **New** and click **Library** to open a new library model window.

3. Expand the Simulink **Ports & Subsystems** library in the Simulink Library Browser and drag a `Subsystem` block into your model.

4. Click on the `Subsystem` text below the block and rename the block `DelayFIFO`.

☞ You should always rename a block representing an HDL Subsystem to ensure that all the generated entities in a hierarchical design are unique.

5. Click **Save** on the File menu and save the library file as **NewLib.mdl**.

Building the HDL Subsystem Functionality

To add functionality to the `DelayFIFO` block, follow these steps:

1. Double-click on the `DelayFIFO` block to open the **NewLib/DelayFIFO** subsystem window.

2. Drag and drop a `Shift Taps` block from the **Storage** library in the **Altera DSP Builder Blockset** into your model window. Insert the `Shift Taps` block between the input and output blocks (Figure 9–2).

Figure 9–2. Shift Taps Block

3. Double-click the `Shift Taps` block to open the **Block Parameters** dialog box. Table 9–1 shows the parameters to set.

Table 9–1. Parameters for the Shift Taps Block (Part 1 of 2)

Parameter	Value
Main Tab	
Number Of Taps	1
Distance Between Taps	10
Optional Ports and Settings Tab	
Use Shift Out Port	Off
Use Enable port:	On

Table 9–1. Parameters for the Shift Taps Block (Part 2 of 2)

Parameter	Value
Use Dedicated Circuitry	On
Memory Block Type	Auto

4. Click **OK** to close the **Block Parameters** dialog box.

5. Add an Input block (In2) from the Simulink **Ports & Subsystems** library and connect it to the ena port on the Shift Taps block.

6. Rename the blocks (Table 9–2).

Table 9–2. Renaming the Blocks

Old Name	New Name
In1	InDin
In2	InEna
Shift Taps	DRB
Out1	OutDout

7. Click **Save** on the File menu.

Figure 9–3 shows the completed **DelayFIFO** subsystem.

Figure 9–3. DelayFIFO Subsystem
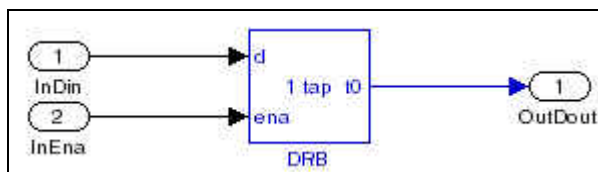
Figure 9–4 shows the **NewLib** library model that now shows the input and output ports defined in the **DelayFIFO** subsystem.

Figure 9–4. NewLib Model

Defining Parameters Using the Mask Editor

Use the Mask Editor to create parameters for the DelayFIFO block by following these steps:

1. Right-click the DelayFIFO block in the **NewLib** model and click **Mask Subsystem** on the pop-up menu.

2. In the **Mask Editor** dialog box set the parameters (Table 9–3 on page 9–4).

Table 9–3. Parameters for the Mask Editor

Parameter	Value
Icon Tab	
Frame	Visible
Transparency	Opaque
Rotation	Fixed
Units	Autoscale

Drawing Commands	port_label('input',1,'din');
	port_label('input',2,'ena');
	port_label('output',1,'dout');
	fprintf('Delay %d',d)

Parameters Tab	
Prompt	Delay
Variable	d
Documentation Tab	
Mask type	SubSystem AlteraBlockSet
Mask description	RAM-Based Delay Element Altera Corporation

3. Click **OK** in the **Mask Editor** dialog box.

4. Double-click on the `DelayFIFO` block in your **NewLib** model to display the **Block Parameters** dialog box.

5. Specify a **Delay** of 5.

6. Click **OK** in the **Block Parameters** dialog box.

7. Click **Save** on the File menu to save your library model.

For more information about the Mask Editor, refer to the MATLAB Help.

Linking the Mask Parameters to the Block Parameters

To pass parameters from the symbol's mask into the block, use a model workspace variable, by following these steps:

1. Double-click the `DRB` block in the **NewLib/DelayFIFO** window to open the **Block Parameters** dialog box.

2. Copy the mask parameter variable name *d* from the **Parameters** tab of the Mask Editor into the **Distance Between Taps** field in the **Block Parameters** dialog box.

3. Click **OK** to close the Shift Taps Block Parameters dialog box.

4. Close your model window.

Making the Library Block Read Only

Make a library block read only so that you do not accidentally edit it in a design model. To set the read and write permissions, follow these steps:

1. Right-click the DelayFIFO block in the NewLib model and click **SubSystem Parameters** on the pop-up menu to display the **Block Parameters** dialog box.

2. In the **Read/Write permissions** list, select **ReadOnly**.

   ☞ The **ReadWrite** option allows edits from both the library and the design. The **NoReadOrWrite** option does not allow Signal Compiler to generate HDL for the design. If you want to modify a library model, open your model, click **Unlock Library** on the File menu and change the read and write permissions in the **Block Parameters** dialog box. Remember to reset **ReadOnly** after changing the library model. Your changes are automatically propagated to all instances in your design.

3. Click **OK** to close the **Block Parameters** dialog box.

4. Click **Save** on the File menu to save your library model.

Adding the Library to the Simulink Library Browser

You can add a custom library to the Simulink library browser by creating a **slblocks.m** file. This file must be in the same location as your library file and both files must be in search path for MATLAB. To create this file, follow these steps:

1. On the File menu in MATLAB, point to **New** and click **M-File** to open a new editor window.

2. Enter the following text in the editor window:

```
function blkStruct = slblocks

blkStruct.Name = ['Custom Library DSP Builder'];
blkStruct.OpenFcn = 'NewLib';
blkStruct.MaskDisplay = '';
% Define the Browser structure array, the first
% element contains the information for the Simulink
% block library and the second for the Simulink
% Extras block library.
Browser(1).Library = 'NewLib';
Browser(1).Name = 'Custom Library DSP Builder';
Browser(1).IsFlat = 0;
blkStruct.Browser = Browser;

% End of slblocks
```

3. Save the M-file with the file name **slblocks.m** in the same directory as **NewLib.mdl**. The next time that you display the Simulink library browser the custom library is available (Figure 9–5).

Figure 9–5. Custom Library in the Simulink Library Browser
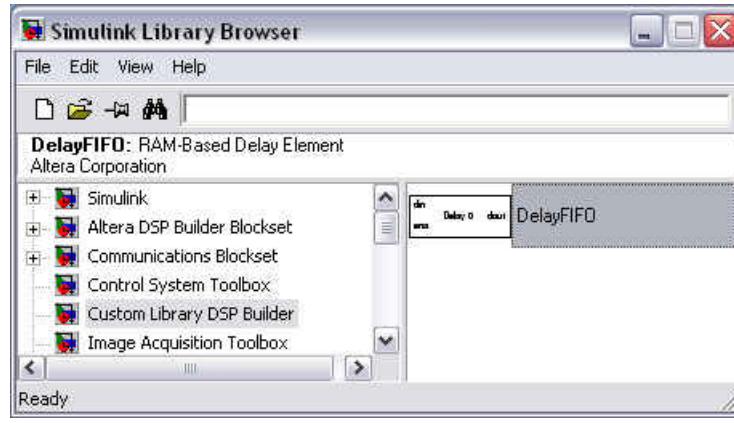
You can drag and drop a block from your custom library in the same way as from any other library in the Simulink library browser.

You can create a custom library with multiple blocks by creating the required blocks in the same library file.

For more information about M-files, refer to the MATLAB Help. A template **slblocks.m** file with explanatory comments is at <*MATLAB install path*>**\toolbox\ simulink\blocks\slblocks.m**.

Synchronizing a Custom Library

A custom library can contain MegaCore functions, HDL import, or state machine editor blocks. If you move or copy your design, synchronize your model containing these blocks by using the following command:

```
alt_dspbuilder_refresh_user_library_blocks
```

☞ This command calls automatically when you use either of the commands:

```
alt_dspbuilder_refresh_hdlimport

or

alt_dspbuilder_refresh_megacore
```

This chapter describes how to create and build a custom board library to use inside DSP Builder using built-in board components.

Each board library is defined by a XML board description file. This board description file contains all the board components and their FPGA pin assignments.

Creating a New Board Description

To add additional boards create new board description files. You only need to create a board description file for each new board and run a MATLAB command to build it into a DSP Builder Library. You can use the existing components or create your own custom components.

Predefined Components

Predefined components are in the following folder:

> *<install dir>***quartus****dsp_builder****lib****boardsupport****components**

There is a single XML file, *<component_name>***.component,** that describes each separate board component. This file defines its data type, direction, bus width, and appearance. The file also contains a brief description of the component.

Component Types

There are three main types of component: single bit, fixed size bus, and selectable single bit.

Single Bit Type

These components have a single bit with one FPGA pin assigned to each component. The components are either inputs or outputs and you cannot change them. DSP Builder offers the following single-bit predefined components:

■ Red and green LEDs (LED0 to LED17 and LEDG0 to LEDG8)

■ Software switches (SW0 to SW17)

■ User push buttons (PB0 to PB3)

■ Reset push buttons (IO_DEV_CLRn and USER_RESETN)

■ RS232 receive output and RS232 transmit input pins (RS232Rout and RS232Tin)

Fixed-Size Bus Type

These components have a fixed-sized group of same type (either input or output) pins with one FPGA pin assigned to each bit of the bus. DSP Builder offers the following fixed-size bus type predefined components:

■ 12-bit analog-to-digital converter (A2D1Bit12 and A2D2Bit12)

■ 14-bit analog-to-digital converter (A2D1Bit14 and A2D2Bit14)

■ 14-bit digital-to-analog converter (`D2A1` and `D2A2`)

■ 8-bit dual in-line package switch (`DipSwitch`)

■ 7-Segment display with a decimal point (`SevenSegmentDisplay0` to `SevenSegmentDisplay1`)

■ Simple 7-Segment display without a decimal point (`Simple7SegmentDisplay0` to `Simple7SegmentDisplay7`)

Selectable Single Bit Type

These components have a single bit, you can select the pin from a group of predefined FPGA pins. Furthermore, the pin can be set as either input or output. DSP Builder offers the following selectable single-bit predefined components:

■ Debug pins (`DebugA` and `DebugB`)

■ Prototyping pins (`PROTO`, `PROTO1` to `PROTO3`)

■ Evaluation input pin (`EvalIoIn`)

■ Evaluation output pin (`EvalIoOut`)

Component Description File

To define a new component create a corresponding component file, *<component_name>*.**component,** in the same folder as the predefined components.

The component description file contains a root element `component` that contains several attributes and subelements that define the component. The `component` has the following attributes:

■ `displayname=` Specifies the name of the component, which the board description file references.

■ `direction=` Specifies the direction of the signal. It can have the value of `Input` or `Output`. You can omit this attribute for the Selectable Single Bit Type, because it is set later.

■ `type=` Specifies the data type of the signal. The type can be `BIT`, `INT`, or `UINT`. followed by the size in square brackets. For example, `"BIT[1,0]"` defines a single bit while `"UINT[12,0]"` is a 12-bit unsigned integer.

The component subelements have the following definitions:

■ `<documentation>` *text* `</documentation>` This subelement contains text describing the component and one of the following variable that define how the pin name, or list of pin-names appears in the new board library:

    ■ `%pinname%` for single bit type

    ■ `%pinlist%` for selectable single bit type

    ■ `%indexedpinliat%` for fixed size bus type

■ `<display [attributes]>` This subelement has the following attributes:

    ■ `icon=` specifies the image file name for the component

    ■ `width=` specifies the display width for the image file

    ■ `height=` specifies the display height for the image file

☞ For components without an image, you can omit the `icon` display attribute and define a visual representation using the `plot` and `fprintf` commands. For example:

```
<display width="90" height="26">
plot([0 19 20 21 22 21 20 19], [0 0 1 0 0 0 -1 0]);
fprintf('EVAL IO OUT \n%pinname% ');
</display>
```

Example Component Description File

The following code shows an example of a component description file:

```
<component displayname="EVAL IO OUT" direction="Output"
type="BIT[1,0]">
    <documentation>
Prototyping Area Pin Single Bit Output
%pinlist%
    </documentation>
    <display width="90" height="26">
        plot([0 19 20 21 22 21 20 19],[0 0 1 0 0 0 -1 0]);
        fprintf('EVAL IO OUT \n%pinname% ');
    </display>
</component>
```

Board Description File

Create the board description file, *<board_name>*.**board,** in the following folder:

<install dir>**quartus****dsp_builder****lib****boardsupport****boards**

The board description file has header and board description sections.

Header Section

This section contains the following line that defines the XML version and character encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

In this case, the document conforms to the 1.0 specification of XML and uses the ISO-8859-1 (Latin-1/West European) character set.

You should not modify this line.

Board Description Section

The main body of the document is a root element `board` that has several attributes and subelements that define the details of the board.

```
<board Attributes>
    <displayname> Text </displayname>
    <component Attributes />
    ...........
```

```
<component Attributes />
<configuration Attributes>
    <devices> Attributes>
    </devices>
    <option Attributes>
    </option>
</configuration>
</board>
```

☞ The last line in the file must be a closing tag for the root element board </board>.

The board attributes have the following definitions:

■ uniquename= A unique name to reference the board.

■ family= Device family of the FPGA on board (assuming only one device is on the board).

The board must contain a displayname subelement containing text that describes the board. For example:

```
<displayname>Cyclone II XYZ Board</displayname>
```

The following component subelements declare the components:

■ Single bit type examples:

```
<component name="LED0" pin="Pin_E5"/>
<component name="LED1" pin="Pin_B3"/>
```

where attribute name defines the name of the component on the board and pin defines the FPGA pin to which the component is connected. The name must match one of the predefined components and you can use it only once per board.

■ Fixed-size bus type example:

```
<component name="DipSwitch" label="S1">
    <pin location="Pin_AC13"/> <!-- LSB -->
    <pin location="Pin_A19"/>
    <pin location="Pin_C21"/>
    <pin location="Pin_C23"/>
    <pin location="Pin_AE18"/>
    <pin location="Pin_AE19"/> <!-- MSB-->
</component>
```

where attribute name defines the name of the component on the board and label defines the name of the component as it appears in Simulink. For a component with width *n*, there must be *n* pin subelements. The pin location must be a valid FPGA pin name. The pin ordering is listed from LSB to MSB, with LSB on top of the list.

■ Selectable single bit type example:

```
<component name="PROTO1">
    <pin location="Pin_C3"/>
```

```
            <pin location="Pin_D2"/>

            <pin location="Pin_L3"/>

            <pin location="Pin_J7"/>

            <pin location="Pin_J6"/>

            <pin location="Pin_K6"/>

    </component>
```

This element has the same format as the fixed-size bus type, but you can choose each pin element from a specified list of available FPGA pin locations.

The `configuration` element defines the board configuration block. For example:

```
<configuration icon="dspboard2c35.bmp" width="166" height="144">

    <devices jtag-code="0x020B40DD">

        <device name="EP2C35F672C6" />

    </devices>

<!-- Input clock selection list -->

    <option name="ClockPinIn" label="Clock Pin In">

        <pin location="Pin_N2"/>

        <pin location="Pin_N25"/>

        <pin location="Pin_AE14"/>

        <pin location="Pin_AF14"/>

        <pin location="None"/>

    </option>

<!-- Global Reset Pin -->

    <option name="GlobalResetPin" label="Global Reset Pin">

        <pin location="Pin_A14"/>

        <pin location="Pin_AC18"/>

        <pin location="Pin_AE16"/>

        <pin location="Pin_AE22"/>

        <pin location="None"/>

    </option>

</configuration>
```

The configuration attributes have the following definitions:

- `icon` = the image file to be used for the board configuration block

- `width` = the width of the image

- `height` = the height of the image

The `devices` subelement has the following attributes:

- `jtag-code` = the JTAG code of the FPGA device

- `device name` = the device name of the FPGA used on the board

Each `option` subelement has the following attributes:

- `name` = the name of the option (clock or reset pin)

■ `label` = labels that identifies the pins on the blocks

■ `pin location` = a list of selectable clock or reset pins

For more examples, refer to any of the existing board description files.

Building the Board Library

To build the board library:

1. Start DSP Builder in MATLAB, on Windows OS, click on **Start**, point to **All Programs**, click **Altera <*version*>**, click **DSP Builder**, and click **Start in MATLAB version *XX***.

2. Run the following command in the MATLAB command window board library:

   ```
   alt_dspbuilder_createComponentLibrary
   ```

This chapter describes how to implement a state machine in DSP Builder.

☞ Use the `State Machine Editor` block in new designs.

The design example, **fifo_control_logic.mdl**, contains a simple state machine to implement the control logic for a first-in first-out (FIFO) memory structure.

The design files for this example are in the *<DSP Builder install path>*\ **DesignExamples\Tutorials\StateMachine** directory.

Figure 11–1 shows the top-level schematic for the FIFO design example.

Figure 11–1. FIFO Design Example Top-Level Schematic
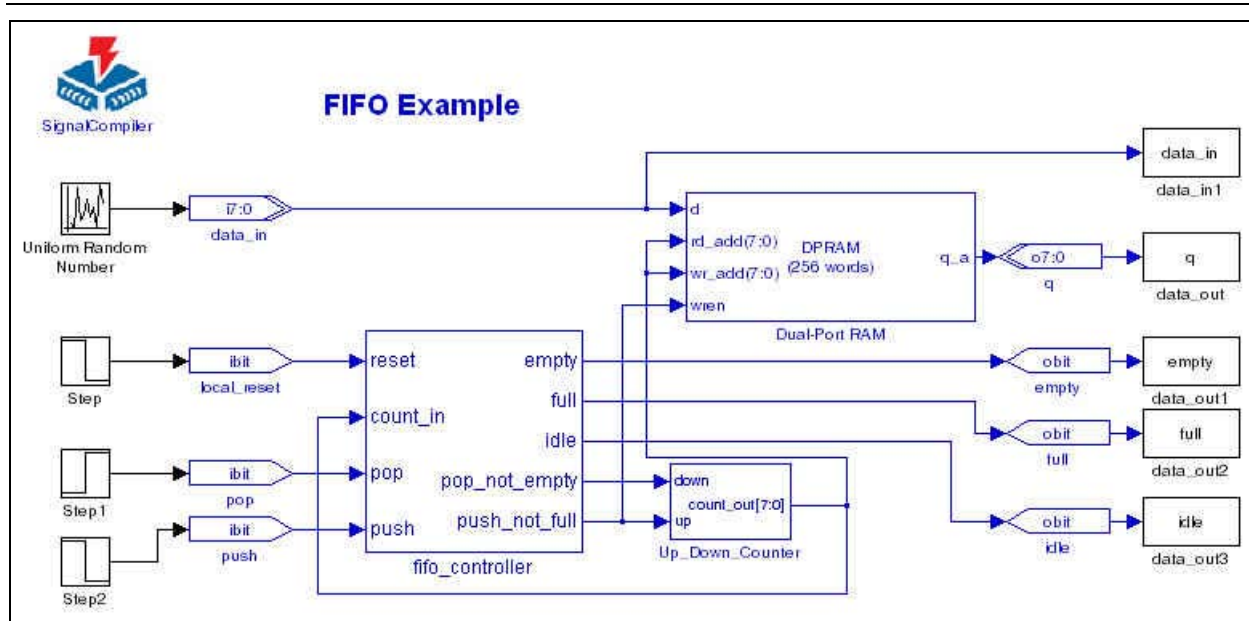
The state machine in this design example feeds the control inputs of a `Dual-Port RAM` block and the inputs of an address counter.

The state machine has the following operation:

■ When you assert the `push` input and the address counter is less than 250, the address counter increments and a byte of data writes to memory.

■ When you assert the `pop` input and the address counter is greater than 0, the address counter decrements and a byte of data reads from memory.

■ When the address counter is equal to 0, the `empty` flag asserts

■ When the address counter is equal to 250, the `full` flag asserts.

Using the State Machine Editor Block

To use the `State Machine Editor` block to create the FIFO controller in the design example, follow these steps:

1. Add a `State Machine Editor` block to your Simulink design and assign it a new name. Figure 11–2 shows the default `State Machine Editor` block. In this example, the block is named `fifo_controller`.

Figure 11–2. fifo_controller State Machine Editor Block

> ☞ You should save you model and change the default name of the `State Machine Editor` block before you define the state machine properties.

2. Double-click the `fifo_controller` block to open the State Machine Editor in the Quartus II software (Figure 11–3).

Figure 11–3. Quartus II State Machine Editor Window
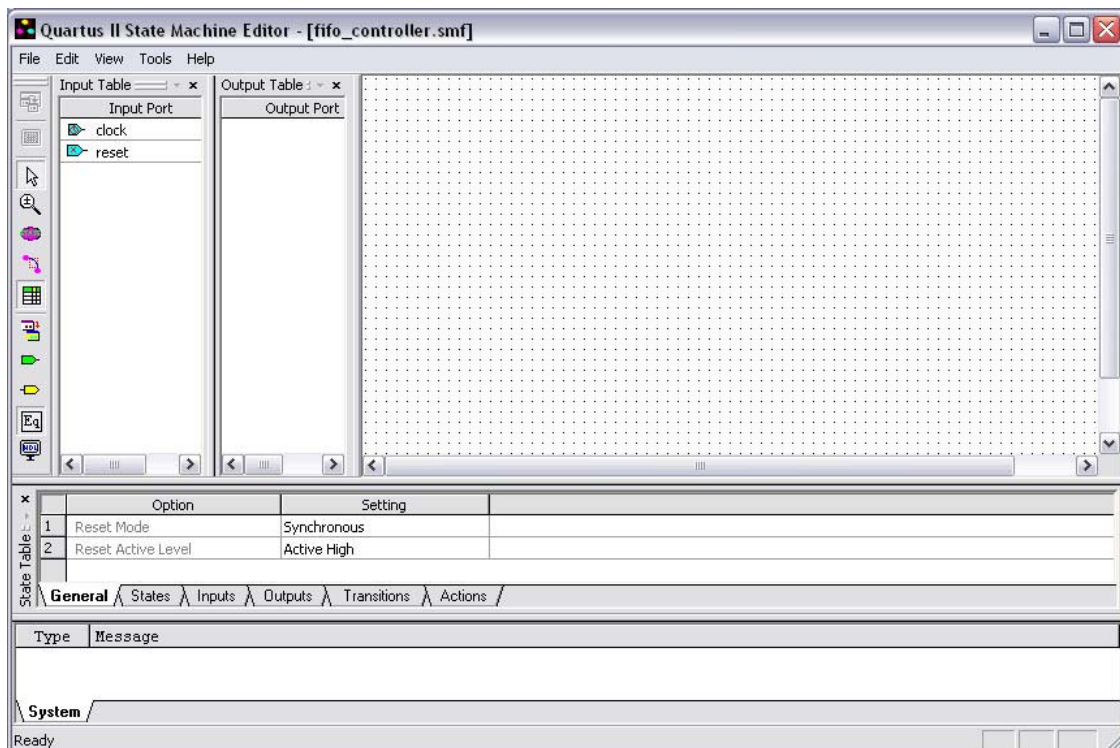
3. On the Tools menu in the Quartus II State Machine Editor, point to **State Machine Wizard** and click **Create a new state machine design**.

4. The first page of the wizard allows you to choose the reset mode, whether the reset is active-high or active-low, and whether the outputs are registered. Accept the default values (synchronous, active-high, registered outputs) and click **Next** to display the **Transitions** page of the wizard.

5. Delete the default state names (state1, state2, state3) and type the following new state names:

   ■ empty

   ■ full

   ■ idle

   ■ pop_not_empty

   ■ push_not_full

6. Delete the default input port names (input1, input2) and type the following new input port names:

   ■ count_in[7:0]

   ■ pop

   ■ push

   ☞ Do not change the clock and reset port names. The count_in port must be defined as an 8-bit vector to allow count values up to 250.

7. Edit the state transitions by entering the statements (Table 11–1).

Table 11–1. FIFO Controller Transitions

Source State	Destination State	Condition
empty	push_not_full	(push==1)&(count_in!=250)
empty	idle	(push==0)&(pop==0)
full	idle	(push==0)&(pop==0)
full	pop_not_empty	(pop==1)
idle	empty	(pop==1)&(count_in==0)
idle	push_not_full	(push==1)
idle	pop_not_empty	(pop==1)&(count_in!=0)
idle	full	(push==1)&(count_in==250)
pop_not_empty	idle	(push==0)&(pop==0)
pop_not_empty	empty	(pop==1)&(count_in==0)
pop_not_empty	push_not_full	(push==1)&(count_in!=250)
pop_not_empty	pop_not_empty	(pop==1)&(count_in!=0)
pop_not_empty	full	(push==1)&(count_in==250)
push_not_full	idle	(push==0)&(pop==0)
push_not_full	empty	(pop==1)&(count_in==0)
push_not_full	push_not_full	(push==1)&(count_in!=250)
push_not_full	full	(push==1)&(count_in==250)
push_not_full	pop_not_empty	(pop==1)&(count_in!=0)

☞ The transitions are validated on entry and must conform with Verilog HDL syntax.

Figure 11–4 shows the **Transitions** page after you define the states, inputs, and transitions.

Figure 11–4. State Machine Editor Wizard Transitions Page
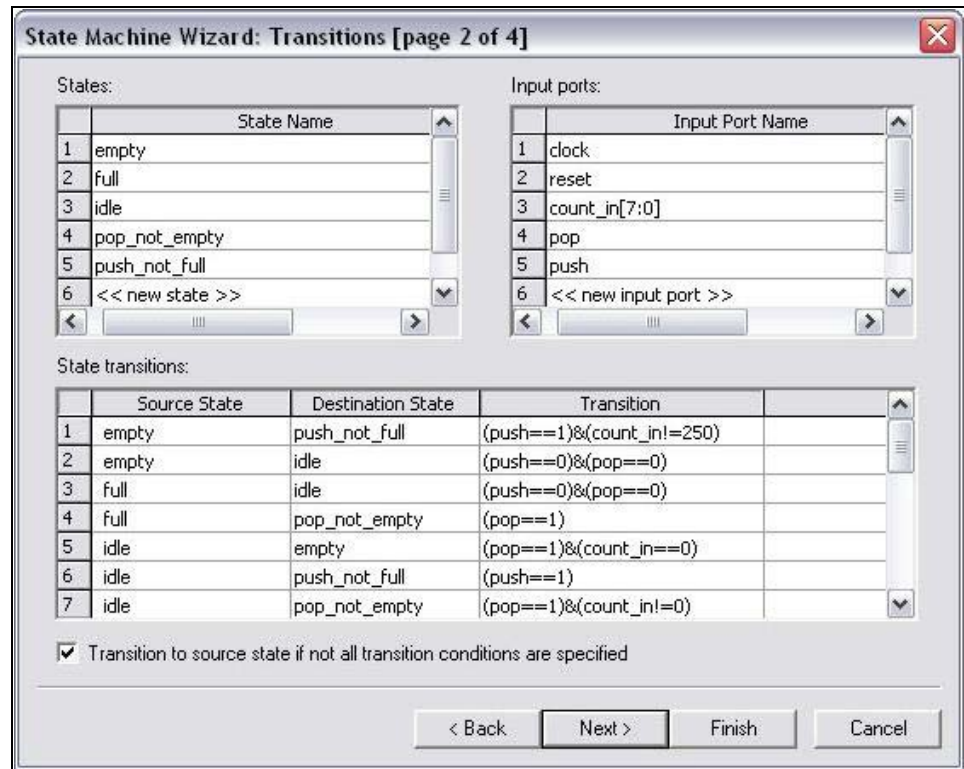
8. Click **Next** to display the **Actions** page. Delete the default output port name (`output1`) and enter the following new output port names:

■ `out_empty`

■ `out_full`

■ `out_idle`

■ `out_pop_not_empty`

■ `out_push_not_full`

9. Specify the output logic for each output port by specifying the action conditions to set each output port to 1 when the state is true and 0 for all other states (Table 11–2).

Table 11–2. FIFO Controller Output Actions

Output Port	Output Value	In State
out_empty	1	empty
out_full	1	full
out_idle	1	idle
out_pop_not_empty	1	pop_not_empty
out_push_not_full	1	push_not_full
out_empty	0	full
out_empty	0	idle
out_empty	0	pop_not_empty
out_empty	0	push_not_full
out_full	0	empty
out_full	0	idle
out_full	0	pop_not_empty
out_full	0	push_not_full
out_idle	0	empty
out_idle	0	full
out_idle	0	pop_not_empty
out_idle	0	push_not_full
out_pop_not_empty	0	empty
out_pop_not_empty	0	full
out_pop_not_empty	0	idle
out_pop_not_empty	0	push_not_full
out_push_not_full	0	empty
out_push_not_full	0	full
out_push_not_full	0	idle
out_push_not_full	0	pop_not_empty

Figure 11–5 shows the **Actions** page after you define the output ports, and action.

Figure 11–5. State Machine Editor Wizard Actions Page
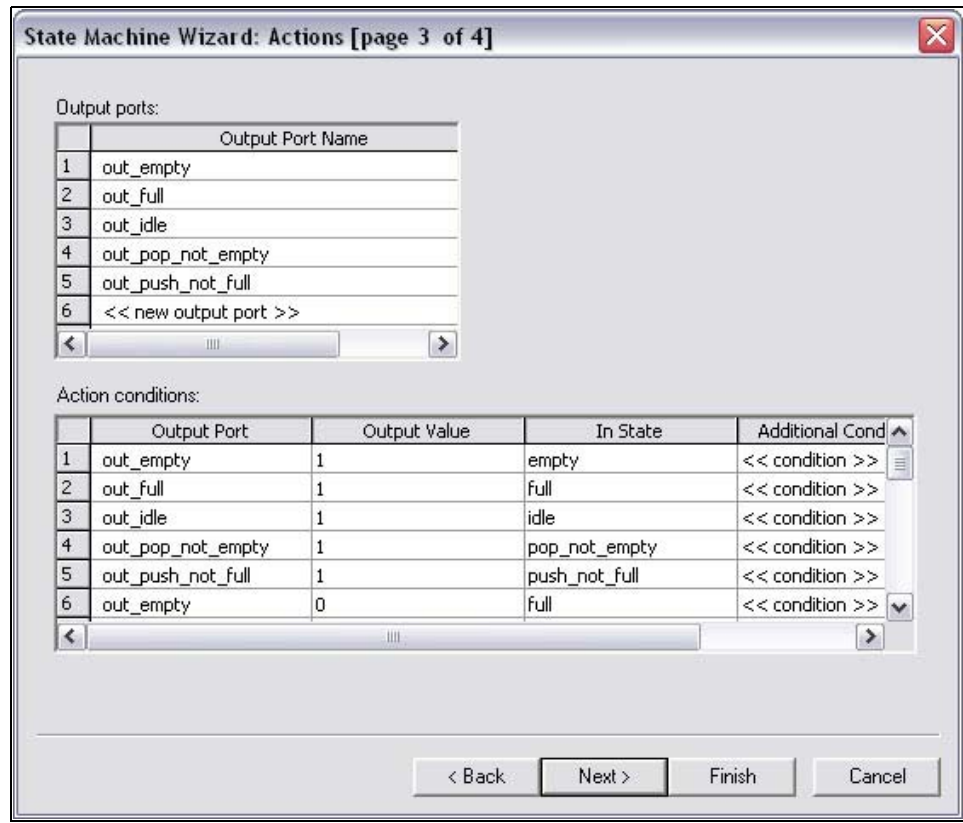
10. Click **Next** to display the **Summary** page. Check that the summary lists the five states (empty, full, idle, pop_not_empty, and push_not_full), the five input ports (clock, count_in[7:0], pop, push, and reset), and the five output ports (out_empty, out_full, out_idle, out_pop_not_full, and out_push_not_full).

11. Click **Finish** to complete the state machine definition. The state machine displays graphically in the State Editor window (Figure 11–6 on page 11–6).

Figure 11–6. Graphical fifo_controller State Machine Diagram
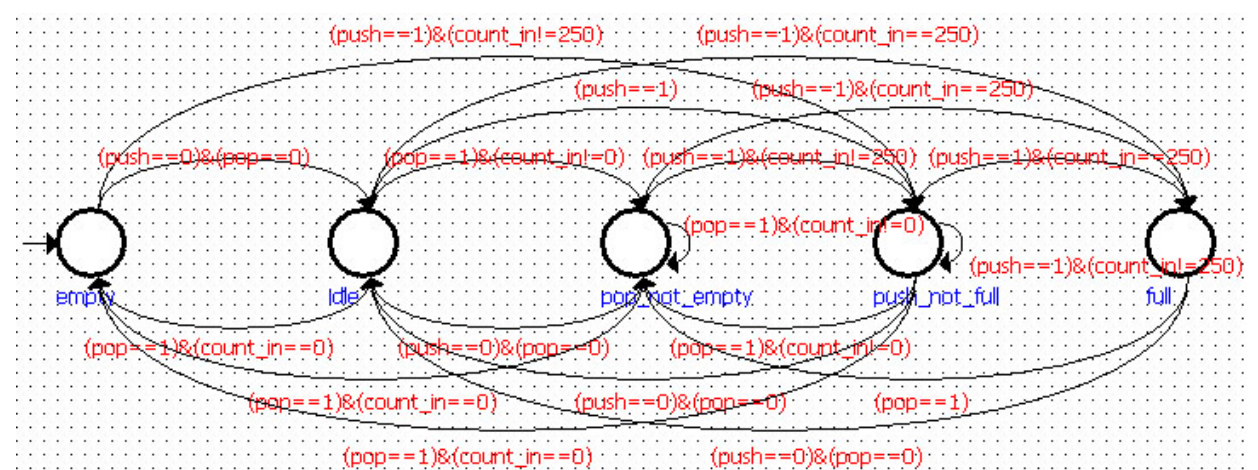
☞ DSP Builder marks the first state that you enter in the wizard as the default state. This state is the empty state and is the state to which the state machine transitions when you assert the reset input.

12. On the Tools menu in the Quartus II State Machine Editor, click **Generate HDL File** to display the **Generate HDL File** dialog box. Select **VHDL** and click **OK** to confirm your choice. Click **Yes** to save the **fifo_controller.smf** file and check that there are no FSM verification errors.

☞ There are five warning messages stating that FSM verification skips in each state. You can ignore these messages.

If there are any errors, you can edit the state machine using the **Properties** dialog boxes that you can display from the right button pop-up menu when you select a state or transition. You can also edit the state machine in table format by clicking the tabs at the bottom of the State Machine Editor window.

👣 For information about editing state machine properties and drawing a graphical state machine, refer to the *About the State Machine Editor* topic in the Quartus II Help.

13. On the File menu in the Quartus II State Machine Editor, click **Exit**.

The fifo_controller block on your model updates with the input and output ports defined in the state machine.

☞ You may need to resize the block to ensure that the input and state names do not overlap and are displayed correctly.

Figure 11–7 shows the updated fifo_controller block for the FIFO design example.

Figure 11–7. fifo_controller Block After Closing the State Machine Editor

DSP Builder design requires the following files to store all the components:

■ The top-level Simulink model **<*top_level_name*>.mdl**

■ The import directory **DSPBuilder_<*top_level_name*>_import** and its contents.

■ Any source files for imported HDL.

■ Any Intel format hexadecimal memory initialization (.**hex**) files.

■ Any referenced custom library files.

■ The analyzed Simulink model file **<*top_level_name*>.mdlxml**.

☞ When you include the .**mdlxml** file in a Quartus Prime project, you do not need to call MATLAB to synthesize the design. You can still synthesize a project without the .**mdlxml** file, but you must call MATLAB as part of the generation flow.

If you do not want Quartus Prime synthesis to call MATLAB, or are passing the design a user without access to MATLAB, follow one of these steps:

■ Include both the .**mdl** and corresponding .**mdlxml** files in the project,

■ Export HDL and specify the exported HDL as the source with no references to the .**mdl** or .**mdlxml** files in the project.

☞ Any design that includes HDL Import, State Machine Editor or MegaCore functions requires the import directory.

Integrating with Source Control Systems

Altera recommends that you store Quartus Prime archive (.**qar**) files rather than individual HDL files for source control purposes.

To create a .**qar** file, follow these steps in the Quartus Prime software:

1. Create a Quartus Prime project that sources the top-level Quartus Prime IP (.**qip**) file that the DSP Builder Export HDL flow generates ("About Exporting HDL" on page 12–3).

2. Perform analysis and elaboration to ensure the design incorporates any black-box system files.

3. Archive the project by clicking **Archive Project** on the project menu in the Quartus II software) to generate the .**qar** file.

☞ Any HDL elements that you introduce into DSP Builder with custom library blocks may require their own source control. Additional .**qip** files, which are referenced in the "# Imported IP files" section of the top-level .**qip** file, list the required files.

HDL Import

In general, source files that you import with HDL Import are not part of a DSP Builder project. DSP Builder references them in projects that generate with the Export HDL flow as external files, with absolute paths.

When you move a design to a new version of the tools or to a location on a different computer, run the `alt_dspbuilder_refresh_HDLimport` script to ensure the HDL Import blocks are up-to-date.

When migrating to a new computer, re-import the HDL to enable hardware generation (although simulation in Simulink may be possible without this step).

MegaCore Functions

The MegaCore IP Library always installs in the same parent directory as the Quartus Prime installation. This directory is not a subdirectory of the **quartus** directory but a relative path to an install directory at the same level as the **quartus** directory. The expected directory structure is:

> *<install_path>*<**QUARTUS_ROOTDIR**>**\..\ip**

This feature allows the Export HDL flow to use relative paths, and improves portability.

When moving a design to a new version of the tools or a different location, run the `alt_dspbuilder_refresh_megacore` script to ensure that the MegaCore function blocks are up-to-date.

Successful migration of designs with MegaCore Functions assumes that the new environment has all the required IP installed. It may be necessary to install the MegaCore IP Library and run the `alt_dspbuilder_setup_megacore` script.

Memory Initialization Files

Intel-format hexadecimal (.**hex**) files are required for memory initialization in simulation and hardware generation. If they are generated by HDL Import or MegaCore function blocks, ensure that they are in the import directory. This fact is generally not the case if you generate the files with HDL Import.

About Exporting HDL

You can export the DSP Builder-generated synthesizable HDL to a Quartus Prime project and then use the **Export** tab in the `Signal Compiler` block to export them (Figure 12–1).

Figure 12–1. Export Tab in Signal Compiler
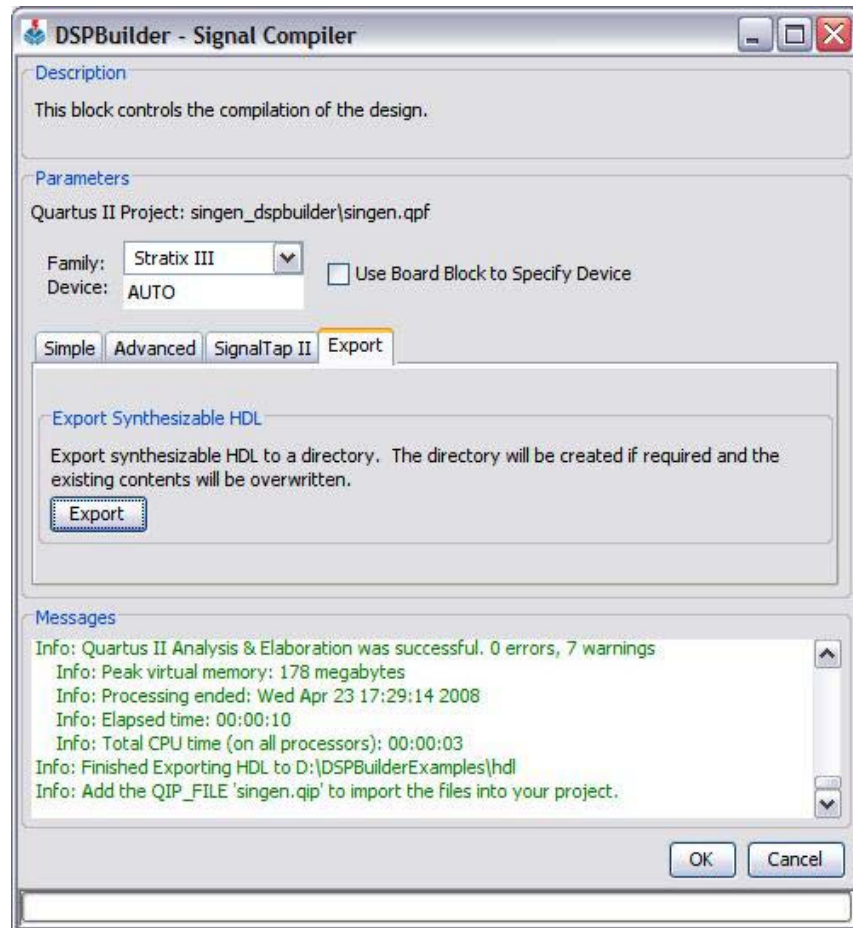
You can also export HDL by executing the `alt_dspbuilder_exportHDL` command in the MATLAB command window.

The syntax for the export HDL command is:

> *<exportDir_value>* `alt_dspbuilder_exportHDL(`*<model>***,** *<exportDir>*`)`

where:

- *model* is the name of the .**mdl** file to export. This file is always the top-level name in the exported Quartus Prime project.

- *exportDir* is the directory that contains the exported files. If you omit this optional argument, DSP Builder uses the default or previous export directory.

- *exportedDir_value* is the return string indicating the output directory containing the newly generated files.

Running this flow creates a set of source files in the export directory, including a .**qip** file corresponding to the top-level of your design.

Using Exported HDL

After the Export HDL flow completes:

1. Create a project using the **New Project Wizard**, which is available on the File menu in the Quartus Prime software.

2. Enter the top-level name of the exported project and add the corresponding .**qip** files as the single source file for the project. Additional .**qip** files may describe the requirements for black-box components. The top-level files sources these files automatically.

3. Use the **Archive Project** command in the Quartus Prime software to archive this project.

☞ When migrating designs that include MegaCore function blocks to a different location, edit their corresponding .**qip** files if they include absolute paths to library components.

Use the Export HDL flow on a Windows computer to a Linux-based computer to migrate the files generated. However, this act requires adding an additional file to the project. This additional **alt_dspbuilder_package.vhd** file is in the **<QUARTUS_ROOTDIR>\libraries\vhdl\altera** directory on a Windows computer.

Moving Standard Blockset Files to a New Location

When moving DSP Builder (standard blockset) projects to a new directory or machine, you can recreate the project by transferring a minimum set of design files. You do not need to copy the entire project directory. However, in some cases, when the relative paths for the design files change in the new location, recreate the auxiliary files to achieve a successful compilation.

You require the following minimum set of design files to recreate a project:

■ DSP Builder model (.**mdl**)

■ HDL source files associated with HDL import blocks (if any)—maintain same relative path to MDL

■ HDL wrapper files associated with IP MegaCore function blocks (if any)— maintain same relative path to the .**mdl** file (they should be in the **DSPBuilder_<**mdl name**>_import** subdirectory)

■ Memory initialization .**hex** files (if any)

■ Custom library files (if any)

To recreate the project in the new location, follow these steps:

1. If the model contains IP MegaCore function or HDL import blocks, regenerate the auxiliary files (.**qip**, .**entityimport**, .**simdb**) associated with the IP MegaCore function or HDL import block by following these steps:

   a. HDL import—run the **alt_dspbuilder_refresh_HDLimport** script to automatically update all the HDL import blocks in the new location.

   b. IP MegaCore functions—run the **alt_dspbuilder_refresh_megacore** script to automatically update all the IP MegaCore function blocks in the new location. Successful migration of design with IP MegaCore functions assumes that you install the required MegaCore IP library on the new environment.

   ☞ If the **..\DSPBuilder_<**mdl name**>_import** subdirectory copies and the design files, skip this step.

2. Re-analyze the model by clicking **Analyze** in the Advanced tab of the `Signal Compiler` block to regenerate the auxiliary files (.**mdlxml**, .**qip**, .**ipx**) associated with the model.

Multiple Models in a Top-Level Quartus Prime Project

To integrate multiple DSP Builder (standard blockset) designs in a top-level Quartus II project, you need the .**mdl** and .**ipx** files.

Use the Quartus Prime IP (.**qip**) file as the single source file for each DSP Builder model. The .**qip** file is a single file that contains paths for all the files for an IP design. The .**qip** file allows you to add an IP design to the project by adding only one file, rather than adding all the necessary files individually. You only need the .**qip** file for Quartus Prime archiving for DSP Builder, which does not use it for generation.

If the multiple .**mdl** files uses the same IP cores, the .**qip** references the IP libraries and the Quartus Prime software reports conflicts when resolving the libraries. You must edit the .**qip** to remove any redundant libraries.

If the DSP Builder design includes HDL import or IP MegaCore functions, the top-level .**qip** may reference embedded .**qip** file(s). Also, some older versions of IP MegaCore functions (before v8.0) and HDL import blocks may have absolute paths in the generated .**qip** files. If you migrate the files from a different location, it may be necessary to manually edit their corresponding .**qip** files to reflect the new environment. By running the **Analyze** process from the `Signal Compiler` block in the new location, the .**qip** file updates automatically with the new path settings. These embedded .**qip** file(s) contain the information concerning the projects, libraries and source HDL required by the Quartus Prime software for successful integration of these external entities into DSP Builder.

In addition to the .**qip** source files, the top-level project also requires an IP Index (.**ipx**) file that specifies additional paths to find components. Specifically for DSP Builder designs, the Quartus Prime software needs the .**ipx** file for the HDL import and IP MegaCore function blocks that your model uses. Essentially, the DSP Builder system is an entity composed of DSP Builder blocks, (which are entities but are easily discoverable), and non-native entities like HDL import and MegaCore functions.

☞ The DSP Builder specifies the main entities in the main Quartus Prime **.ipx** file and need no special action—you just need to add the extra HDL import and MegaCore function entities.

To update the IP library search path for the top-level Quartus Prime project, create an additional directory *<project directory>*/**ip/***<module name>* and create a file *<module name>*.**ipx** in that subdirectory.

The ipx File Contents

The **.ipx** file has the following contents:

```
<library>

    <path path='../../../<module name>/**/*'/>

</library>
```

These statements specify the relative path to the directory where to locate the **.mdl** file and where to search for directories containing further **.ipx** files. The ** means search recursively, and the final * locates all identifiable elements there.

You can combine all the search paths into a single **.ipx** file. For example:

```
<library>

<path path='../../../<module name1>/**/*'/>

<path path='../../../<module name2>/**/*'/>

...

</library>
```

You can also specify a path to a specific **.ipx** file using:

```
<index file='../../../blockdemo.ipx'/>
```

Integrating Multiple Designs into a Top-Level Quartus Prime Project

This design example shows how you can integrate multiple DSP Builder designs into a top-level Quartus Prime project. If your top-level design consists of the following three DSP Builder models:

■ **fir1.mdl**—containing two Avalon-MM slave interfaces

■ **fir2.mdl**—containing multiple HDL import blocks

■ **fir3.mdl**—containing one IP MegaCore function block with two Avalon-ST interfaces

The top-level Quartus Prime project has the following four design files:

■ **top.vhd**—Top-level wrapper that instantiates the three separate models

■ **fir1.qip**—Quartus IP file for **fir1.mdl**

■ **fir2.qip**—Quartus IP file for **fir2.mdl**

■ **fir3.qip**—Quartus IP file for **fir3.mdl**

shows the design example in the Quartus Prime **Project Navigator** window.

☞ In this example, **fir2.qip** has an embedded **.qip** associated with the HDL import block and **fir3.qip** has an embedded **.qip** associated with the IP MegaCore function block.

1. To update the IP Librarian search path, create additional directories *<project directory>*/**ip/***<module name>* and create an **.ipx** file in each subdirectory. Thus in this design example, create the following directories:

   ■ **../***<project directory>*/**ip/fir1**

   ■ **../***<project directory>*/**ip/fir2**

   ■ **../***<project directory>*/**ip/fir3**

2. In each subdirectory, create a text file *<module name>***.ipx** with the following contents:

```
<library>
        <path path='../../../<module name>/**/*'/>
</library>
```

These **.ipx** files specify the relative path to the directory, where the **.mdl** file is located.

Figure 12–2. Project Navigator Window in the Quartus Prime Software
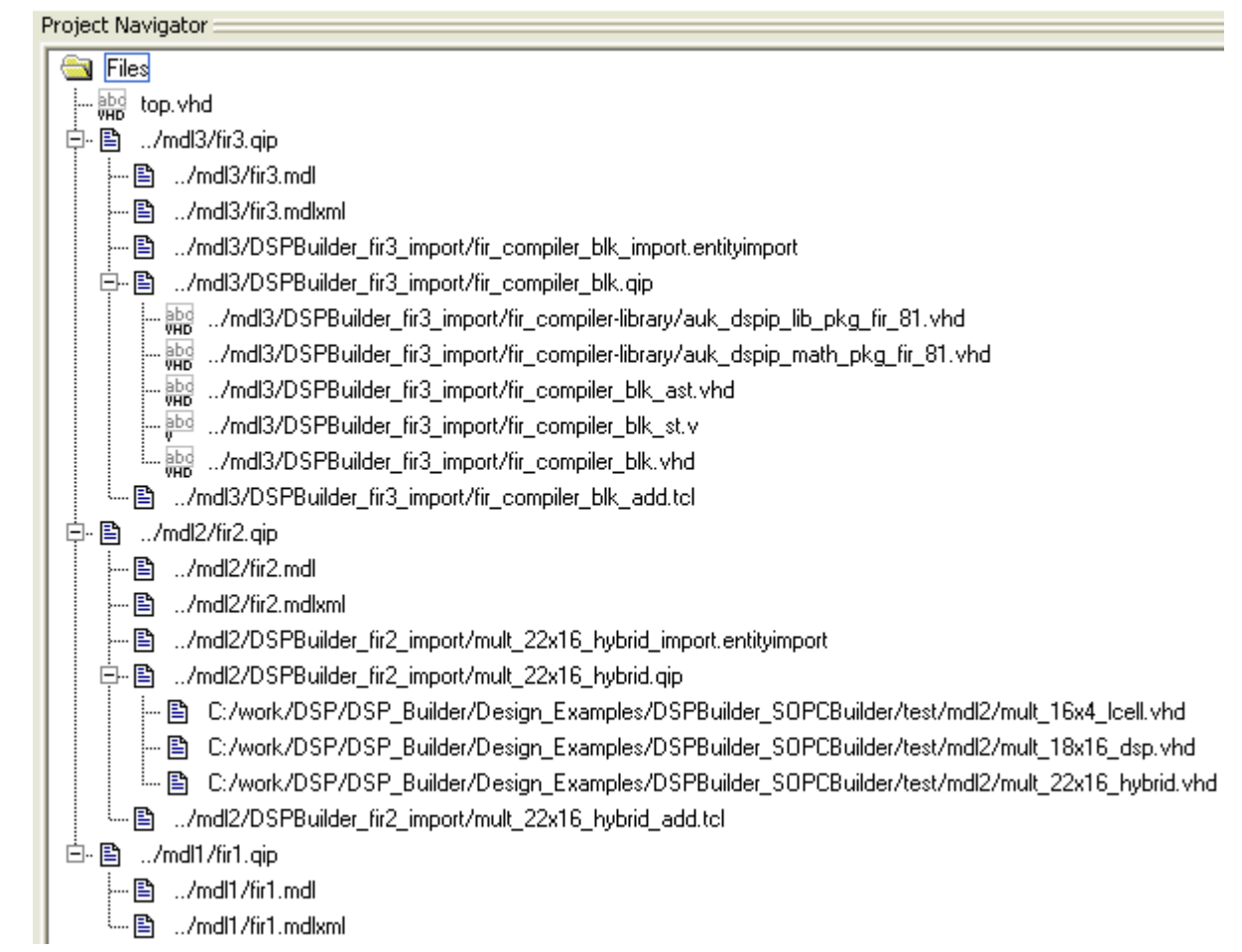
This chapter contains information about resolving the following DSP Builder problems and error conditions:

- Signal Compiler Cannot Checkout a Valid License
- Loop Detected While Propagating Bit Widths
- The MegaCore Functions Library Does Not Appear in Simulink
- The Synthesis Flow Does Not Run Properly
- DSP Development Board Troubleshooting
- SignalTap II Analysis Appears to Hang
- Error if Output Block Connected to an Altera Synthesis Block
- Warning if Input/Output Blocks Conflict with clock or aclr Ports
- Wiring the Asynchronous Clear Signal
- Error Issues when a Design Includes Pre-v7.1 Blocks
- Creating an Input Terminator for Debugging a Design
- A Specified Path Cannot be Found or a File Name is Too Long
- Incorrect Interpretation of Number Format in Output from MegaCore Functions
- Simulation Mismatch For FIR Compiler MegaCore Function
- Simulation Mismatch After Changing Signals or Parameters
- Unexpected Exception Error when Generating Blocks
- VHDL Entity Names Change if a Model is Modified
- Algebraic Loop Causes Simulation to Fail
- Parameter Entry Problems in the DSP Block Dialog Box
- DSP Builder System Not Detected in Qsys
- MATLAB Runs Out of Java Virtual Machine Heap Memory
- ModelSim Fails to Invoke From DSP Builder
- Unexpected End of File Error When Comparing Simulation Results

Signal Compiler Cannot Checkout a Valid License

You may receive this error message if you try to generate VHDL files and Tcl scripts (or try to generate VHDL stimuli) and you have not installed a license for DSP Builder.

For information about how to obtain a license, refer to *Volume 1: Introduction to DSP Builder* in the *DSP Builder Handbook*.

Verifying That Your DSP Builder Licensing Functions Properly

Type the following command in the MATLAB Command Window:

```
dos('lmutil lmdiag C4D5_512A') ↵
```

where C4D5_512 is the DSP Builder feature ID.

This command outputs the status of the DSP Builder license.

For example, if you are using a node locked license:

```
lmutil - Copyright (C) 1989-2006 Macrovision Europe Ltd. and/or
Macrovision Corporation. All Rights Reserved.

FLEXnet diagnostics on Mon 8/11/2008 14:36

-----------------------------------------------------

License file: c:\qdesigns\license.dat

-----------------------------------------------------

"C4D5_512A" v0000.00, vendor: alterad

uncounted nodelocked license, locked to Vendor-defined
                         "GUARD_ID=T000001297" no expiration date
```

☞ You receive a message about the `hostid` if you are using an Altera software guard for licensing.

Alternatively, if you are using a floating license:

```
>> dos('lmutil lmdiag  C4D5_512A')

lmutil - Copyright (c) 1989-2006 Macrovision Europe Ltd. and/or
Macrovision Corporation. All Rights Reserved.

FLEXnet diagnostics on Mon 8/11/2008 10:49

-----------------------------------------------------

License file: node@lic_server

-----------------------------------------------------

"C4D5_512A" v2030.12, vendor: alterad

License server: lic_server

floating license  expires: 31-dec-2030

This license can be checked out

-----------------------------------------------------
```

If the command does not work, your license file may not be set up correctly. For information about how to check your system path and registry settings, refer to .

If your license file has a SERVER line, type the following command in the MATLAB Command Window:

```
dos('lmutil lmstat -a') ↵
```

This command outputs the status of the DSP Builder license in the following format:

```
lmutil - Copyright (c) 1989-2006 Macrovision Europe Ltd. and/or
Macrovision Corporation. All Rights Reserved.

Flexible License Manager status on Mon 8/11/2008 15:36
```

```
License server status:

[Detecting lmgrd processes...]
License server status: node@lic_server
    License file(s) on shama: /usr/licenses/quartus/license.dat:

lic_server: license server UP (MASTER) v10.8

Vendor daemon status (on lic_server):

    alterad: UP v9.2

Feature usage info:

Users of C4D5_512A: (Total of 100 licenses issued; Total of 0 licenses
in use)
```

If the command does not work, your license file may not be set up correctly.

Verifying That the LM_LICENSE_FILE Variable Is Set Correctly

The `LM_LICENSE_FILE` system variable must point to your **license.dat** file that includes the DSP Builder `FEATURE` line for the DSP Builder to operate properly.

☞ If you have multiple versions of software that uses a **license.dat** file (for example, Quartus II Limited Edition and a full version of the Quartus II software), make sure that `LM_LICENSE_FILE` points to the version of software that you want to use with DSP Builder.

Other software products, such as Mentor Graphics LeonardoSpectrum, also use the `LM_LICENSE_FILE` variable to point to a license file. You can combine several license.dat files into one or you can specify multiple license.dat files in the steps below.

Follow these steps to set the `LM_LICENSE_FILE` variable:

1. On the Windows Start menu point to **Settings** and click **Control Panel**.

2. Double-click the **System** icon in the Control Panel window.

3. In the **System Properties** dialog box, click the **Advanced** tab.

4. Click on **Environment Variables**.

5. Click the **System Variable** list to highlight it, and then click **New**.

6. In the **Variable Name** box, type `LM_LICENSE_FILE`.

7. In the **Variable Value** box, type `<path to license file>\license.dat`.

8. Click **OK**.

Verifying the Quartus Prime Path

Verify that the `QUARTUS_ROOTDIR` environment variable points at the correct version of the Quartus Prime software by typing the following command in the MATLAB Command Window:

```
!echo %QUARTUS_ROOTDIR% ↵
```

This command returns the path that the `QUARTUS_ROOTDIR` environment variable specifies. For example:

```
C:\altera\81\quartus
```

If You Still Cannot Get a License

■ Try adding the following paths to your system path:

   ■ **quartus/bin**

   ■ **matlab/bin**

■ Remove and reinstall DSP Builder. After removing DSP Builder, delete any DSP Builder files or directories that remain in the file system to ensure that you re-install a clean file set.

Loop Detected While Propagating Bit Widths

You may get an error if you have a feedback loop in your design and you have not explicitly defined the feedback loop's bit width. Figure 13–1 shows this error.

Figure 13–1. Feedback Loop With Unresolved Width Error
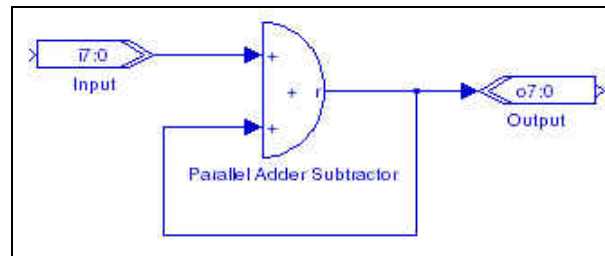
To avoid this error, include an `AltBus` block configured as an internal node to specify the bit width in the feedback loop explicitly (Figure 13–2).

Figure 13–2. Feedback Loop With AltBus Block as an Internal Node
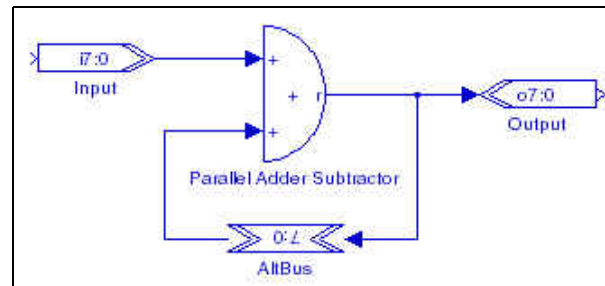
The MegaCore Functions Library Does Not Appear in Simulink

The Simulink Library Browser may not display MegaCore functions library if you install DSP Builder before you install the Altera MegaCore IP Library.

To fix this problem, type the following command after you install the Altera MegaCore IP Library:

```
alt_dspbuilder_setup_megacore ↵
```

The Synthesis Flow Does Not Run Properly

The DSP Builder automated flows allow you to control your entire synthesis and compilation flow in the MATLAB or Simulink environment using the `Signal Compiler` block. With the automated flow, the `Signal Compiler` block outputs VHDL files and Tcl scripts and then automatically begins synthesis and compilation in the Quartus II software.

If the Quartus II software does not run automatically, check the software paths and if necessary, change the system path settings.

Check the Software Paths

If you have multiple versions of the same software product on your PC (for example, Quartus II Web Edition and a full version of the Quartus II software), your registry

DSP Development Board Troubleshooting

If `Signal Compiler` does not configure the device on the DSP development board, check the following points:

■ Ensure that you set up and connect the board to your PC and you install any necessary drivers.

■ When the board powers up, the `CONF_DONE` LED illuminates. The `CONF_DONE` LED turns off and then on when configuration completes successfully. If you do not observe the LED operating in this way, configuration is unsuccessful.

■ You can configure the DSP board manually with an SRAM Object File (.**sof**), a ByteBlasterMV, ByteBlaster II, ByteBlaster, or USB-Blaster download cable, and the Quartus II Programmer in JTAG mode. `Signal Compiler` generates the SRAM object file (.**sof)** file in your working directory.

SignalTap II Analysis Appears to Hang

The SignalTap II logic analyzer should terminate successfully after it meets all trigger conditions. However, if it does not meet one or more of the trigger conditions, the SignalTap II analyzer does not terminate and the JTAG node remains locked.

You can either disconnect and reconnect the USB cable, or switch off the board and switch it on again. You must program the board again if you power it off.

Error if Output Block Connected to an Altera Synthesis Block

An `Output` block maps to output ports in VHDL and marks the edge of the generated system. You should normally use these blocks to connect simulation blocks (Simulink blocks) for your testbench. If you want to use DSP Builder blocks outside your synthesizable system (such as for test bench generation or verification) put `Non-synthesizable Input` and `Non-synthesizable Output` blocks around them.

Warning if Input/Output Blocks Conflict with clock or aclr Ports

A warning issues if an input or output port has the same name as a clock or reset signal that your model uses. For example if your design has an input port `aclr`, this name is the same name as the default system reset and the following warning issues during analysis:

```
Warning: aclrInputPortTest/aclr has been renamed to avoid conflict:
                         aclr has been renamed to aclr_1:
```

The input port renames during HDL conversion. If you want to keep the port `aclr`, add a `Clock` block and use it to rename the reset port.

Wiring the Asynchronous Clear Signal

Wire the asynchronous clear signal with a register to make sure that the end of the `aclr` cycle synchronizes with the clock (Figure 13–3).

Figure 13–3. Wiring the Asynchronous Clear Signal
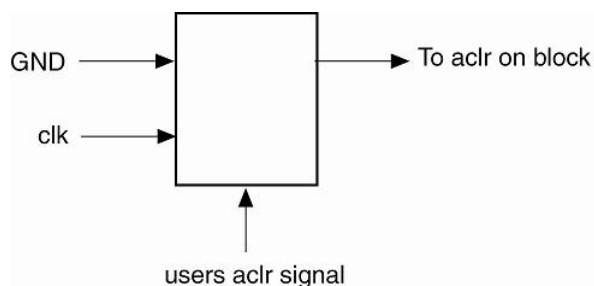
☞ A design may not match the hardware if an asynchronous clear performs during simulation because the `aclr` cycle may last several clocks - depending on clock speed and the device.

Error Issues when a Design Includes Pre-v7.1 Blocks

An error of the following form issues if you attempt to simulate a design that includes unupgraded pre-v7.1 blocks:

```
Data type mismatch. Input port 1 of '<old block>' expects a signal
of data type 'double'. However, it is driven by a signal of data type
'DSPB_Type'.
```

👣 For information about upgrading your designs, refer to *Volume 1: Introduction to DSP Builder* in the *DSP Builder Handbook*.

Creating an Input Terminator for Debugging a Design

If there is a problem somewhere in a design, disconnect some subsystems so that you can analyze a small portion of your design. This procedure may cause bit width propagation and inheritance problems.

You can avoid these problems by inserting a `Non-synthesizable Output` block followed immediately by a `Non-synthesizable Input` block. This combination functions as a temporary input terminator and you can remove them after you debug your design.

A Specified Path Cannot be Found or a File Name is Too Long

The maximum length for a path is limited to 256 characters in the Windows operating system.

When the file path to a model or the name of the model is very long, DSP Builder may attempt to create a file path exceeding this limit.

If this problem occurs, reduce the length of the file path to the model or the length of its name.

Incorrect Interpretation of Number Format in Output from MegaCore Functions

For some MegaCore functions, DSP Builder may be unable to infer whether it should interpret output signals as signed, unsigned, signed fractional. This issue can cause problems when visualizing the output. For example, by directly attaching scopes, when the signal waveform may obscur because of the incorrectly inferred number formats.

Correct this issue by connecting to the output with an `AltBus` block or a `Non-synthesizable Output` block (as appropriate) with the correct bus type assignment.

Simulation Mismatch For FIR Compiler MegaCore Function

FIR Compiler MegaCore function-generated functional simulation models generally do not output valid data until the data storage of these models is clear.

For more information including a formula that estimates the number of cycles before relevant samples are available, refer to the *Simulate the Design* section in the *FIR Compiler User Guide*.

Simulation Mismatch After Changing Signals or Parameters

The simulation results may not match after changing any signal names or parameters. If this problem occurs, delete the previous testbench directory (**tb_<*model name*>**) and run the simulation again.

Unexpected Exception Error when Generating Blocks

DSP Builder issues errors of the following form when you generate a DSP Builder system:

```
Info: IP Generator Info: stderr: No clock info for
                                my_alt_dspbuilder_clock
Info: IP Generator Info: stderr: Failed to find clock
                                my_alt_dspbuilder_clock
```

```
Info: IP Generator Info: stderr: Failed to find clock
                                  my_alt_dspbuilder_clock
Error: IP Generator Error: Unexpected exception thrown by MDLFactory:
                                  java.lang.NullPointerException
Error: Node instance "dut" instantiates undefined entity
        "TestBarrelShifter" File: <path>/mytoplevel.vhd Line: 30
```

This problem is caused by corrupted Librarian IP cache and can be resolved by deleting the IP cache directory which is normally located at:

C:\Documents and Settings*<user>*\\.altera.quartus\ip_cache

VHDL Entity Names Change if a Model is Modified

The Signal Compiler VHDL files have a random number suffix appended to the file if you modify the model.

For example, if you change the pipeline delay on a Delay block, the corresponding VHDL file: **alt_dspbuilder_delay_*<randomnumber>*** changes, while the VHDL file name for the rest of the blocks in the model remain the same.

Solve this problem with a regular expression in the project assignments ("About Quartus II Assignments and Block Entity Names" on page 3–28).

Algebraic Loop Causes Simulation to Fail

HDL import and IP Toolbench-based MegaCore function blocks provide an interface for changing the direct feedthrough settings of their inputs.

Algebraic loops are loops entirely consisting of blocks having some inputs that are direct feedthrough, that is, inputs that have a purely combinational path to at least one output of the block.

For more information about algebraic loops, refer to the MATLAB Help.

The feature to automatically infer the correct direct feedthrough values is disabled by default for HDL Import (and DSP Builder treats all inputs as direct feedthrough). Enable it by typing the following command in the MATLAB command window:

```
set_param(<HDL Import block name>, 'use_dynamic_feedthrough_data', 'on')
```

The direct feedthrough settings for the HDL Import block update after a successful compile of the HDL when this parameter is on.

☞ This feature may not generate correct settings when importing low-level LPM-based HDL.

A more direct method of changing the direct feedthrough settings is to modify the *InDelayed* parameter on HDL Import or MegaCore function blocks, with the following command:

```
set_param(<block name>, 'inDelayed', <feedthrough setting>)
```

For example, if the block is named My_HDL:

```
set_param(<My_HDL>, 'inDelayed', '1 0 0 1')
```

A valid value of this parameter is a series of digits, one for each of the inputs on the block (from top to bottom), with a `0` indicating direct feedthrough, and a `1` indicating that all paths to outputs from this input are registered.

☞ Specifying a value of `1` for an input, when it is in fact direct feedthrough, causes Simulink to treat combinational paths as registered, and results in incorrect simulation results.

Adjust the order in which Simulink exercises all the blocks in a feedback loop, by giving blocks a priority value. This procedure is useful if you know which block is providing the correct initial values.

The priority of a block can be set with the **General** tab in the block properties for a block. A lower value of priority causes DSP Builder to execute a block before a block with a higher value.

Parameter Entry Problems in the DSP Block Dialog Box

There are issues with the **Block Properties** dialog box for the `DSP` block. Some interdependencies require that you close and re-open the dialog box to edit further parameters. This issue may be occur after a warning message issues or when a required option is not available.

For example, if you change the **Output Rounding Operation Type** you may get an error when **Symmetric** is selected for the **Output Saturation Operation Type**. If this occurs, set the saturation type to **None** (wrap) and close the dialog box. Reopen the dialog box and you can select now select **Symmetric** saturation.

DSP Builder System Not Detected in Qsys

Qsys may not detect DSP Builder systems whose hardware has been generated using previous versions of the DSP Builder software. Altera does not guarantee backwards compatibility of these modules when you use them in Qsys.

To workaround this issue, follow these steps:

1. Remove the ***<dspbuilder system name>*_dspbuilder** directory that the older DSP Builder version generated.

2. Re-run compilation from the *Signal Compiler* block with the current DSP Builder version.

3. Refresh the Qsys system.

MATLAB Runs Out of Java Virtual Machine Heap Memory

For a very large design (containing many thousand blocks), MATLAB may have insufficient heap memory available for the Java virtual machine and issues an error message of the form:

```
"OutofMemoryError: Java heap space"
```

For information about how to increase the heap space available to the Java virtual machine, refer to:

http://www.mathworks.com/support/solutions/data/1-18I2C.html

ModelSim Fails to Invoke From DSP Builder

If ModelSim fails to invoke from within DSP Builder, check that the currently supported ModelSim executable (**vsim.exe**) is in the path. Your PC should automaticall include ModelSim in your path if you install ModelSim-Altera but you may need to be add it manually if you use a different ModelSim installation.

You can verify the ModelSim installation by typing the following command at the MATLAB prompt:

```
!vsim
```

This command returns the ModelSim version and the path to the ModelSim preferences Tcl file. If an error message issues or the returned path is incorrect, you may need to move ModelSim to be ahead of any other similar tool in the path.

For information about the supported version of ModelSim, refer to the *DSP Builder Installation and Licensing* manual.

Unexpected End of File Error When Comparing Simulation Results

Occasionally an "Unexpected End of File" error issues when you are comparing the Simulink and ModelSim simulation results for a design with multiple clocks.

This error occurs because the rounding calculation that aligns the clock signals sets ModelSim simulation to run for one additional clock cycle (on the fastest clock) and there is no stimulus data for this extra cycle. You can ignore the error message.

The blocks in the AltLab library manage design hierarchy and generate RTL VHDL for synthesis and simulation.

The AltLab library contains the following blocks:

- BP (Bus Probe)
- Clock
- Clock_Derived
- Display Pipeline Depth
- HDL Entity
- HDL Import
- HDL Input
- HDL Output
- HIL (Hardware in the Loop)
- Quartus Prime Global Project Assignment
- Quartus Prime Pinout Assignments
- Resource Usage
- Signal Compiler
- SignalTap II Logic Analyzer
- SignalTap II Node
- Subsystem Builder
- TestBench
- VCD Sink
- Virtual Pins

BP (Bus Probe)

The `Bus Probe` (BP) block is a sink, which you can place on any node of a model. The `Bus Probe` block does not have any hardware representation and therefore does not appear in the VHDL RTL representation generated by the Signal Compiler block.

The **Display in Symbol** parameter selects the graphical shape of the symbol in your model and the information that is reported there (Table 14–1).

Table 14–1. Bus Probe Block "Display in Symbol" Parameter

Shape of Symbol	Data Reported in Symbol
Circle	Maximum number of integer bits required during simulation.
Rectangle	Maximum or minimum value reached during simulation.

After simulating your model, the `Bus Probe` block back-annotates the following information in the parameters dialog box for the `Bus Probe` block:

■ Maximum value reached during simulation

■ Minimum value reached during simulation

■ Maximum number of integer bits required during simulation

Clock

Use the `Clock` block in the top level of a design to set the base hardware clock domain.

The block name is the name of the clock signal and must be a valid VHDL identifier.

A design can have zero or one base clock in a design and an error issues if you try to use more than one base clock. You can specify the required units and enter any positive value with the specified units. However, the clock period must be greater than 1ps but less than 2.1ms.

If no base clock exists in your design, a default clock (clock) with a 20-ns real-world period and a Simulink sample time of 1 is automatically created with a default Active Low reset (aclr).

☞ To avoid sample time conflicts in the Simulink simulation, ensure that the sample time specified in the Simulink source block matches the sample time specified in the `Input` block (driven by the `Clock` block or a derived clock).

Place additional clocks in the system by adding Clock_Derived blocks.

Each clock must have a unique reset name. As all clock blocks have the same default reset name (aclr) ensure you specify a valid unique name with multiple clocks.

You can add reset synchronizer circuitry for this clock domain by specifying the reset type to be either synchronized active low or synchronized active high.

When you specify these reset types, DSP Builder adds two extra registers to avoid metastability issues during reset removal.

Table 14–2 lists the parameters for the `Clock` block.

Table 14–2. Clock Block Parameters

Name	Value	Description
Real-World Clock Period	user specified	Specify the clock period, which should be greater than 1ps but less than 2.1 ms.
Period Unit	ps, ns, us, ms, s	Specify the units for the clock period (picoseconds, nanoseconds, microseconds, milliseconds, or seconds).
Simulink Sample Time	> 0	Specify the Simulink sample time.
Reset Name	User defined	Specify a unique reset name. The default reset is `aclr`.
Reset Type	Active Low, Active High, Synchronized Active Low, Synchronized Active High	Specify whether the reset signal is active high or active low.
Export As Output Pin	On or Off	Turn on to export this clock as an output pin.

Clock_Derived

Use the `Clock_Derived` block in the top level of a design to add additional clock pins to your design. Specify these clocks as a rational multiple of the base clock for simulation purposes.

DSP Builder uses the block name as the name of the clock signal. It must be a valid VHDL identifier.

You can specify the numerator and denominator multiplicands calculates the derived clock. However, the resulting clock period should be greater than 1ps but less than 2.1ms.

If no base clock is set in your design, DSP Builder creates a 20ns base clock and determines the derived clock period. You must use a Clock block to set the base clock if you want the sample time to be anything other than 1.

☞ To avoid sample time conflicts in the Simulink simulation, ensure that the sample time specified in the Simulink source block matches the sample time specified in the `Input` block (driven by the Clock block or a derived clock).

Each clock must have a unique reset name. As all clock blocks have the same default reset name (`aclr`) ensure you specify a valid unique name with multiple clocks.

You can add reset synchronizer circuitry for this clock domain by specifying the reset type to be synchronized active low or synchronized active high.

When you specify these reset types, DSP Builder adds two extra registers to avoid metastability issues during reset removal.

Table 14–3 lists the parameters for the `Clock_Derived` block:

Table 14–3. Clock_Derived Block Parameters

Name	Value	Description
Base Clock Multiplicand Numerator	>= 1	Multiply the base clock period by this value. The resulting clock period should be greater than 1ps but less than 2.1ms.
Base Clock Multiplicand Denominator	>= 1	Divide the base clock period by this value. The resulting clock period should be greater than 1ps but less than 2.1ms.
Reset Name	User defined	Specify a unique reset name. The default reset is `aclr`.
Reset Type	Active Low, Active High, Synchronized Active Low, Synchronized Active High	Specify whether the reset signal is active high or active low.
Export As Output Pin	On or Off	Turn on to export this clock as an output pin.

Display Pipeline Depth

The `Display Pipeline Depth` block controls whether the pipeline depth displays on primitive blocks.

You can change the display mode by double-clicking on the block. When set, the current pipeline depth displays at the top right corner of each block that adds latency to your design. The currently selected mode shows on the `Display Pipeline Depth` block symbol.

Changing modes causes a Simulink display update, which may be slow for very large designs.

The `Display Pipeline Depth` block has no parameters.

HDL Entity

Use the `HDL Entity` block for black-box simulation subsystems that you include in your design with a Subsystem Builder block. The `HDL Entity` block specifies the name of the HDL file that DSP Builder substitutes for the subsystem and the names of the clock and reset ports for the subsystem.

The Subsystem Builder block usually creates this block.

Table 14–4 shows the parameters for the `HDL Entity` block.

Table 14–4. HDL Entity Block Parameters

Name	Value	Description
HDL File Name	User defined	Specifies the name of the HDL file that DSP Builder substitutes for the subsystem represented by a `Subsystem Builder` block.
Clock Name	User defined	Specifies the name of the clock signal that the black-box subsystem uses.
Reset Name	User defined	Specifies the name of the reset signal that the black-box subsystem uses.
HDL takes port names from Subsystem	On or Off	Turn on to use the subsystem port names as the entity port names instead of the names of the `HDL Input` and `HDL Output` blocks.

HDL Import

Use the `HDL Import` block to import existing blocks implemented in HDL into DSP Builder. Individually specify the VHDL or Verilog HDL files or define in a Quartus® II project file (**.qpf**).

☞ You must save your model file before you can import HDL with the `HDL Import` block.

When you click **Compile**, a simulation file generates and the block in your model configures with the required input and output ports. The Quartus Prime software synthesizes the imported HDL or project as a netlist of megafunctions, LPM functions, and gates.

DSP Builder may explicitly instantiate the megafunctions and LPM functions in the imported files, or the Quartus Prime software may infer them. The netlist then compiles into a binary simulation netlist for use by the HDL simulation engine in DSP Builder.

When simulating imported VHDL in ModelSim, which includes FIFO buffers, Xs in the simulation results may give a mismatch with the Simulink simulation. You should use the FIFO buffer carefully to avoid any overflows or underflows. Examine and eliminate any warnings of Xs that ModelSim reports during simulation before you compare to the Simulink results.

The simulator supports many of the common megafunctions and LPM functions although it does not support some. If DSP Builder encounters an unsupported function, it issues an error message after you click `Compile` and it cannot import the HDL. However, you may be able to rewrite the HDL so that the Quartus Prime software infers a different megafunction or LPM function.

HDL Import Parameters

Table 14–5 shows the parameters for the `HDL Import` block.

Table 14–5. HDL Import Block Parameters (Part 1 of 2)

Name	Value	Description
Import HDL	**On** or Off	You can import individual HDL files when this option is on.
Add	**.v** or **.vhd** file	Click to browse for one or more VHDL files or Verilog HDL files.
Remove	—	Click to remove the selected file from the list.
Up, Down	—	Click to change the compilation order by moving the selected HDL file up or down the list. The file order is not important when you use the Quartus Prime software but may be significant when you use other downstream tools (such as ModelSim).
Enter name of top level design entity	Entity name	Specifies the name of the top level entity in the imported HDL files.
Import Quartus Prime Project	On or **Off**	When this option is on, you can specify the HDL to import with a Quartus Prime project file (**.qpf**). DSP Builder imports the current HDL configuration. To import a different revision, specify the required revision in the Quartus Prime software. The source files that the Quartus Prime project uses must be in the same directory as your model file or be explicitly referenced in the Quartus Prime settings file (**.qsf**). Error messages issue for any entities that DSP Builder cannot find. Refer to the Quartus Prime documentation for information about setting the current revision of a project and how to explicitly reference the source files in your design.

Table 14–5. HDL Import Block Parameters (Part 2 of 2)

Browse	.**qpf** file	Click to browse for a Quartus Prime project file.
Sort top-level ports by name	On or **Off**	Turn on to sort the ports that the top-level HDL file alphabetically defines instead of the order specified in the HDL.
Compile	—	Compiles a simulation model from the imported HDL and displays the ports defined in the imported HDL on the block.

HDL Import Example

Figure 14–1 shows an example of an imported HDL design implementing a simple adder with four input ports (Input, Input1, Input2, sclrp), and two output ports (Output, Output1).

Figure 14–1. Typical HDL Import Block
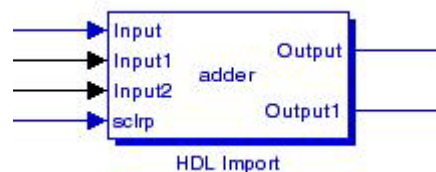
Use *std_logic_1164* types to define the input and output interfaces to the imported VHDL. If your design uses any other VHDL type definitions (such as arithmetic or numeric types), you should write a wrapper that converts them to *std_logic* or *std_logic_vector*.

HDL import only supports single clock designs. If you import a design with multiple clocks, DSP Builder uses one clock as the implicit clock and shows any others as input ports on the Simulink block.

☞ Store HDL source files in any directory or hierarchy of directories.

HDL Import Supported Megafunctions

Table 14–6 lists the supported megafunctions and LPM functions.

Table 14–6. Supported Megafunctions and LPM Functions

Megafunctions		LPM Functions	
a_graycounter	altsyncram	lpm_abs	lpm_mult [1]
altaccumulate	parallel_add	lpm_add_sub	lpm_mux
altmult_add	scfifo	lpm_compare	lpm_ram_dp
altshift_taps		lpm_counter	

Note to Table 14–6:

(1) The lpm_mult LPM function is not supported when configured to perform a squaring operation.

HDL Input

Connect the `HDL Input` block directly to an input node in a subsystem. Use with the Subsystem Builder and HDL Entity blocks for black-box simulation.

The type and bit width must match the type and bit width on the corresponding input port in the HDL file referenced by the `HDL Entity` block. `HDL Input` blocks are automatically generated by the Subsystem Builder block.

You can optionally specify the external Simulink type. If set to `Simulink Fixed Point Type`, the bit width is the same as the input. If set to `Double`, the width may be truncated if the bit width is greater than 52.

Table 14–7 shows the `HDL Input` block parameters.

Table 14–7. HDL Input Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, Single Bit	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
External Type	Inferred, Simulink Fixed Point Type, Double	Specifies whether the external type is inferred from the Simulink block it is connected to or explicitly set to either Simulink Fixed Point or Double type. The default is Inferred.

Table 14–8 on page 14–7 shows the `HDL Input` block I/O formats.

Table 14–8. HDL Input Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$	I1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit - Optional
O	$O1_{[LP].[RP]}$	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 14–8:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

HDL Output

The `HDL Output` block should be connected directly to an output node in a subsystem. Use with the Subsystem Builder and `HDL Entity` blocks for black-box simulation.

The type and bit width must match the type and bit width on the corresponding output port in the HDL file referenced by the `HDL Entity` block. `HDL Output` blocks are automatically generated by the Subsystem Builder block.

Table 14–9 shows the HDL Output block parameters.

Table 14–9. HDL Output Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, Single Bit	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.

Table 14–10 shows the HDL Output block I/O formats.

Table 14–10. HDL Output Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$	I1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit - Optional
O	$O1_{[LP].[RP]}$	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 14–10:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

HIL (Hardware in the Loop)

The HIL (Hardware in the Loop) block allows you to use an FPGA as a simulation device inside a Simulink design. This configuration accelerates the simulation time, and also allows access to real hardware in a simulation.

To use an HIL block, you need an FPGA development board with a JTAG interface. Use any JTAG download cable, such as a ByteBlasterMV™, ByteBlaster™, or USB-Blaster™ cable.

HIL supports advanced features, including:

■ Exported ports (allows the use of hardware components connected to the FPGA)

■ Burst mode (improves HIL simulation speed)

☞ This block supports only single clock designs with registered paths in a design. The simulation results may be unreliable for combinational paths.

Figure 14–2 shows an example with the HIL block.

Figure 14–2. Example With the HIL Block
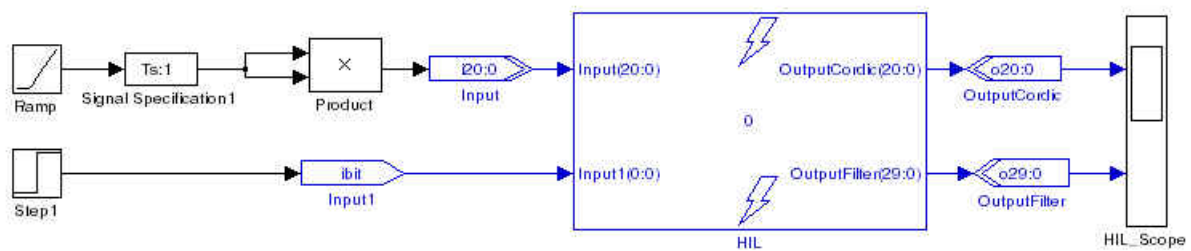
☞ Refer to the *"Using Hardware in the Loop"* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

Table 14–11 shows the parameters specified in page 1 of the HIL dialog box.

Table 14–11. HIL Block Parameters, Page 1

Name	Value	Description
Select the Quartus Prime project	.qpf file	Browse for a Quartus Prime project file ,which describes the hardware design that the HIL block uses.
Select the clock pin	Port name	The clock pin name for the hardware design in the Quartus Prime software.
Select the reset pin	Port name	The reset pin name for the hardware design in the Quartus Prime software.
Identify the signed ports	Signed or Unsigned	Set the number of bits and select the type (signed or unsigned) of each input and output port in the hardware design.
Export	On or Off	When on, the selected port is exported on an FPGA pin (or on multiple pins for buses). When off (the default), the port is exported to the Simulink model.
Select the reset level	Active_High, Active_Low	The reset level that matches the setting in the original design. For designs originated from the standard blockset, the reset level is specified in the Clock or Clock_Derived block. If your design uses no clock block, it uses a default clock with reset level active high. For designs originated from the advanced blockset, the reset level is specified in the Signals block.
Burst Mode	On or Off	When on, allows sending data to the FPGA in bursts, which improves the simulation speed, but delays the outputs by the burst length. When Off, it defaults to single-step mode.
Burst Length	*(1)*	Specify the length of a burst ("1" is equivalent to disabling burst mode). Use higher values to produce faster simulations (although the extra gain becomes negligible with bigger burst sizes).
Sampling Period	Integer	Specify the sample time period in seconds. (A value of -1 means that the sampling period is inherited from the block connected to the inputs.)
Assert "Sclr" before starting the simulation	On or Off	When on, asserts the synchronous clear signal before the simulation starts.

Note to Table 14–11:

(1) The record size is 32×1024×1024, which is the product of (*packet size*) × (*burst length*) while the packet size is the larger of the total input data width and the total output data width. For example, for a packet size of 1024 bits, set the burst length to 32×1024. However, due to the limitations of the JTAG interface, the optimal record size is between 1 to 2 MBPS (depending on the host computer, USB driver and cables). Hence, setting a bigger burst size might not give significant speed up.

☞ The `HIL` block needs recompilation if you change the Quartus Prime project, clock pin, or any of the exported ports.

Table 14–12 shows the parameters specified in page 2 of the `HIL` dialog box.

Table 14–12. HIL Block Parameters, Page 2

Name	Value	Description
FPGA device	device name	The FPGA device.
Compile with Quartus Prime	—	Click to compile the `HIL` block with the Quartus Prime software.
JTAG Cable	cable name	The JTAG cable.
Device in chain	device location	The required entry for the location of the device.
Scan JTAG	—	Click to scan the JTAG interface for all JTAG cables attached to the system (including any remote computers) and the devices on each JTAG cable. The available cable names and device names are loaded into the **JTAG Cable** and **Device in chain** list boxes.
Configure FPGA	—	Click to configure the FPGA.
Transcript window	—	Displays the progress of the compilation.

Quartus Prime Global Project Assignment

This block passes Quartus® II global project assignments to the Quartus Prime project. Each block sets a single assignment. If you need to make multiple assignments, use multiple blocks (Figure 14–3). These assignments could set Quartus II compilation directives such as target device or timing requirements.

☞ You cannot assign the device, family, or f_{MAX} requirement with this block. Use the `Signal Compiler` block to make device and family settings, or the Clock and Clock_Derived blocks to make explicit clock settings.

👣 For a full list of Quartus Prime global assignments and their syntax, refer to the *Quartus Prime Settings File Reference Manual* or use the following Quartus Prime shell command:

```
quartus_sh --tcl_eval get_all_assignment_names
```

Table 14–13 shows the `Quartus Prime Global Project Assignment` block parameters.

Table 14–13. Quartus Prime Global Project Assignment Block Parameters

Name	Value	Description
Assignment Name	String	Specify the assignment name.
Assignment Value	String	Specify the assignment value with any optional arguments. Any values or arguments that contain spaces or other special characters must be enclosed in quotes.

Figure 14–3 shows an example defining multiple assignments with Quartus Prime Global Project Assignment blocks.

Figure 14–3. Assignments With Quartus Prime Global Project Assignment Blocks
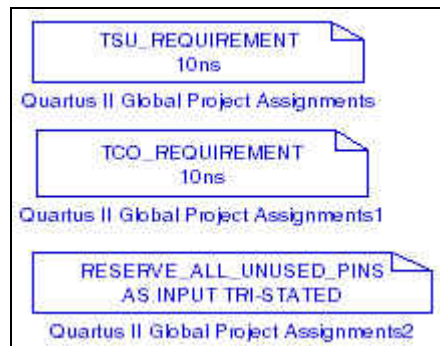
Quartus Prime Pinout Assignments

The Quartus Prime Pinout Assignments block passes Quartus® II project pinout assignments to the Quartus Prime project generated by the Signal Compiler block.

Only use this block at the top level of your model. This block sets the pinout location of the Input or Output blocks in your model, which have the specified pin names.

For buses, use a comma to separate the bit pin assignment location from LSB to MSB.

For example:

Pin Name: abc
Pin Location: Pin_AA, Pin_AB, Pin_AC

assigns abc[0] to Pin_AA, abc[1] to Pin_AB, and abc[2] to Pin_AC

To set the pin assignment for a clock, use the name of the Clock block (for example, the default is clock) for the pin name. For example:

Pin Name: clock
Pin Location: Pin_AM17

To set the pin assignment for a reset, use the name of the reset signal specified in the Clock block (for example the default global reset is aclr) for the pin name. For example:

Pin Name: aclr
Pin Location: Pin_B4

Table 14–14 shows the Quartus Prime Pinout Assignments block parameters.

Table 14–14. Quartus Prime Pinout Assignments Block Parameters

Name	Value	Description
Pin Name	String	The pin name must be the exact instance name of the Input or Output block from the IO & Bus library.
Pin Location	String	Pin location value of the FPGA IO. Refer to the Quartus Prime Help for the pinout values of a device.

Figure 14–4 shows an example with the `Quartus Prime Pinout Assignments` block.

Figure 14–4. Assignments With Quartus Prime Pinout Assignments Blocks
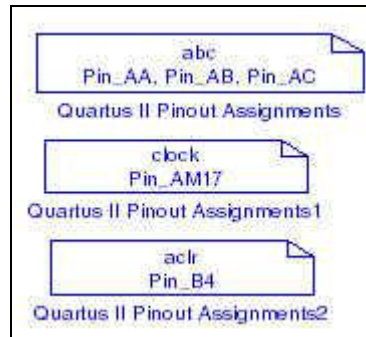
Resource Usage

Use the `Resource Usage` block to check the hardware resources, display timing information, and highlight the critical paths in your design.

☞ You must save your model file and run Signal Compiler before you can use the `Resource Usage` block.

The `Resource Usage` block displays an estimate of the logic, block RAM and DSP blocks resources required by your design.

You can double-click on the `Resource Usage` block to display more information about the blocks in your design that generate hardware.

👣 The information that displays depends on the selected device family. Refer to the device documentation for more information.

Select the **Timing** tab and click **Highlight path** to highlight the critical paths on your design.

☞ When the source and destination in the dialog box are the same and you highlight a single block, the critical path is because of the internal function or a feedback loop.

Signal Compiler

Use the `Signal Compiler` block to create and compile a Quartus Prime project for your DSP Builder design, and to program your design onto an Altera® FPGA.

☞ You must save your model file before you can use the `Signal Compiler` block.

Table 14–15 shows the controls and parameters for the `Signal Compiler` block.

Table 14–15. Signal Compiler Block Parameters Settings Page

Name	Value	Description
Family	—	The Altera device family you want to target. If you use the automated design flow, the Quartus II software automatically uses the smallest device in which your design fits.
Use Board Block to Specify Device	On or Off	Turn on to get the device information from the development board block.
Compile	—	Click to compile your design.
Scan JTAG	List of ports connected to the JTAG cable.	The required JTAG cable port.
Program	—	Click to download your design to the connected development board.
Analyze	—	Click to analyze the DSP Builder system.
Synthesis	—	Click to run Quartus Prime synthesis.
Fitter	—	Click to run the Quartus Prime Fitter tool.
Enable SignalTap II	On or Off	Turn on to enable use of a SignalTap II Logic Analyzer block in your design. Turn on this setting to add extra logic and memory to capture signals in hardware in real time.
SignalTap II depth	2, 4, 8, 16, 32, 64, 128, 256, 512, 1k, 2K, 4K, 8K	The required depth for the SignalTap II Logic Analyzer.
SignalTap II clock	User defined	Specifies the clock to use for capturing data with the SignalTap II feature from a list of available signals.
Use Base Clock	On or Off	Turn on if you want to use the base clock for the SignalTap II Logic Analyzer.
Export	—	Exports synthesizable HDL to a user-specified directory.

☞ Use a Clock or Clock_Derived block to specify the clock and reset signals.

SignalTap II Logic Analyzer

As programmable logic design complexity increases, system verification in software becomes time consuming and replicating real-world stimulus is increasingly difficult. To alleviate these problems, you can supplement traditional system verification with efficient board-level verification.

DSP Builder supports the SignalTap® II embedded logic analyzer, which lets you capture signal activity from internal Altera device nodes while the system under test runs at speed. Use the `SignalTap II Logic Analyzer` block to set up event triggers, configure memory, and display captured waveforms.

You use the SignalTap II Node block to select signals to monitor. Samples are saved to internal embedded system blocks (ESBs) when the logic analyzer is triggered, and are subsequently streamed off chip via the JTAG port with an Altera download cable. The captured data is then stored in a text file, displayed as a waveform in a MATLAB plot, and transferred to the MATLAB workspace as a global variable.

Table 14–16 shows the `SignalTap II Logic Analyzer` block parameters.

Table 14–16. SignalTap II Logic Analyzer Block Parameters Page

Name	Value	Description
Scan JTAG	List of ports connected to the JTAG cable.	The JTAG cable port.
Acquire	—	Click to acquire data from the development board.
SignalTap Nodes	List of SignalTap II node blocks.	Click to select a node and use the Change button to set a trigger condition.
Change	Don't Care, High, Low, Rising Edge, Falling Edge, Either Edge	Click the Change button to set the specified logic condition as the trigger condition for the selected node.

For detailed instructions on with the `SignalTap II Logic Analyzer` and `SignalTap II Node` blocks, refer to the *Performing SignalTap II Logic Analysis* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

Use the `SignalTap II Node` block with the SignalTap II Logic Analyzer block to capture signal activity from internal Altera device nodes while the system under test runs at speed. The `SignalTap II Node` block specifies the signals (also called nodes) for which you want to capture activity.

The `SignalTap II Node` block has no parameters.

For an example of a design with the `SignalTap II Logic Node` block, refer to the description of the SignalTap II Logic Analyzer block.

Refer to the *Performing SignalTap II Logic Analysis* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

SignalTap II Node

Use the `SignalTap II Node` block with the SignalTap II Logic Analyzer block to capture signal activity from internal Altera device nodes while the system under test runs at speed. The `SignalTap II Node` block specifies the signals (also called nodes) for which you want to capture activity.

The `SignalTap II Node` block has no parameters.

For an example of a design with the `SignalTap II Logic Node` block, refer to the description of the SignalTap II Logic Analyzer block.

Refer to the *Performing SignalTap II Logic Analysis* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

Subsystem Builder

The `Subsystem Builder` block allows you to build black-box subsystems that synthesize user-supplied VHDL and simulate non-DSP Builder Simulink blocks. This alternative to HDL import gives better simulation speed. You can also use this block if you cannot use HDL import because of unsupported megafunctions or LPMs.

The subsystem connects the inputs and outputs in the specified VHDL to `HDL Input` and `HDL Output` blocks and creates an `HDL Entity` block, which you can modify if the clock and reset signals are not correctly identified.

The `Subsystem Builder` block automatically maps any input ports named `simulink_clock` in the VHDL entity section to the global VHDL clock signal, and maps any input ports named `simulink_sclr` in the VHDL entity section to the global VHDL synchronous clear signal.

The VHDL entity should be formatted according to the following guidelines:

■ The VHDL file should contain a single entity

■ Port direction: `in` or `out`

■ Port type: `STD_LOGIC` or `STD_LOGIC_VECTOR`

■ Bus size:

   ■ `a(7 DOWNTO 0)` is supported (0 is the LSB, and must be 0)

   ■ `a(8 DOWNTO 1)` is not supported

   ■ `a(0 TO 7)` is not supported

■ Single port declaration per line:

   ■ `a:STD_LOGIC;` is supported

   ■ `a,b,c:STD_LOGIC;` is not supported

The Verilog HDL module should be formatted according to the following guidelines:

■ The Verilog HDL file should contain a single module

■ Port direction: `input` or `output`

■ Bus size:

   ■ `input [7:0] a;` is correct (0 is the LSB, and must be 0)

   ■ `input [8:1] a;` is not supported

   ■ `input [0:7] a;` is not supported

■ Single port declaration per line:

   ■ `input [7:0] a;` is correct

   ■ `input [7:0] a,b,c;` is not supported

To use the `Subsystem Builder` block, drag and drop it into your model, click **Select HDL File**, specify the file to import, and click **Build**.

Table 14–17 shows the `Subsystem Builder` block parameters.

Table 14–17. Subsystem Builder Block Parameters

Name	Value	Description
Select HDL File	User defined	Browse for the VHDL or Verilog HDL file to import.
Build SubSystem	—	Click to build a subsystem for the selected HDL file.

Figure 14–5 shows an example with the Subsystem Builder block.

Figure 14–5. Example With the Subsystem Builder Block
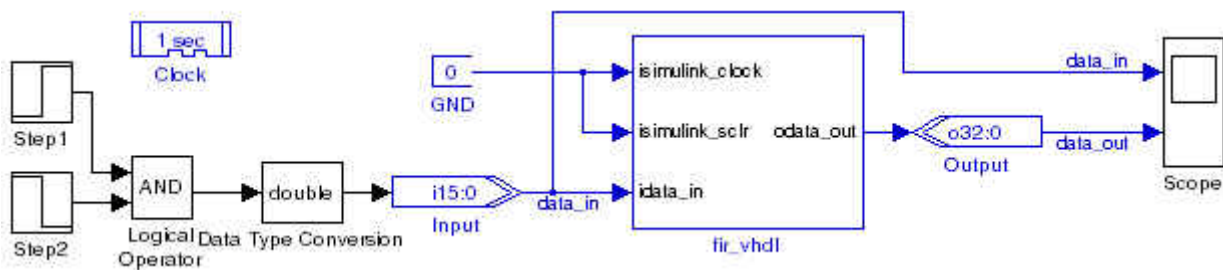
TestBench

The TestBench block controls the generation of a testbench. If the ModelSim executable (**vsim.exe**) is available on your path, you can load the testbench into ModelSim and compare the results with Simulink. Input and output vectors are generated when you use the **Compare against HD**L option in the **Simple** tab or **Run Simulink** in the **Advanced** tab.

You can optionally launch the ModelSim GUI to visually view the ModelSim simulation.

☞ Enabling testbench generation may slow simulation as all input and output values are stored to a file.

Table 14–18 shows the TestBench block parameters.

Table 14–18. TestBench Block Parameters

Name	Value	Description
Enable Testbench generation	On or Off	Turn on to enable automatic testbench generation.
Compare against HDL	—	Click to generate HDL, run Simulink and compare the Simulink simulation results with ModelSim.
Generate HDL	—	Click to generate a VHDL testbench from the Simulink model.
Run Simulink	—	Re-run the Simulink simulation.
Run ModelSim	—	Load the testbench into the ModelSim simulator.
Launch GUI	On or Off	Turn on to launch the ModelSim graphical user interface.
Compare Results	—	Compare the Simulink and ModelSim results.
Mark ModelSim Unknowns (X's) as	**Error**, Warning, Info	Display ModelSim unknown values as error, warning, or info messages. Errors display in red; warnings in blue; info in green.
Maximum number of mismatches to display	>=0 Default = **10**	Specify the maximum number of mismatches to display.

VCD Sink

The VCD Sink block exports Simulink signals to a third-party waveform viewer. When you run the simulation of your model, the VCD Sink block generates a value change dump (.**vcd**) *<VCD Sink block name>***.vcd** file, which a third-party waveform viewer can read.

The VCD Sink block does not have any hardware representation and therefore does not appear in the VHDL RTL representation created by the Signal Compiler block.

Table 14–19 shows the parameters for the VCD Sink block.

Table 14–19. VCD Sink Block Parameters

Name	Value	Description
Number of Inputs	An integer greater than 0	Specify the number of input ports on the VCD Sink block.

Figure 14–6 shows an example of the VCD Sink block

Figure 14–6. Simulink Model With the VCD Sink Block
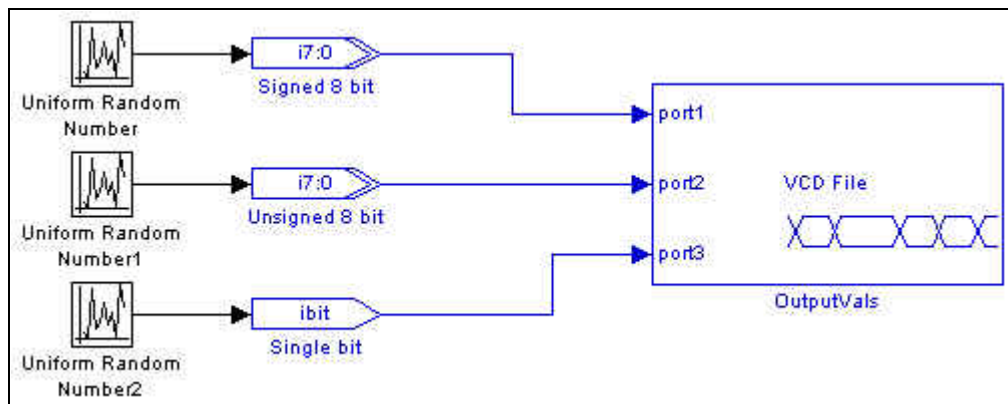
Virtual Pins

The Virtual Pins block allows you to create global assignments in the Quartus Prime project to force all the pins to be virtual. Adding this block to your design allows you to avoid any Quartus Prime compilation errors when you are targeting smaller devices.

The Arithmetic library contains two's complement signed arithmetic blocks such as multipliers and adders. Some blocks have a **Use Dedicated Circuitry** option, which implements functionality into dedicated hardware in the Altera FPGA devices (that is, in the dedicated DSP blocks of these devices).

The Arithmetic library contains the following blocks:

- Barrel Shifter
- Bit Level Sum of Products
- Comparator
- Counter
- Differentiator
- Divider
- DSP
- Gain
- Increment Decrement
- Integrator
- Magnitude
- Multiplier
- Multiply Accumulate
- Multiply Add
- Parallel Adder Subtractor
- Pipelined Adder
- Product
- SOP Tap
- Square Root
- Sum of Products
- Variable Precision DSP

Barrel Shifter

The `Barrel Shifter` block shifts the input data `a` by the amount set by the `distance` bus. The `Barrel Shifter` block can shift data to the left (toward the MSB) or to the right (toward the LSB).

The `Barrel Shifter` block shift data to the left only, or to the right only, or in the direction specified by the optional `direction` input. The shifting operation is an arithmetic shift and not a logical shift; that is, the shifting operation preserves the input data sign for a right shift although the input sign is lost for a left shift.

Table 15–1 shows the `Barrel Shifter` block inputs and outputs.

Table 15–1. Barrel Shifter Block Inputs and Outputs

Signal	Direction	Description
a	Input	Data input.
distance	Input	Distance to shift.
direction	Input	Direction to shift (0 = shift left, 1 = shift right).
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
r	Output	Result after shift.

Table 15–2 shows the `Barrel Shifter` block parameters.

Table 15–2. Barrel Shifter Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The bus number format that you want to use.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This field is zero (0) unless Signed Fractional is selected.
Enable Pipeline	On or Off	Turn on to pipeline the barrel shifter with a latency of 3. Enabling pipeline, increases latency and may increase the f_{MAX} of your design.
Infer size of distance port from input port	On or Off	Turn off to specify the bit width of the distance port. When on, the design uses the full input bus width.
Bit width of distance port	>= 0 (Parameterizable)	Specify the width in bits of the distance port. Defaults to the size of the input port.
Shift Direction	Shift Left, Shift Right, Use direction input pin	The direction you want to shift the bits or specify the direction with the `direction` input.
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use asynchronous Clear Port	On or Off	Turn on to enable the asynchronous clear input. This option is available only when the pipeline option is enabled.
Use Dedicated Circuitry	On or Off	If you target devices that support DSP blocks, turn on to implement the functionality in DSP blocks instead of logic elements.

Table 15–3 shows the `Barrel Shifter` block I/O formats.

Table 15–3. Barrel Shifter Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[L1].[R1]	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
	I2[L2].[R2]	I2: in STD_LOGIC_VECTOR({L2 + R2 - 1} DOWNTO 0)	Explicit
	I3[1]	I3: in STD_LOGIC	
O	O1[L1].[R1]	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0	Explicit

Notes to Table 15–3:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 15–1 shows an example with the `Barrel Shifter` block.

Figure 15–1. Barrel Shifter Block Example
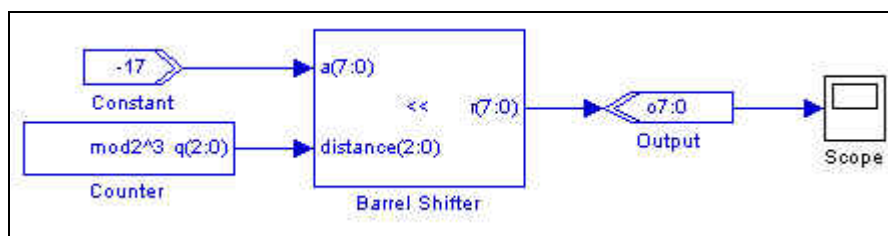
Bit Level Sum of Products

The `Bit Level Sum of Products` block performs a sum of the multiplication of one-bit inputs by signed integer fixed coefficients.

The `Bit Level Sum of Products` block uses the equation:

$$q = a(0)C0 + ... + a(i)Ci + ... + a(n–1)C_{n-1}$$

where:

■ q is the output result

■ $a(i)$ is the one-bit input data

■ Ci are the signed integer fixed coefficients

■ n is the number of coefficients in the range one to eight

Table 15–4 on page 15–4 shows the `Bit Level Sum of Products` block inputs and outputs.

Table 15–4. Bit Level Sum of Products Block Inputs and Outputs

Signal	Direction	Description
`a(0)` to `a(n-1)`	Input	1 to 8 ports corresponding to the signed integer fixed coefficient values specified in the block parameters.
`ena`	Input	Optional clock enable.
`sclr`	Input	Optional synchronous clear.
`q`	Output	Result.

Table 15–5 shows the `Bit Level Sum of Products` block parameters.

Table 15–5. Bit Level Sum of Products Block Parameters

Name	Value	Description
Number of Coefficients	1–8	The number of coefficients.
Coefficient Number of Bits	>= 1–51 (Parameterizable)	Specify the bit width as a signed integer. The bit width must be capable of being expressed as a double in MATLAB.
Signed Integer Fixed-Coefficient Values	User Defined (Parameterizable)	Specify the coefficient values for each port as a sequence of signed integers. the coefficient values must be capable of being expressed as a double in MATLAB. For example: [-21 2 13 5]
Register Inputs	On or Off	When on, a register is added on the input signal.
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear input (`sclr`).

Table 15–6 shows the `Bit Level Sum of Products` block I/O formats.

Table 15–6. Bit Level Sum of Products Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[1].[0]}$ ... $Ii_{[1].[0]}$ ... $In_{[1].[0]}$ $I(n+1)_{[1]}$ $I(n+2)_{[1]}$	I1: in STD_LOGIC ... Ii: in STD_LOGIC ... In: in STD_LOGIC I(n+1): in STD_LOGIC I(n+2): in STD_LOGIC	Explicit
O	$O1_{[L0].[0]}$	O1: out STD_LOGIC_VECTOR({L0 - 1} DOWNTO 0)	Explicit

Notes to Table 15–6:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 15–2 shows an example with the `Bit Level Sum of Products` block.

Figure 15–2. Bit Level Sum of Products Block Example
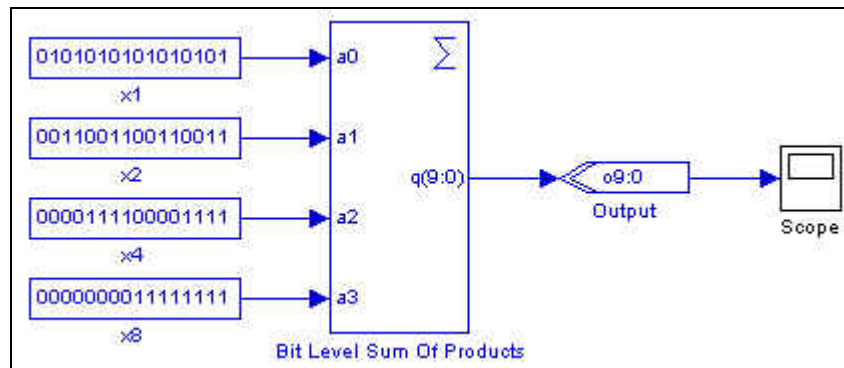
Comparator

The `Comparator` block compares two Simulink signals and returns a single bit. The `Comparator` block implicitly understands the input data type (for example, signed binary or unsigned integer) and produces a single-bit output.

Table 15–7 shows the `Comparator` block inputs and outputs.

Table 15–7. Comparator Block Inputs and Outputs

Signal	Direction	Description
a	Input	Operand a.
b	Input	Operand b.
<unnamed>	Output	Result.

Table 15–8 shows the `Comparator` block parameters.

Table 15–8. Comparator Block Parameters

Name	Value	Description
Operator	a == b, a ~= b, a < b, a <= b, a >= b, a > b	The operation you want to perform on the two buses.

Table 15–9 shows the `Comparator` block I/O formats.

Table 15–9. Comparator Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
I	$I1_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit
	$I2_{[L2].[R2]}$	I1: in STD_LOGIC_VECTOR({L2 + R2 - 1} DOWNTO 0)	Implicit

Table 15–9. Comparator Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
O	O1[1]	O1: out STD_LOGIC	Implicit

Notes to Table 15–9:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 15–3 shows an example with the Comparator block.

Figure 15–3. Comparator Block Example
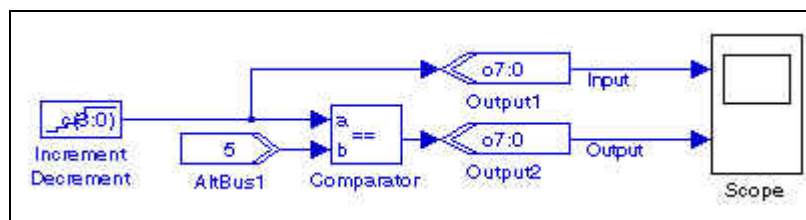
Counter

The Counter block is an up/down counter. For each cycle, the counter increments or decrements its output by the smallest amount that DSP Builder can represent with the selected bus type.

Table 15–10 shows the Counter block inputs and outputs.

Table 15–10. Counter Block Inputs and Outputs

Signal	Direction	Description
data	Input	Optional parallel data input.
sload	Input	Optional synchronous load signal.
sset	Input	Optional synchronous set port. (Loads the specified constant value into the counter.)
updown	Input	Optional direction (1 = up; 0 = down).
clk_ena	Input	Optional clock enable. (Disables counting and sload, sset, sclr signals.)
ena	Input	Optional counter enable. (Disables counting but not sload, sset, and sclr signals.)
sclr	Input	Optional synchronous clear. (Loads zero into the counter.)
q	Output	Result.

Table 15–11 shows the Counter block parameters.

Table 15–11. Counter Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Unsigned Integer, Signed Fractional	The bus number format that you want to use for the counter.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point.

Table 15–11. Counter Block Parameters

Name	Value	Description
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This field is ignored unless Signed Fractional selected.
Use Modulo	On or Off	Turn on to enable the Count Modulo parameter. This option is not available for bit widths greater than 31.
Count Modulo	User defined (Parameterizable)	Specify the maximum count plus 1. This represents the number of unique states in the counter's cycle.
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined	Specify the clock signal name.
Counter Direction	Increment, Decrement, Use Direction Port (updown)	The direction you want to count or specify the direction with the `direction` input.
Use Synchronous Load Ports	On or Off	Turn on to use the synchronous load inputs (`data`, `sload`).
Use Synchronous Set Port	On or Off	Turn on to use the synchronous set input (`sset`). This option is not available for bit widths greater than 31.
Set Value	User defined	Specify the constant value loaded when the design uses the `sset` input. This value must be less than the Count Modulo value (if used).
Use Clock Enable Port	On or Off	Turn on to use the clock enable input (`clk_ena`).
Use Counter Enable Port	On or Off	Turn on to use the counter enable input (`ena`).
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear input (`sclr`).

Table 15–12 shows the `Counter` block I/O formats.

Table 15–12. Counter Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L].[R]}$	I1: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Explicit
	I2$_{[1]}$	I2: in STD_LOGIC	
	I3$_{[1]}$	I3: in STD_LOGIC	
	I4$_{[1]}$	I4: in STD_LOGIC	
	I5$_{[1]}$	I5: in STD_LOGIC	
	I6$_{[1]}$	I6: in STD_LOGIC	
O	O1$_{[L].[R]}$	O1: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Explicit

Notes to Table 15–12:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Differentiator

The `Differentiator` block is a signed integer differentiator with the equation:

$q(n) = d(n) - d(n-D)$

where D is the delay parameter.

Use this block for DSP functions such as CIC filters.

The equation $1-z^{-D}$ describes the transfer function that the `Differentiator` block implements.

Table 15–13 shows the `Differentiator` block inputs and outputs.

Table 15–13. Differentiator Block Inputs and Outputs

Signal	Direction	Description
d	Input	Data input.
ena	Input	Optional clock enable.
sclr	Input	Optional synchronous clear.
q	Output	Result.

Table 15–14 shows the `Differentiator` block parameters.

Table 15–14. Differentiator Block Parameters

Name	Value	Description
Number of Bits	>= 1 (Parameterizable)	Specify the number of bits.
Depth	Any positive number (Parameterizable)	Specify the depth of the differentiator register.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear input (sclr).

Table 15–15 shows the `Differentiator` block I/O formats.

Table 15–15. Differentiator Block I/O Formats [(1)]

I/O	Simulink [(2), (3)]	VHDL	Type [(4)]
I	I1$_{[L1].[0]}$ I2$_{[1]}$ I3$_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC	Explicit
O	O1$_{[L1].[0]}$	O1: out STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0)	Explicit

Notes to Table 15–15:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 15–4 shows an example with the `Differentiator` block.

Figure 15–4. Differentiator Block Example
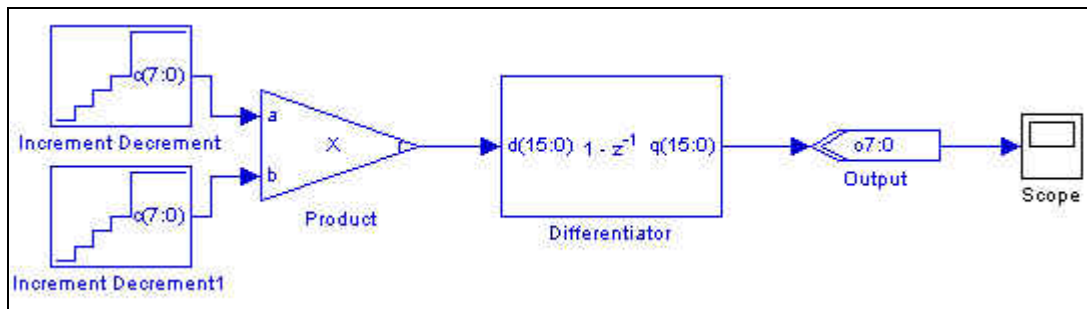
Divider

The `Divider` block takes a numerator and a denominator and returns the quotient and a remainder with the equation:

$a = b \times q + r$.

q and r are undefined if b is zero.

☞ Dividing a maximally negative number by a minimally negative one (-1 if using signed integers), outputs a truncated answer.

The numerator and denominator inputs can have different widths but convert to the specified bit width.

Table 15–16 shows the `Divider` block inputs and outputs.

Table 15–16. Divider Block Inputs and Outputs

Signal	Direction	Description
a	Input	Numerator.
b	Input	Denominator.
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
q	Output	Quotient.
r	Output	Remainder.

Table 15–17 shows the `Divider` block parameters.

Table 15–17. Divider Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The bus number format that you want to use for the divider.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point.

Table 15–17. Divider Block Parameters

Name	Value	Description
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This option applies only to signed fractional formats.
Number of Pipeline Stages	0 to number of bits (Parameterizable)	When non-zero, adds pipeline stages to increase the data throughput. The clock enable and asynchronous clear ports are available only if the block is registered (that is, if the number of pipeline stages is greater than or equal to 1).
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (`aclr`).

Table 15–18 shows the `Divider` block I/O formats.

Table 15–18. Divider Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[L].[R]	I1: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	
	I2[L].[R]	I2: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Explicit
	I3[1]	I3: in STD_LOGIC	Explicit
	I4[1]	I4: in STD_LOGIC	
O	O1[L].[R]	O1: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Explicit
	O2[L].[R]	O2: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Explicit

Notes to Table 15–18:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 15–5 shows an example with the `Divider` block.

Figure 15–5. Divider Block Example
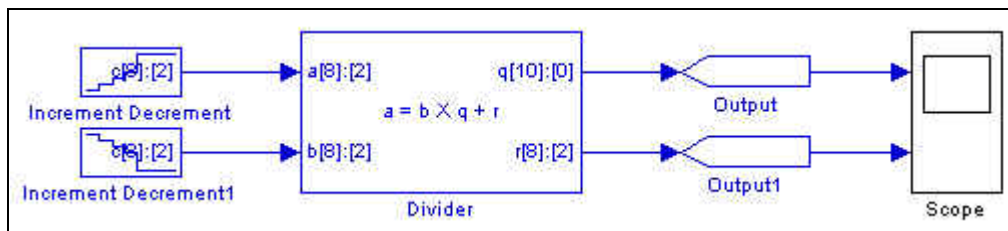
DSP

The `DSP` block consists of one to four multipliers feeding a parallel adder. It is equivalent to the Multiply Add block but exposes extra features (including chaining) that are available only on Stratix IV DSP blocks.

The `DSP` block accepts one to four pairs of multiplier inputs *a* and *b*. The operands in each pair are multiplied together. The second and fourth multiplier outputs can optionally be added or subtracted from the total.

The following equation expresses the block function:

$$res = a_0 \times b_0 \pm a_1 \times b_1 \ [+ a_2 \times b_2 \ [\pm a_3 \times b_3]] \ [+ chainin]$$

If there are four multipliers and the input bit widths are both less than or equal to 18, you can optionally enable a chainout adder output (`chainout`) instead of the normal output (`res`).

If there are four multipliers and the input bit widths are both equal to 18, you can enable a chainout adder input (`chainin`). Only drive this `chainin` port from the `chainout` output of a `DSP` block at the preceding stage.

Other features include:

■ Parameterizable input and output data widths

■ Optional asynchronous clear and clock enable inputs

■ Optional accumulator synchronous load input

■ Optional shiftin instead of an `a` input

■ Optional shift out from the a input of the last multiplier

■ Optional saturation overflow outputs

■ Optional registers to pipeline the adder and chainout adder

■ Optional accumulator mode

For more information about multiplier or adder operations, refer to the *altmult_add Megafunction User Guide*.

Table 15–19 shows the `DSP` block inputs and outputs.

Table 15–19. DSP Block Inputs and Outputs

Signal	Direction	Description
a0–a3	Input	Operand a.
b0–b3	Input	Operand b.
ena	Input	Optional clock enable.
chainin	Input	Optional input bus from the preceding stage. [1]
zero_chainout	Input	Optional reset to zero for the chainout value.
aclr	Input	Optional asynchronous clear.
accum_sload	Input	Optional accumulator synchronous load input.
res	Output	Result.
shiftouta	Output	Optional shift out from A input of last multiplier.
overflow	Output	Optional saturation overflow output.
chainout	Output	Optional chainout output. (Replaces the res output when enabled.)

Note to Table 15–19:

(1) Use the `chainin` port to feed the adder result (`chainout`) from a previous stage. Do not use for any other signal.

Figure 15–6 shows a basic multiplier or adder with two inputs where the product is subtracted.

Figure 15–6. Basic 2-Input Multiplier or Adder
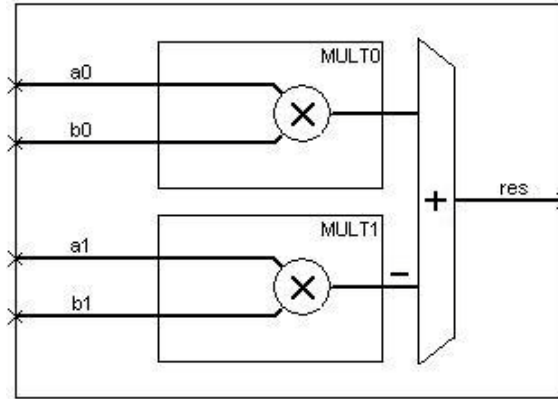
Figure 15–7 shows a 4-input multiplier or adder with shiftin inputs, registered outputs, rounding and saturation enabled, a chainout adder and saturation overflow outputs.

Figure 15–7. 4-Input Multiplier or Adder with Chainout Adder
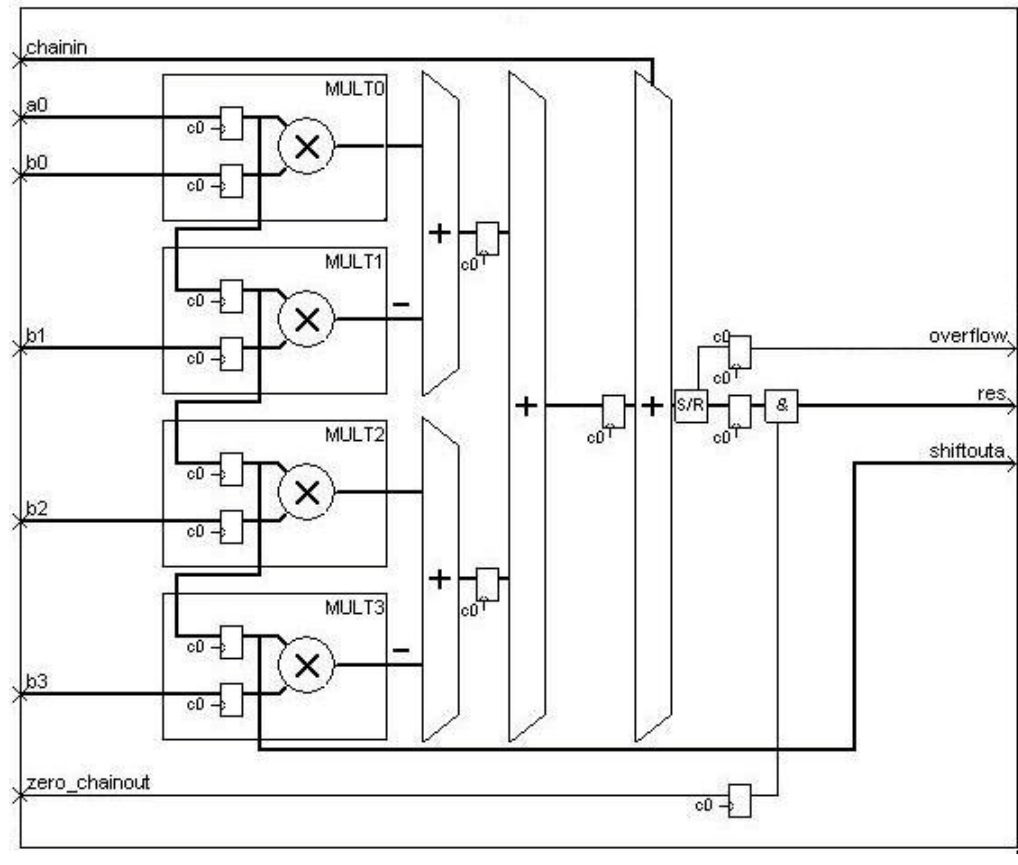
Table 15–20 shows the `DSP` block parameters.

Table 15–20. DSP Block Parameters

Name	Value	Description
Number of Multipliers	1, 2, 3, 4	The number of multipliers you want to feed the adder.
Bus Type	Signed Integer, Unsigned Integer, Signed Fractional	The number format you want to use for the bus.
a Inputs [number of bits].[]	>= 0 (Parameterizable)	Specify the number of data `a` input bits to the left of the binary point, including the sign bit.
a Inputs [].[number of bits]	>= 0 (Parameterizable)	Specify the number of data `a` input bits to the right of the binary point. This option applies only to signed fractional formats.
b Inputs [number of bits].[]	>= 0 (Parameterizable)	Specify the number of data `b` input bits to the left of the binary point, including the sign bit.
b Inputs [].[number of bits]	>= 0 (Parameterizable)	Specify the number of data `b` input bits to the right of the binary point. This option applies only to signed fractional formats.
Connect Multiplier Input a to shiftin	On or Off	Turn on to connect the multiplier input `a` to shiftin from the previous multiplier. The design uses separate inputs for each multiplier.)
Use Shiftout from a Input of Last Multiplier	On or Off	Turn on to create a `shiftouta` output from the `a` input of the last multiplier.
Output Operation on First Multiplier Pair	ADD, SUB	Add or subtract the product of the first multiplier pair.
Output Operation on Second Multiplier Pair	ADD, SUB	Add or subtract the product of the second multiplier pair.
Enable Accumulator Mode	On or Off	Turn on to enable accumulator mode. When this option is on, you can select the accumulator direction and use the optional `accum_sload` input.
Accumulator Direction	ADD, SUB	Add or subtract values in the accumulator.
Use Accumulator Synchronous Load Input	On or Off	Turn on to use the optional `accum_sload` input.
Use Chainout Adder Input (chainin)	On or Off	Turn on to use the `chainin` input for the chainout adder to add the result from a previous stage. This option is available only if the input bit widths are less than or equal to 18 and the number of multipliers is 4.
Use Chainout Adder Output (chainout)	On or Off	Turn on to use the `chainout` output from the chainout adder output instead of the `res` output. This option is available only if the input bit widths are less than or equal to 18 and the number of multipliers is 4.
Use Zero Chainout Input	On or Off	Turn on to use the `zero_chainout` input, which dynamically sets the chainout value to zero.
Full Resolution for Output Result	On or Off	When on, the multiplier output bit width is full resolution. When off, you can specify a different output width. Rounding and saturation are available for certain input/output type combinations.
Output [number of bits].[]	>= 0 (Parameterizable)	Specify the number of data output bits to the left of the binary point, including the sign bit.
Output [].[number of bits]	>= 0 (Parameterizable)	Specify the number of data output bits to the right of the binary point. This option applies only to signed fractional formats.
Output Rounding Operation Type	None (truncate), Nearest Integer, Nearest Even	You can disable rounding (truncate), round to the nearest integer or round to the nearest even.

Table 15–20. DSP Block Parameters

Name	Value	Description
Output Saturation Operation Type	None (wrap), Symmetric, Asymmetric	You can disable (wrap), or enable saturation. Symmetric saturation specifies that the absolute value of the maximum negative number is equal to the maximum positive number. Asymmetric saturation specifies that the absolute value of the maximum negative number is 1 greater than the maximum positive number. Do not enable rounding unless you have enabled saturation.
Use Output Overflow Port	On or Off	Turn on to use the overflow output for the saturation unit.
Register Data Inputs to the Multiplier(s)	On or Off	Turn on to create registers at the data inputs to the multiplier. (Always on if in shiftin mode.)
Register Output of the Multiplier	On or Off	Turn on to create a register at the data output from the multiplier.
Register Output of the Adder	On or Off	Turn on to create a register at the output of the adder. (Always on if accumulator mode is enabled.)
Register Chainout Adder	On or Off	Turn on to create a register at the output of the chainout adder (if it is used).
Register Shiftout	On or Off	Registers the shiftouta output (if it is used).
Use Enable Port	On or Off	Turn on to use the clock enable input (ena) if using registers.
Use User Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (aclr) if using registers.
OR Asynchronous Clear Input with Global Aclr	On or off	This parameter turns on, if you enable **User Asyncrounous Clear input (aclr)**. If parameter value is on, the user asynchrounous clear signal is ORed with global clear signal before it is used. If off, global clear is ignored, and only user clear is used. The default value for the parameter is on (as DSP Builder does reset ORing by default for all blocks).

☞ Compilation in the Quartus Prime software requires that the input bit widths are 18 bits when you use the chainout adder input, output rounding with an output LSB in the range 6 to 21, or output saturation with an output MSB in the range 28 to 43.

Table 15–21 shows the DSP block I/O formats.

Table 15–21. DSP Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[L1].[R1] …. In[L1].[R1] I(n+1)[1] I(n+2)[1] where 3 < n < 9	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) …. In: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I(n+1): in STD_LOGIC I(n+2): in STD_LOGIC where 3 < n < 9	Explicit … Explicit

Table 15–21. DSP Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
O	O1$_{2 \times [L1]}$+ ceil(log2(n))$_{.2}$ $_{\times [R1]}$	O1: out STD_LOGIC_VECTOR(((2 x L1) + ceil(log2(n)) + (2 x R1) - 1) DOWNTO 0)	Implicit

Notes to Table 15–21:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 15–8 shows an example of a basic low-pass filter with two DSP blocks.

Figure 15–8. DSP Block Example
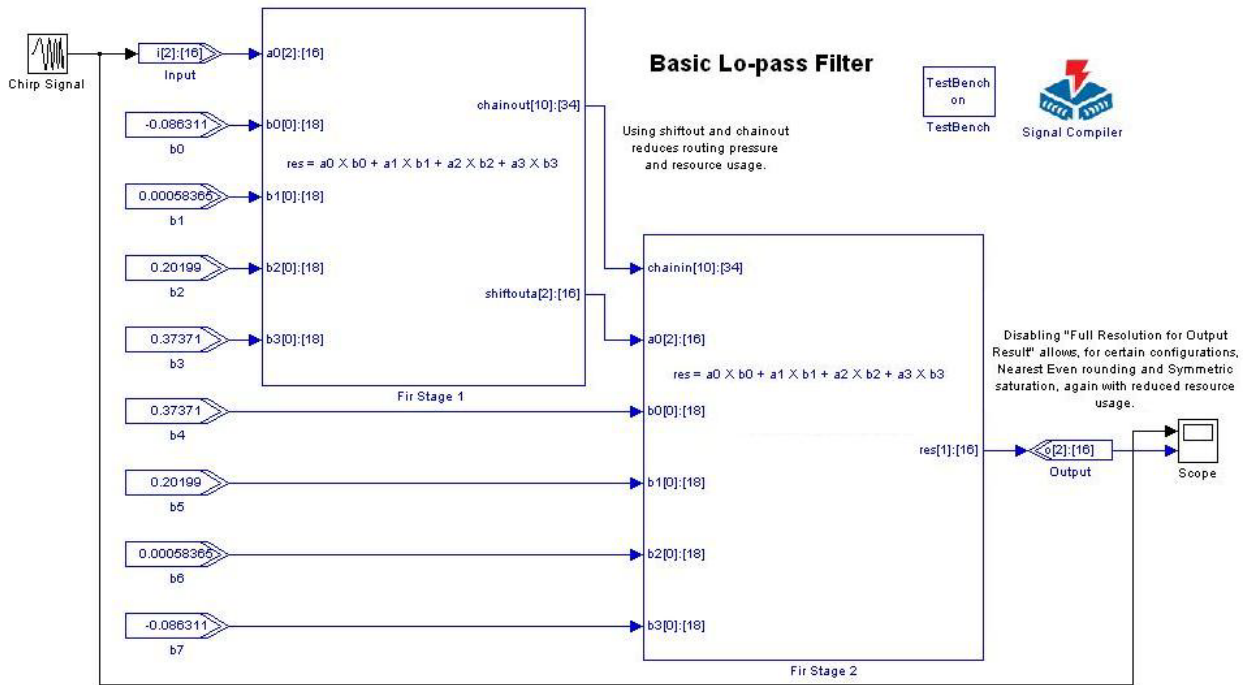
Gain

The Gain block generates its output by multiplying the signal input by a specified gain factor. You must enter the gain as a numeric value in the Gain block parameter field. The gain factor must be a scalar.

☞ The Simulink software also provides a Gain block. If you use the Simulink Gain block in your model, you can use it only for simulation; Signal Compiler cannot convert it to HDL.

Table 15–22 shows the Gain block inputs and outputs.

Table 15–22. Gain Block Inputs and Outputs

Signal	Direction	Description
d	Input	Data input.
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
<unnamed>	Output	Result.

Table 15–23 shows the Gain block parameters.

Table 15–23. Gain Block Parameters

Name	Value	Description
Gain Value	User Defined	Specify the gain value you want to use as a decimal number (or an expression that evaluates to a decimal number). The gain is masked to the number format (bus type) you select.
Map Gain Value to Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The bus number format you want to use for the gain value.
[Gain value number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point, including the sign bit.
[].[Gain value number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This option applies only to signed fractional formats.
Number of Pipeline Stages	>= 0 (Parameterizable)	The number of pipeline delay stages. The **Clock Phase Selection** and **Optional Ports** options are available only if the block is registered (that is, if the number of pipeline stages is greater than or equal to 1).
Clock Phase Selection	User Defined	Specify the phase selection with a binary string, where a 1 indicates the phase in which the block is enabled. For example: 1—The block is always enabled and captures all data passing through the block (sampled at the rate 1). 10—The block is enabled every other phase and every other data (sampled at the rate 1) passes through. 0100—The block is enabled on the second phase of and only the second data of (sampled at the rate 1) passes through. That is, the data on phases 1, 3, and 4 do not pass through the block.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (aclr).
Use LPM	On or Off	This parameter is for synthesis. When on, the Gain block is mapped to the LPM_MULT library of parameterized modules (LPM) function and the VHDL synthesis tool uses the Altera LPM_MULT implementation.

Table 15–24 shows the Gain block I/O formats.

Table 15–24. Gain Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type
I	$I1_{[L1].[R1]}$ $I2_{[1]}$ $I3_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC	Implicit [4]
O	$O1_{[L1 + LK].2*max(R1,RK)]}$ [5]	O1: out STD_LOGIC_VECTOR({L1+LK+2*max(R1,RK)-1} DOWNTO 0)	Implicit

Notes to Table 15–24:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

(5) K is the gain constant with the format $K_{[LK].[RK]}$

Figure 15–9 shows an example with the Gain block.

Figure 15–9. Gain Block Example
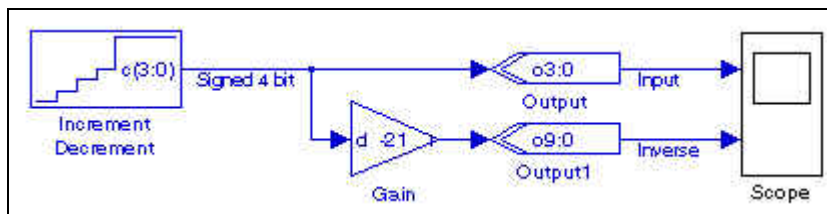
Increment Decrement

The Increment Decrement block increments or decrements a value in time. The output is a signed integer, unsigned integer, or signed binary fractional number. For all number formats, the counting sequence increases or decreases by the smallest representable value; for integer types, the value always changes by 1.

Table 15–25 shows the Increment Decrement block inputs and outputs.

Table 15–25. Increment Decrement Block Inputs and Outputs

Signal	Direction	Description
ena	Input	Optional clock enable.
sclr	Input	Optional synchronous clear.
c	Output	Result.

Table 15–26 shows the `Increment Decrement` block parameters.

Table 15–26. Increment Decrement Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format you want to use for the bus.
<number of bits>.[]	>= 0 (Parameterizable)	Select the number of bits to the left of the binary point, including the sign bit.
[].<number of bits>	>= 0 (Parameterizable)	Select the number of bits to the right of the binary point. This option applies only to signed fractional formats.
Direction	Increment, Decrement	Count up or down.
Starting Value	User Defined (Parameterizable)	Enter the value with which to begin counting. This value is the initial output value of the block after a reset.
Clock Phase Selection	User Defined	Specify the phase selection with a binary string, where a 1 indicates the phase in which the block is enabled. For example: `1`—The block is always enabled and captures all data passing through the block (sampled at the rate 1). `10`—The block is enabled every other phase and every other data (sampled at the rate 1) passes through. `0100`—The block is enabled on the second phase of and only the second data of (sampled at the rate 1) passes through. That is, the data on phases 1, 3, and 4 do not pass through the block.
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined	Specify the clock signal name.
Use Enable Port	On or Off	Turn on if you want to use the clock enable input (`ena`).
Use Synchronous Clear Port	On or Off	Turn on if you want to use the synchronous clear input (`sclr`).

Table 15–27 shows the `Increment Decrement` block I/O formats.

Table 15–27. Increment Decrement Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
I	I1[1] I2[1]	I1: in STD_LOGIC I2: in STD_LOGIC	
O	O1[LP].[RP]	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 15–27:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 15–10 shows an example with the `Increment Decrement` block.

Figure 15–10. Increment Decrement Block Example
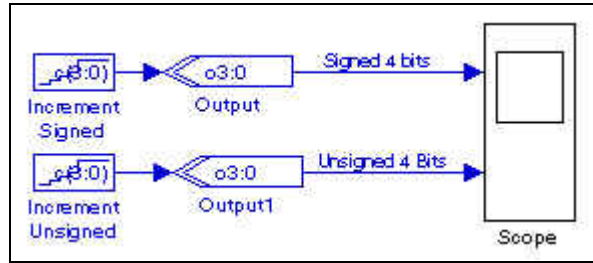
Integrator

The `Integrator` block is a signed integer integrator with the equation:

$q(n+D) = q(n) + d(n)$

where D is the delay parameter.

Use this block for DSP functions such as CIC filters.

The equation $z^{-D}/(1-z^{-D})$ describes the transfer function that the `Integrator` block implements. This behavior of this transfer function is slightly different from the more typical $1/(1-z^{-D})$.

Figure 15–11 shows the block diagrams for these functions.

Figure 15–11. Integrator Transfer Functions
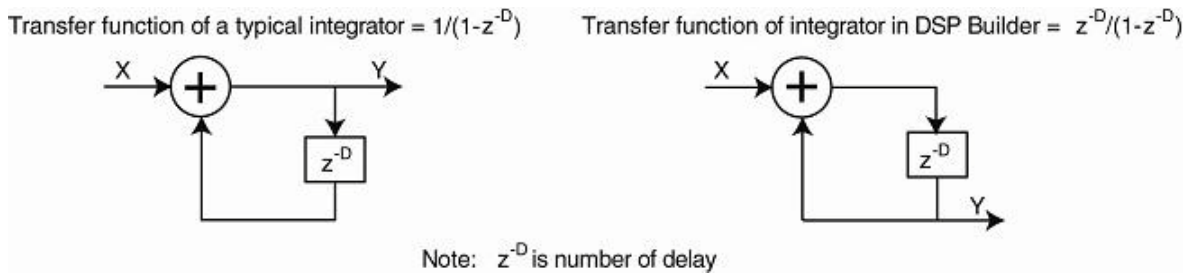
The magnitude response of these two functions is the same although their phase response is different. For the typical integrator function, $1/(1-z^{-D})$, there is an impulse on the output at time = 0, whereas the output delays by a factor of D for the $z^{-D}/(1-z^{-D})$ function that the DSP Builder integrator uses.

This behavior effectively registers the output and gives a better F_{max} performance compared to the typical function where if you chained a row of n integrators together, it is equivalent to n unregistered adder blocks in a row, and is slow in hardware.

Table 15–28 shows the `Integrator` block inputs and outputs.

Table 15–28. Integrator Block Inputs and Outputs

Signal	Direction	Description
d	Input	Data input.
ena	Input	Optional clock enable.
sclr	Input	Optional synchronous clear.
q	Output	Result.

Table 15–29 shows the `Integrator` block parameters.

Table 15–29. Integrator Block Parameters

Name	Value	Description
Number of Bits	>= 1 (Parameterizable)	Specify the number of bits.
Depth	A positive number (Parameterizable)	Specify the depth of the integrator register.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear input (sclr).

Table 15–30 shows the `Integrator` block I/O formats.

Table 15–30. Integrator Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[0]}$ $I2_{[1]}$ $I3_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0) I2: STD_LOGIC I3: STD_LOGIC	Explicit
O	$O1_{[L1].[0]}$	O1: out STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0)	Explicit

Notes to Table 15–30:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 15–12 shows an example of the `Integrator` Block.

Figure 15–12. Integrator Block Design Example
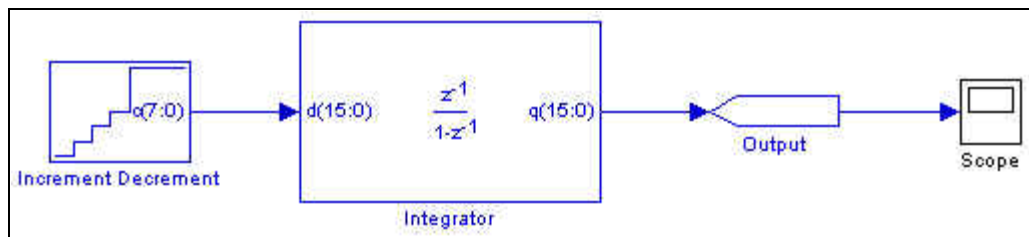
Magnitude

The scalar `Magnitude` block returns the absolute value of the incoming signed binary fractional bus.

The `Magnitude` block has no parameters.

Table 15–31 shows the `Magnitude` block I/O formats.

Table 15–31. Magnitude Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[L1].[R1]	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit
O	O1[L1].[R1]	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit

Notes to Table 15–31:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 15–13 shows an example with the `Magnitude` block.

Figure 15–13. Magnitude Block Example
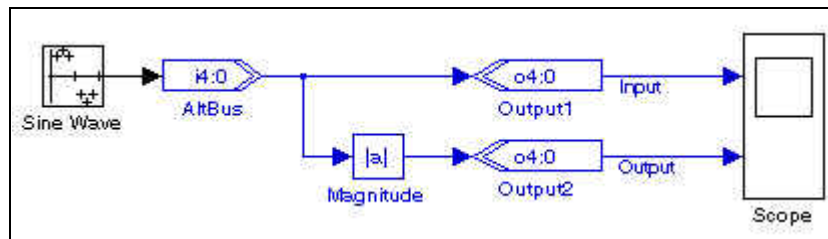
Multiplier

The `Multiplier` block supports two scalar inputs (no multidimensional Simulink signals). Operand *a* is multiplied by operand *b* and the result *r* output as the following equation shows:

$$r = a \times b$$

The differences between the `Multiplier` block and the Product block are:

■ The `Product` block supports clock phase selection while the `Multiplier` block does not.

■ The `Product` block uses implicit input port data widths that it inherits from the signals' sources, whereas the `Multiplier` block uses explicit input port data widths that you must specify as parameters.

■ The `Product` block allows you to use the LPM multiplier megafunction, whereas the `Multiplier` block always uses the LPM.

Table 15–32 shows the `Multiplier` block inputs and outputs.

Table 15–32. Multiplier Block Inputs and Outputs

Signal	Direction	Description
a	Input	Operand a.
b	Input	Operand b.
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
r	Output	Result r.

Table 15–33 lists the parameters for the `Multiplier` block.

Table 15–33. Multiplier Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The bus number format to use for the `Multiplier` block.
Input [number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point for input a (or both input signals if set to have the same width).
Input [].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point for input a (or both input signals if set to have the same width). This option applies only to signed fractional formats.
Number of Pipeline Stages	>= 0 (Parameterizable)	The number of pipeline stages. The `ena` and `aclr` ports are available only if the block is registered (that is, if the number of pipeline stages is greater than or equal to 1).
Both Inputs Have Same Bit Width	On or Off	Turn on if you want input a and input b to have the same bit width. When off, additional fields are available to specify the number of bits to the left and right of the binary point for input b.
Input b [number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point for input b.
Input b [].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point for input b. This option applies only to signed fractional formats.
Full Resolution for Output Result	On or Off	When on, the multiplier output bit width is full resolution. When off, you can specify the number of bits for the output.
Output MSB	>= 0 (Parameterizable)	Specify the number of MSBs in the output for an integer bus.
Output LSB	>= 0 (Parameterizable)	Specify the number of LSBs in the output for an integer bus.
Output [number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point for the output r. This option applies only to signed fractional formats.
Output [].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point for the output r. This option applies only to signed fractional formats.
Use Dedicated Circuitry	AUTO, YES, NO	Use dedicated multiplier circuitry (if supported by your target device). A value of **AUTO** means that the Quartus II software uses the dedicated multiplier circuitry based on the width of the multiplier.

Table 15–33. Multiplier Block Parameters

Name	Value	Description
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Asynchronous Clear Port	On or Off	Turn on to use the synchronous clear input (`aclr`).

Table 15–34 shows the `Multiplier` block I/O formats.

Table 15–34. Multiplier Block Input/Output Ports [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
I	I1$_{[L].[R]}$	I1: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	
	I2$_{[L].[R]}$	I2: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Explicit
	I3$_{[1]}$	I3: STD_LOGIC	Explicit
	I4$_{[1]}$	I4: STD_LOGIC	
O	O1$_{[Lo].[Ro]}$	O1: out STD_LOGIC_VECTOR({Lo + Ro - 1} DOWNTO 0)	Explicit
	O2$_{[Lo].[Ro]}$	O2: out STD_LOGIC_VECTOR({Lo + Ro - 1} DOWNTO 0)	Explicit

Notes to Table 15–34:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 15–14 shows an example with the `Multiplier` block.

Figure 15–14. Multiplier Block Example
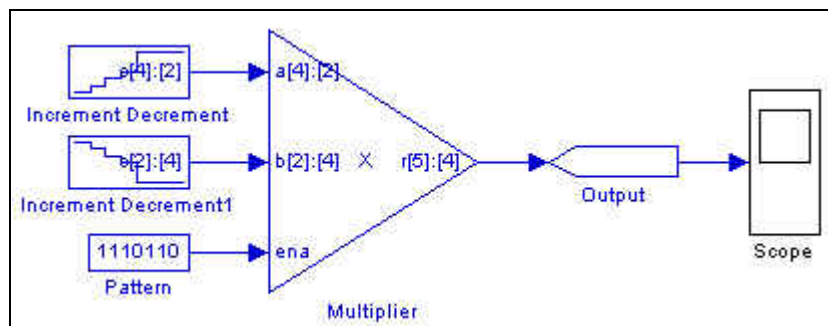
For more information about multiplier operations, refer to the *Multiplier Megafunction User Guide*.

Multiply Accumulate

The `Multiply Accumulate` block consists of a single multiplier feeding an accumulator, which performs the calculation $y += a \times b$.

The input is signed integer, unsigned integer, or signed binary fractional formats.

Table 15–35 shows the `Multiply Accumulate` block inputs and outputs.

Table 15–35. Multiply Accumulate Block Inputs and Outputs

Signal	Direction	Description
a	Input	Operand A.
b	Input	Operand B.
sload	Input	Synchronous load signal.
addsub	Input	Optional accumulator direction (1= add, 0 = subtract).
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
y	Output	Result.

Table 15–36 shows the `Multiply Accumulate` block parameters.

Table 15–36. Multiply Accumulate Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format you want to use for the bus.
Input A [number of bits].[]	>= 0 (Parameterizable)	Specify the number of data input bits to the left of the binary point for operand A, including the sign bit.
Input A [].[number of bits]	>= 0 (Parameterizable)	Specify the number of data input bits to the right of the binary point for operand A. This option applies only to signed fractional formats.
Input B [number of bits].[]	>= 0 (Parameterizable)	Specify the number of data input bits to the left of the binary point for operand B, including the sign bit.
Input B [].[number of bits]	>= 0 (Parameterizable)	Specify the number of data input bits to the right of the binary point for operand B. This option applies only to signed fractional formats.
Output Result number of bits	>= 0 (Parameterizable)	Specify the number of output bits.
Pipeline Register	None, Data Inputs, Multiplier Output, Data Inputs and Multiplier	Add pipelining to the data inputs, multiplier output, both, or neither.
Use Dedicated Multiplier Circuitry	AUTO, YES, NO	Select **AUTO** to automatically implement the functionality in DSP blocks. Select **YES** or **NO** to explicitly enable or disable this option. If your target device does not support DSP blocks or you select **NO**, the functionality implements in logic elements.
Accumulator Direction	Add, Subtract	Add or subtract the result of the multiplier.
Use Add/Subtract Port	On or Off	Turn on to use the direction input (addsub).
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (aclr).

Table 15–37 shows the `Multiply Accumulate` block I/O formats.

Table 15–37. Multiply Accumulate Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
I	I1$_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	
	I2$_{[L2].[R2]}$	I2: in STD_LOGIC_VECTOR({L2 + R2 - 1} DOWNTO 0)	
	I3$_{[1]}$	I3: in STD_LOGIC	Explicit
	I4$_{[1]}$	I4: in STD_LOGIC	Explicit
	I5$_{[1]}$	I5: in STD_LOGIC	
	I6$_{[1]}$	I6: in STD_LOGIC	
O	O1$_{[L0].[R0]}$	O1: out STD_LOGIC_VECTOR({L0 + R0 - 1} DOWNTO 0)	Explicit

Notes to Table 15–37:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

The `sload` input controls the accumulator feedback path. If the accumulator is adding and `sload` is high, the multiplier output is loaded into the accumulator. If the accumulator is subtracting, the opposite (negative value) of the multiplier output is loaded into the accumulator.

Figure 15–15 shows an example with the `Multiply Accumulate` block.

Figure 15–15. Multiply Accumulate Block Example
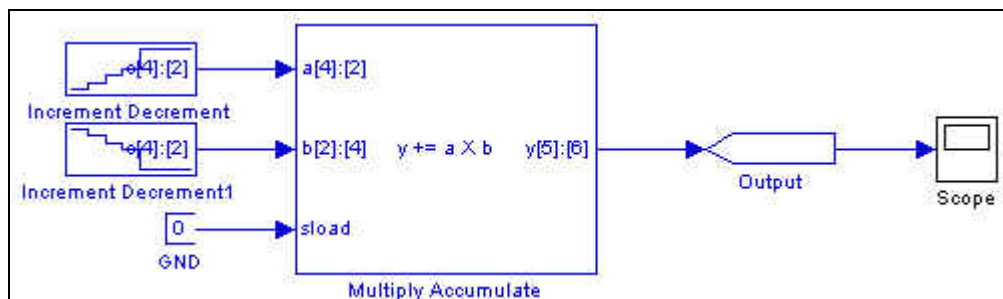
Multiply Add

The `Multiply Add` block consists of two, three, or four multiplier pairs feeding a parallel adder. The operands in each pair are multiplied together and the second and fourth multiplier outputs can optionally be added to or subtracted from the total.

The following equation expresses the block function:

$y = a0{\times}b0 \pm a1{\times}b1\ [+\ a2{\times}b2\ [\pm\ a3{\times}b3]]$

The operand *b* inputs can optionally be hidden and instead have constant values assigned in the **Block Parameters** dialog box.

The input is a signed integer, unsigned integer, or signed binary fractional formats.

Table 15–38 shows the `Multiply Add` block inputs and outputs.

Table 15–38. Multiply Add Block Inputs and Outputs

Signal	Direction	Description
a0–a3	Input	Operand a.
b0–b3	Input	Operand b.
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear
y	Output	Result.

Table 15–39 shows the `Multiply Add` block parameters.

Table 15–39. Multiply Add Block Parameters

| Name	Value	Description
Number of Multipliers	2, 3, 4	The number multipliers you want to feed the adder.
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format you want to use for the bus.
Input [number of bits].[]	>= 0 (Parameterizable)	Specify the number of data input bits to the left of the binary point, including the sign bit.
Input [].[number of bits]	>= 0 (Parameterizable)	Specify the number of data input bits to the right of the binary point. This option applies only to signed fractional formats.
Adder Mode	Add Add, Add Sub, Sub Add, Sub Sub	The operation mode of the adder. ■ **Add Add:** Adds the products of each multiplier. ■ **Add Sub**: Adds the second product and subtracts the fourth. ■ **Sub Add**: Subtracts the second product and adds the fourth. ■ **Sub Sub:** Subtracts the second and fourth products.
Pipeline Register	No Register, Inputs Only, Multiplier Only, Adder Only, Inputs and Multiplier, Inputs and Adder, Multiplier and Adder, Inputs Multiplier and Adder	The elements to pipeline. The clock enable and asynchronous clear ports are available only if the block is registered.
Use Dedicated Circuitry	On or Off	If you target devices that support DSP blocks, turn on to implement the functionality in DSP blocks instead of with logic elements. This option is not available if you select the **Unsigned Integer** bus type.
One Input is Constant	On or Off	Turn on to assign the operand b inputs to constant values. Use this option with the **Constant Values** parameter but is not available when you enable **Use Dedicated Circuitry**.
Constant Values	User Defined	Type the constant values in this box as a MATLAB array. This option is available only if **One Input is Constant** is on.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (aclr).

Table 15–40 shows the `Multiply Add` block I/O formats.

Table 15–40. Multiply Add Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[L1].[R1] …. Ii[L1].[R1] … In[L1].[R1] I(n+1)[1] I(n+2)[1] where 3 < n < 9	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) … Ii: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) …. In: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I(n+1): in STD_LOGIC I(n+2): in STD_LOGIC where 3 < n < 9	Explicit ... Explicit ... Explicit
O	O1$_{2 \times [L1]+ ceil(log2(n)).2 \times [R1]}$	O1: out STD_LOGIC_VECTOR({(2 x L1) + ceil(log2(n)) + (2 x R1) - 1} DOWNTO 0)	Implicit

Notes to Table 15–40:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a **Bus Conversion** block to set the width.

Figure 15–16 shows an example with the `Multiply Add` block.

Figure 15–16. Multiply Add Block Example
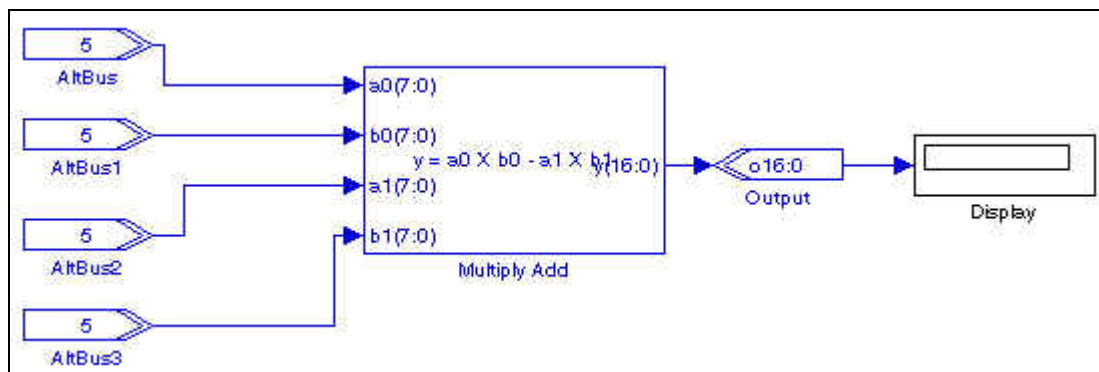
Parallel Adder Subtractor

The `Parallel Adder Subtractor` block takes any input data type. If the input widths are not the same, Signal Compiler sign extends the buses so that they match the largest input width. The generated VHDL has an optimized, balanced adder tree.

Table 15–41 shows the `Parallel Adder Subtractor` block inputs and outputs.

Table 15–41. Parallel Adder Subtractor Block Inputs and Outputs

Signal	Direction	Description
`data0-dataN`	Input	Operands.
`ena`	Input	Optional clock enable.
`aclr`	Input	Optional asynchronous clear
`r`	Output	Result.

Table 15–42 shows the `Parallel Adder Subtractor` block parameters.

Table 15–42. Parallel Adder Subtractor Block Parameters

Name	Value	Description
Number of Inputs	>= 2	The number of inputs you want to use.
Add (+) Sub (−)	User Defined	Specify addition or subtraction operation for each port with the operators + and −. For example + − + implements a − b + c for 3 ports. However, two consecutive subtractions, (− −) are not legal. Missing operators are assumed to be +.
Enable Pipeline	On or Off	When on, DSP Bu idler registers the output from each stage in the adder tree, resulting in a pipeline length that is equal to `ceil(log2(number of inputs))`.
Clock Phase Selection	User Defined	When you enable a pipeline, you can indicate the phase selection with a binary string, where a 1 indicates the phase in which the block is enabled. For example: `1`—The block is always enabled and captures all data passing through the block (sampled at the rate 1). `10`—The block is enabled every other phase and every other data (sampled at the rate 1) passes through. `0100`—The block is enabled on the second phase of and only the second data of (sampled at the rate 1) passes through. That is, the data on phases 1, 3, and 4 do not pass through the block.
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (`aclr`).

Table 15–43 shows the `Parallel Adder Subtractor` block I/O formats.

Table 15–43. Parallel Adder Subtractor Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit
	….	…	
	$Ii_{[Li].[Lil]}$	Ii: in STD_LOGIC_VECTOR({Li + Ri - 1} DOWNTO 0)	...
	…	….	Implicit
	$In_{[Ln].[Rn]}$	In: in STD_LOGIC_VECTOR({Ln + Rn - 1} DOWNTO 0)	...
	$I(n+1)_{[1]}$	I(n+1): in STD_LOGIC	Implicit
	$I(n+2)_{[1]}$	I(n+2): in STD_LOGIC	

Table 15–43. Parallel Adder Subtractor Block I/O Formats [(1)]

| I/O	Simulink [(2), (3)]	VHDL	Type [(4)]
O	O1[max(Li) + ceil(log2(n))].[max(Ri)]	O1: out STD_LOGIC_VECTOR({max(Li) + ceil(log2(n)) + max(Ri) - 1} DOWNTO 0)	Implicit

Notes to Table 15–43:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 15–17 shows an example with the Parallel Adder Subtractor block.

Figure 15–17. Parallel Adder Subtractor Block Example
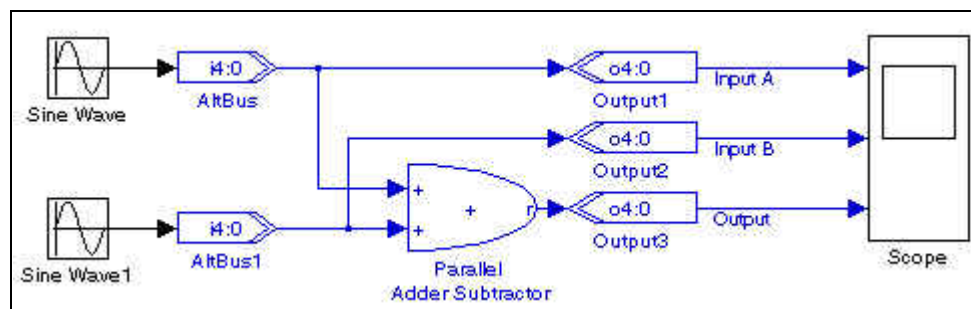
Pipelined Adder

The Pipelined Adder block is a pipelined adder and subtractor that performs the following calculation:

r = a + b + cin (when addsub = 1)

r = a - b + cin -1 (when addsub = 0)

Use the optional ovl port an overflow with signed arithmetic or as a carry out with unsigned arithmetic. For unsigned subtraction, the output is 1 when no overflow occurs.

Table 15–44 shows the Pipelined Adder block inputs and outputs.

Table 15–44. Pipelined Adder Block Inputs and Outputs

Signal	Direction	Description
a	Input	Operand a.
b	Input	Operand b.
cin	Input	Optional carry in.
addsub	Input	Optional control (1= add, 0 = subtract).
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
r	Output	Result r.
ovl	Output	Optional overflow (signed) or carry out (unsigned).

Table 15–45 shows the `Pipelined Adder` block parameters.

Table 15–45. Pipelined Adder Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The bus number format that you want to use.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This option applies only to signed fractional formats.
Number of Pipeline Stages	>= 0 (Parameterizable)	The number of pipeline stages.
Direction	ADD, SUB	Use the block as an adder or subtractor.
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (`aclr`).
Use Carry In Port	On or Off	Turn on to use the carry in input (`cin`).
Use Overflow / Carry Out Port	On or Off	Turn on to use the overflow or carry out output (`ovl`).
Use Direction Port	On or Off	Turn on to use the direction input (`addsub`). 1= add, 0 = subtract.

Table 15–46 shows the `Pipelined Adder` block I/O formats.

Table 15–46. Pipelined Adder Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L].[R]}$	I1: in STD_LOGIC_VECTOR({L + R} DOWNTO 0)	
	$I2_{[L].[R]}$	I2: in STD_LOGIC_VECTOR({L + R} DOWNTO 0)	
	$I3_{[1]}$	I3: in STD_LOGIC	Explicit
	$I4_{[1]}$	I4: in STD_LOGIC	Explicit
	$I5_{[1]}$	I5: in STD_LOGIC	
	$I6_{[1]}$	I6: in STD_LOGIC	
O	$O1_{[L].[R]}$	O1: out STD_LOGIC_VECTOR({L + R} DOWNTO 0)	Explicit
	$O2_{[1]}$	O2: out STD_LOGIC	

Notes to Table 15–46:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 15–18 shows an example with the `Pipelined Adder` block.

Figure 15–18. Pipelined Adder Block Example
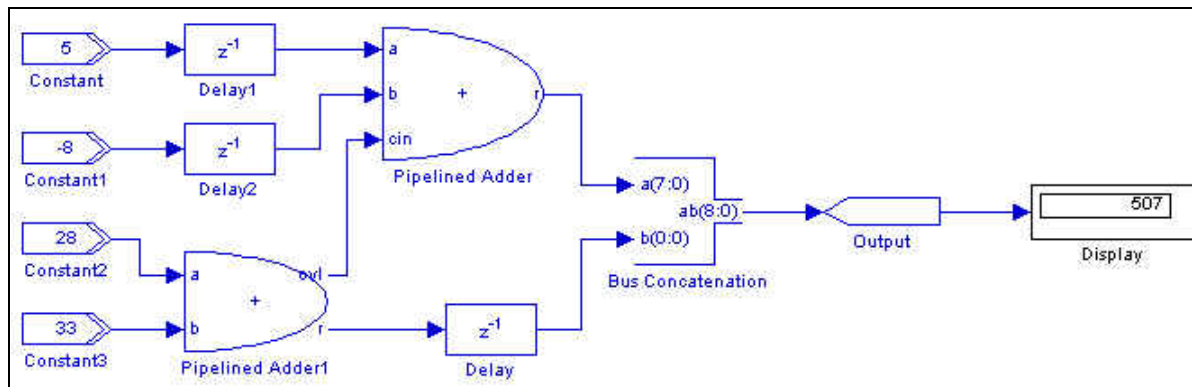
Product

The `Product` block supports two scalar inputs (no multidimensional Simulink signals). Operand *a* is multiplied by operand *b* and the result output on *r* as the following equation shows:

$$r = a \times b$$

The differences between the `Product` block and the Multiplier block are:

■ The `Product` block supports clock phase selection while the `Multiplier` block does not.

■ The `Product` block uses implicit input port data widths that are inherited from the signals' sources, whereas the `Multiplier` block uses explicit input port data widths that you must specify as parameters.

■ The `Product` block allows you to use the LPM multiplier megafunction, whereas the `Multiplier` block always uses the LPM.

☞ The Simulink software also provides a `Product` block. If you use the Simulink `Product` block in your model, you can use it only for simulation. Signal Compiler issues an error and cannot convert the Simulink `Product` block to HDL.

Table 15–47 shows the `Product` block inputs and outputs.

Table 15–47. Product Block Inputs and Outputs

Signal	Direction	Description
a	Input	Operand a.
b	Input	Operand b.
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
r	Output	Result.

Table 15–48 shows the `Product` block parameters.

Table 15–48. Product Block Parameters

Name	Value	Description
Bus Type	Inferred, Signed Integer, Signed Fractional, Unsigned Integer	The bus number format that you want to use. Inferred means that the format is automatically set by the format of the connected signal.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This option applies only to signed fractional formats.
Number of Pipeline Stages	>= 0 (Parameterizable)	The Pipeline represents the delay. The clock enable and asynchronous clear ports are available only if the block is registered (that is, if the number of pipeline stages is greater than or equal to 1).
Clock Phase Selection	User Defined	This option is available only when the **Pipeline** value is greater than 0. Specifies the phase selection with a binary string, where a 1 indicates the phase in which the block is enabled. For example: `1`—The block is always enabled and captures all data passing through the block (sampled at the rate 1). `10`—The block is enabled every other phase and every other data (sampled at the rate 1) passes through. `0100`—The block is enabled on the second phase of and only the second data of (sampled at the rate 1) passes through. That is, the data on phases 1, 3, and 4 do not pass through the block.
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (`aclr`).
Use LPM	On or Off	When on, the `Product` block is mapped to the `LPM_MULT` library of parameterized modules (LPM) function and the VHDL synthesis tool uses the Altera `LPM_MULT` implementation. When off, the VHDL synthesis tool uses the native `*` operator to synthesize the product. If your design does not need arithmetic boundary optimization—such as connecting a multiplier to constant combinational logic or register balancing optimization—the `LPM_MULT` implementation generally yields a better result for both speed and area.
Use Dedicated Circuitry	On or Off	Turn on to use the dedicated multiplier circuitry (if supported by your target device). This option is ignored if not supported by your target device.

Table 15–49 shows the `Product` block I/O formats.

Table 15–49. Product Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[L1].[R1] I2[L2].[R2] I3[1] I4[1]	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L2 + R2 - 1} DOWNTO 0) I3: in STD_LOGIC I4: in STD_LOGIC	Explicit Explicit
O	O1[2×max(L1,L2).[2×max (R1,R2)]	O1: out STD_LOGIC_VECTOR({2×max(L1,L2) + 2×max(R1,R2) - 1} DOWNTO 0)	Implicit

Notes to Table 15–49:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.
(3) I1[L].[R] is an input port. O1[L].[R] is an output port.
(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 15–19 shows an example with the `Product` block.

Figure 15–19. Product Block Example
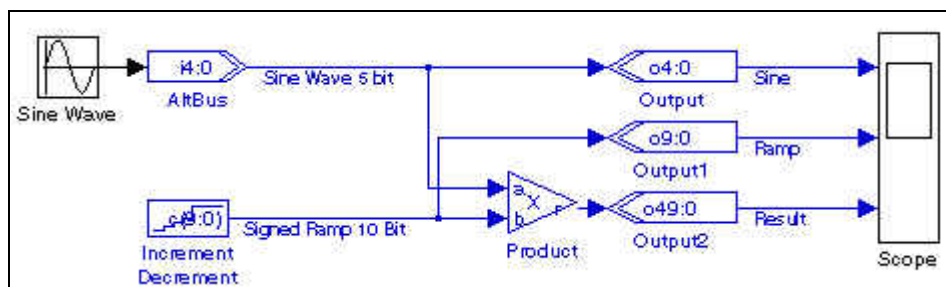
For more information about multiplier operations, refer to the *lpm_mult Megafunction User Guide*.

SOP Tap

The `SOP Tap` block performs a sum of products for two or four taps. Use this block to build two or four tap FIR filters, or cascade blocks to create filters with more taps.

The `SOP Tap` block implements with a multiplier-adder, which has registers on the inputs, multipliers and adders. Thus, the result always lags the input by 3 cycles. The `dout` port is assigned the value of din(n-t) where t is the number of taps. The block has the following equations:

For 2 taps:

$$q(n+3) = c_0(n) \times din(n) + c_1(n) \times din(n-1)$$
$$dout(n+2) = din(n)$$

For 4 taps:

$$q(n+3) = c_0(n) \times din(n) + c_1(n) \times din(n-1) + c_2(n) \times din(n-2) + c_3(n) \times din(n-3)$$
$$dout(n+4) = din(n)$$

Table 15–50 shows the SOP Tap block inputs and outputs.

Table 15–50. SOP Tap Block Inputs and Outputs

Signal	Direction	Description
din	Input	Data input.
c_0, c_1, c_2, c_3	Input	2 or 4 tap coefficients.
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
q	Output	Result.
dout	Output	Shifted input data.

Table 15–51 shows the SOP Tap block parameters.

Table 15–51. SOP Tap Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Unsigned Integer	The bus number format that you want to use for the counter.
Input Number of Bits	>= 0 (Parameterizable)	Specify the number of bits.
Number of Taps	2 or 4	The number of taps.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (aclr).

Table 15–52 shows the SOP Tap block I/O formats.

Table 15–52. SOP Tap Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
I	I1[L].[R] I2[L].[R] ... In[L].[R] I(n+1) I(n+2)	I1: in STD_LOGIC_VECTOR({L + R -1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L + R -1} DOWNTO 0) ... In: in STD_LOGIC_VECTOR({L + R -1} DOWNTO 0) I(n+1): STD_LOGIC I(n+2): STD_LOGIC	Explicit Explicit ... Explicit

Table 15–52. SOP Tap Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
O	O1$_{[2L + cell(log2(N + 1))].[2R]}$ O2	O1: out STD_LOGIC_VECTOR({2L + cell(log2(N + 1)) + 2R - 1} DOWNTO 0) O2: in STD_LOGIC_VECTOR({L + R -1} DOWNTO 0)	Explicit Explicit

Notes to Table 15–52:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 15–20 shows an example with the SOP Tap block.

Figure 15–20. SOP Tap Block Example
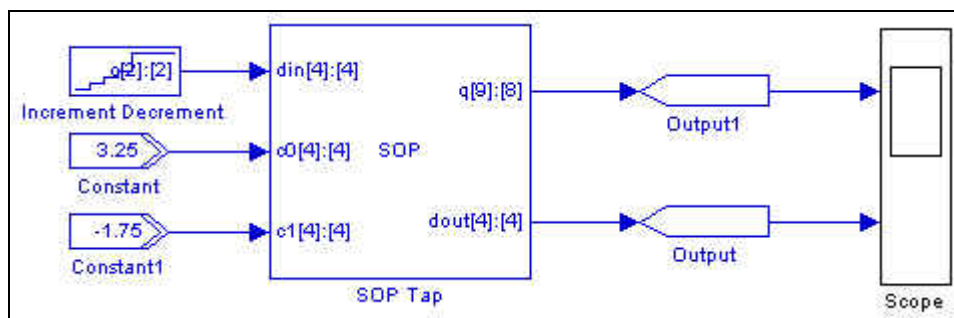
Square Root

The `Square Root` block returns the square root and optional remainder of unsigned integer input data with the equation:

$$q^2 + remainder = d$$

where remainder $<= 2 \times q$

The `Square Root` block supports sequential mode (when the number of pipeline stages> 0) or combinational mode (when the number of pipeline stages = 0).

Assume the radical *d is* an unsigned integer, and that *q* and the *remainder* are always unsigned integers.

Table 15–53 shows the `Square Root` block inputs and outputs.

Table 15–53. Square Root Block Inputs and Outputs

Signal	Direction	Description
d	Input	Data input.
en	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.

Table 15–53. Square Root Block Inputs and Outputs

Signal	Direction	Description
q	Output	Result.
remainder	Output	Optional remainder.

Table 15–54 lists the parameters for the Square Root block.

Table 15–54. Square Root Block Parameters

Name	Value	Description
Input Number of Bits	>= 0 (Parameterizable)	Specify the number of bits of the unsigned input signal.
Number of Pipeline Stages	>= 0 (Parameterizable)	Specify the number of pipeline stages. The computation is sequential when the pipeline is greater than 1 or combinational when the number of pipeline stages is zero. The clock enable and asynchronous clear ports are available only if the number of pipeline stages is greater than or equal to 1.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (aclr).
Use Remainder Port	On or Off	Turn on to use the remainder input (remainder).

Table 15–55 shows the Square Root block I/O formats.

Table 15–55. Square Root Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L].[R]}$ $I2_{[1]}$ $I3_{[1]}$	I1: in STD_LOGIC_VECTOR({L + R} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC	Explicit
O	$O1_{[L].[R]}$ $O2_{[L].[R]}$	O1: out STD_LOGIC_VECTOR({L + R} DOWNTO 0) O2: out STD_LOGIC_VECTOR({L + R} DOWNTO 0)	Explicit

Notes to Table 15–55:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 15–21 shows an example of the Square Root block.

Figure 15–21. Square Root Block Design Example
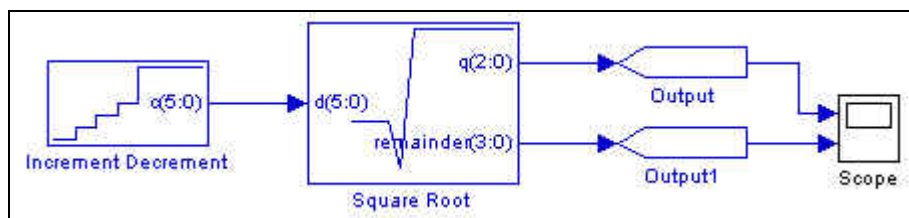
Sum of Products

The Sum of Products block implements the following expression:

$$q = a(0)C0 + ... + a(i)Ci + ... + a(n-1)C_{n-1}$$

where:

- q is the output result

- $a(i)$ is the signed integer input data

- Ci are the signed integer fixed coefficients

- n is the number of coefficients in the range one to eight

Table 15–56 shows the Sum of Products block inputs and outputs.

Table 15–56. Sum of Products Block Inputs and Outputs

Signal	Direction	Description
a(0) to a(n-1)	Input	1 to 8 ports corresponding to the signed integer fixed coefficient values specified in the block parameters.
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
q	Output	Result.

Table 15–57 lists the parameters for the Sum of Products block.

Table 15–57. Sum of Products Block Parameters

Name	Value	Description
Input Data Number of Bits	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point of all input signals.
Number of Coefficients	1–8	The number of coefficients.
Coefficients Number of Bits	>= 1 (Parameterizable)	Specify the number of bits to the left of the binary point of all non-variable coefficients represented as a signed integer.
Signed Integer Fixed-Coefficient Values	Vector (Parameterizable)	Specify the coefficient values for each port as a sequence of signed integers. For example: [-587 -844 -678 -100 367 362 71 -244]
Number of Pipeline Stages	>= 0 (Parameterizable)	Specify the number of pipeline stages.
Full Resolution for Output Result	On or Off	When on, the multiplier output bit width is full resolution. When off, you can specify the number of bits in the output signal and the number of least significant bits (LSBs) truncated from the output signal.
Output Number of Bits	>= 0 (Parameterizable)	Specify the number of bits in the output signal.
Output Truncated LSB	>= 0 (Parameterizable)	Specify the number of LSBs to be truncated from the output signal.

Table 15–57. Sum of Products Block Parameters

Name	Value	Description
FPGA Implementation	Distributed Arithmetic, Dedicated Multiplier Circuitry, Auto	Use a distributed arithmetic, dedicated multiplier or automatically determined implementation.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (aclr).

Table 15–58 shows the Sum of Product block I/O formats.

Table 15–58. Sum of Products Block I/O Formats [1]

| I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L].[0]}$ ... In$_{[L].[0]}$ I(n+1) I(n+2)	I1: in STD_LOGIC_VECTOR({L - 1} DOWNTO 0) ... In: in STD_LOGIC_VECTOR({L - 1} DOWNTO 0) I(n+1): STD_LOGIC I(n+2): STD_LOGIC	Explicit ... Explicit
O	O1$_{[2L + cell(log2(n + 1))].[2R]}$	O1: out STD_LOGIC_VECTOR({2L + cell(log2(n + 1)) + 2R - 1} DOWNTO 0)	Explicit

Notes to Table 15–58:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 15–22 shows an example with the Sum of Product block.

Figure 15–22. Sum of Product Block Example
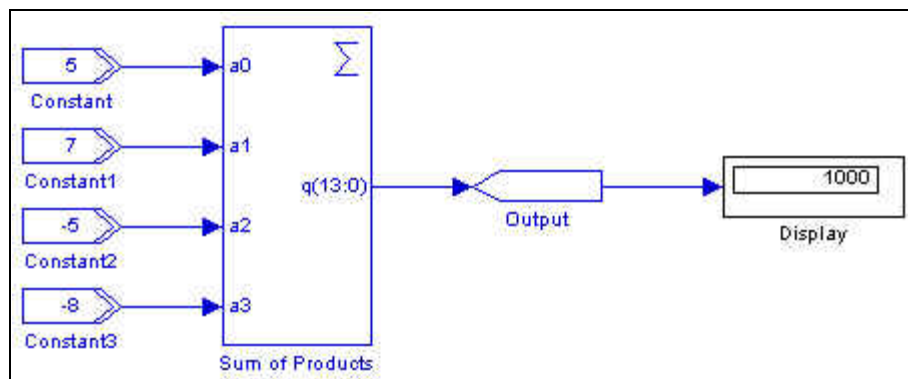
Variable Precision DSP

The Variable Precision DSP block consists of one to four multipliers feeding an adder. It allows preaddition, using a coefficient ROM, and accumulation or chaining. The block is available for the Arria V, Cyclone V, and Stratix V device families.

The block accepts from one to four groups of inputs A, B, C, coef_sel to multipliers. Each set of inputs depends on the parameters you select.

You can select input scan chaining for A inputs only. This feature requires you to enable all input registers. For Stratix V devices, it is available in 18-bit modes only. Enable input scan only when you disable preadder. For Arria and Cyclone V devices, in 18-bit modes input you can combine the scan chain with the preadder. For 27-bit modes, enable it only when you disable the preadder.

You can optionally add or subtract the results of the second and forth multipliers from the result. You can control the operation dynamically by enabling a special control input. The block subtracts the first operand from the second.

For the configurations with input data widths less than 27 (more than 18 in case of one multiplier) and at most two multipliers the chaining and accumulation feature is available.

You can dynamically negate the result of multiplications and additions before passing to accumulation.

Control the accumulation using the special control input. When the accumulation control port is low, load a rounding constant into an accumulator. Otherwise, the block loads 0.

When available, enable the output chaining using chainin and chainout ports. The chainout port on one block can only feed the chainin on another.

The block exposes up to three enable signals. Each register can chose between any of these enable signals.

The block has two dedicated asynchronous clear signals: aclr0 for input registers, aclr1 for output registers.

Parameters

Table 15–59 shows the basic structure parameters.

Table 15–59. basic Structure Parameters

Name	Value	Description
Device Family	Arria V Cyclone V Stratix V	Specifies the target device family.
Use block for	25x18 Complex Mode 18x18+36 Mode Other Modes	Specifies whether the block operates in one of the separately treated modes (25x18 Complex, 18x18+38 modes). For these modes the set of inputs provided for the block differs from the general pattern used for the other blocks. The 25x18 Complex Mode is available only for Stratix V devices.
Number of Multipliers	1, 2, 3, 4	Specifies the number of multipliers. The 3 and 4 options are valid for Stratix V devices only.

For Stratix V devices only, you can configure the block to operate in 25x18 complex mode. In this mode, the block accepts real and imaginary components of two complex operands and outputs real and imaginary components of the multiplication result separately.

For all devices the block can be also configured to operate in 18x18+36 mode, where first two of three data inputs are multiplied together and added with the third one.

Table 15–60 shows the primary data and controls parameters.

Table 15–60. Primary Data and Controls Parameters

Name	Value	Description
Use Enable Inputs	On or off	Specifies whether any of possible three enable signals (if used later) should appear as input for the block. Otherwise, the block ties these signals to ground internally.
Use Asynchrounous Clear Inputs	On or off	Specifies whether block should have dedicated inputs for asynchrounous clear signals.
OR' Asynchrounous Clear Inputs with Global Aclr	On or off	Specifies whether the user asynchrounous signals should be OR-ed with global clear signal before the block uses them. The parameter value is TRUE by the default.
Bus Type	Unsigned Integer Signed Integer Signed Fractional	Type of all data inputs, outputs, and internal values in the block.
A input	[1:36/27]	Bitwidths for A input busses. For Stratix V devices, the maximum allowed value is 36; for Arria V and Cyclone V devices it is 27.
Register A input(s)	On or off	Turn on register A input.
Enable for 'A' Input(s) Register	ENABLE0 ENABLE1 ENABLE2	Specify enable signal for A inputs register.
B input	[1:36/27]	Bitwidths for B input busses. For Stratix V devices, the maximum allowed value is 36; for Arria V and Cyclone V devices it is 27.
Register B input(s)	On or off	Turn on register B input.
Enable for 'B' Input(s) Register	ENABLE0 ENABLE1 ENABLE2	Specify enable signal for B inputs register.
Enable Output Full Resolution	On or off	Turn on to calculate the resolution of the output based on the input configuration for the block.
Output	[0:72/64]	Manually specify the bitwidths for output.
Register Output	On or off	Specify whether the output is registered and enable signal for the output register.
Enable for Output Register	ENABLE0 ENABLE1 ENABLE2	
Enable Systolic Registers	On or off	Turn on systolic registers. This option is available only for 18x18_sum_of_2 mode and turns the block into 18x18_systolic mode.

Table 15–61 shows the input phase structure parameters.

Table 15–61. Input phase Structure Parameters

Name	Value	Description
Enable Input Scan Chain	On or off	Turns on input scan chain. Available when input registers are enabled. Turn on to pass the first data input of the first multiplier to the first inputs of consequent multipliers.
Use 'scanin' Input	On or off	Instead of data input to the first multiplier, the block uses the `scanin` input, which you can drive from the `scanout` of other DSP blocks.

Table 15–61. Input phase Structure Parameters

Name	Value	Description
Use 'scanout' Output	Startix V Arria V Cyclone V	The scan chain is passed out through a `scanout` port. You can use this data to drive the `scanin` input of other DSP block in chain.
Enable Input Scan Chain Delay Registers	On or off	Turns on scan chain delay registers. You cna add a delay register on a path from the previous multiplier input to the next. Add delay registers on the path from 0->1, 2->3 only in 18x18_systolic mode.
Enable Coefficient ROM	On or off	Turns on coefficient ROM.
Enable Preadder	On or off	Turns on preadder.
Preadder direction	ADD SUB	Specifies the preadder operation, if enabled.
Connect Same Preadder Output to Both Multiplier Inputs.	On or off	Specifies whether both inputs of the multiplier should feed from the same preadder block, allowing it to calculate the square of the sum or subtraction of the A and B inputs.

Table 15–62 shows the output phase structure parameters.

Table 15–62. Output Phase Structure Parameters

Name	Value	Description
Output operation on first multiplier pair	ADD SUB VARIABLE	Specifies whether the results of first (third) multiplier should be added or subtracted to or from the result of the second (fourth) multiplier. Select **VARIABLE** so you can control the operation dynamically through a special control port (addnsub1/3).
Output operation on second multiplier pair	ADD SUB VARIABLE	
Use Chainout Adder Input	On or off	If output chain is available, specifies whether the block uses `chainin` input . Only the `chainout` output of a DSP block can feed the `chainin` input of another DSP block.
Use Chainout Adder Output	On or off	If output chain is available, specifies whether the block uses `chainout` output.
Enable Accumulation	On or off	If accumulation is possible, enabling exposes a control input (`accum`) to control when the block accumulates.
Enable Double Accumulation Register	On or off	Allows you to use double accumulation register. Only for Arria V and Cyclone V device families.
Operation on Multiplier-Adder Output.	ADD SUB VARIABLE	The operation that the block performs on the result of multiplications and additions prior to accumulation. Select **VARIABLE** to control the operation dynamically using the special control input (negate).
Use Rounding Constant	On or off	Any time when accumulation is disabled, you can load a prespecified constant into the accumulator. Otherwise, the block loads 0.
Rounding Constant Value	[0 : 64]	Specifies the only high bit in 64-bit rounding constant. The actual constant values is $2^{constant\_value}$.

Table 15–63 shows the optional data and controls parameters.

Table 15–63. Optional Data and Controls Parameters

Name	Value	Description
C Input	[0:22/27]	The bitwidth for conditional C data input busses. C must share the same clock setting (registering, enable signal) with A and B input busses. For Arria and Cyclone V devices the maximum allowed width for C is 27; for Stratix V devices it is limited to 22.
Coefficient Values	[0:18/27]	The bitwidth for fractional part of the coefficients. The width of the coefficients based on B inputs widths and configuration. It can be either 18 (18-bit mode), 27 (27-bit mode) and 36 (in 36x18 mode). Block enters 36x18 mode, of A input width is less than 18, while B input width is more than 27.
Coefficients for the first multiplier	[c1, c2, ... c8]	
Coefficients for the second multiplier	[c1, c2, ... c8]	Specify up to eight coefficient values for each multiplier if available. You can enter the values as MATLAB vectors.
Coefficients for the third multiplier	[c1, c2, ... c8]	Coefficient selection inputs are data input and should share the same clock setting with A, B inputs.
Coefficients for the fourth multiplier	[c1, c2, ... c8]	
Register Control Input for First Multiplier Pair Output Operation (addnsub1)	On or off	
Register Control Input for First Multiplier Pair Output Operation (addnsub3)	On or off	Each of these parameters specifies whether appropriate control input is registered (if the input is available).
Register Control Input for Accumulation (accum)	On or off	
Register Control Input for Multiplier-Adder Output Operation (negate)	On or off	
Enable for Control Input(s) Register	ENABLE0 ENABLE1 ENABLE2	All control inputs should share the same clock settings. This parameter allows you to specify the enable signal for all registered control inputs.

Table 15–64 shows the inputs and outputs.

Table 15–64. Inputs and Outputs

Port	Direction	Description	Availability
a_real	Input		
a_imag	Input	Real and imaginary components of complex a and b inputs.	Only in 25x18_complex mode.
b_real	Input		
b_imag	Input		
out_real	Output	Real and imaginary components of complex results.	—
out_imag	Output		

Table 15–64. Inputs and Outputs

Port	Direction	Description	Availability
a	Input	Up to 18 bit input to 18x18+36 mode.	Only in 18x18+36 mode.
b	Input	Up to 18 bit input to 18x18+36 mode.	
ab	Input	Up to 36 bit input to 18x18+36 mode.	
a0 (scanin)	Input	First input to first multiplier or preadder.	Available for other than complex and 18x18+36 modes. If you enable **Use scanin input**, this block names this port `scanin` and connects it to scan chain input in generated HDL, otherwise it connects it to data input.
b0	Input	Second input to the first multiplier or preadder.	Available if `CoefRom` not enabled, or if `preadder` is enabled.
c0	Input	If you use preadder, a second input to the first multiplier.	Available if preadder is enabled, but coefficient ROM not.
coef_sel0	Input	Input to select coefficient from the ROM	Available if coefficient ROM is enabled.
a1	Input	First input to second multiplier or preadder.	Available if there are at least two multipliers and input scan chain is disabled.
b1	Input	Second input to the second multiplier or preadder.	Available if there are at least 2 multipliers and coeffficient ROM not enabled, or if preadder is enabled.
c1	Input	If you use preadder, second input to the second multiplier.	Available if there are at least 2 multipliers and if preadder is enabled, but coefficient ROM not.
coef_sel1	Input	Input to select coefficient from the ROM	Available if there are at least 2 multipliers and if coefficient ROM is enabled.
a2	Input	First input to third multiplier or preadder.	Available if there are at least 3 multipliers and input scan chain is disabled.
b2	Input	Second input to the third multiplier or preadder.	Available if there are at least 3 multipliers and coefficient ROM not enabled, or if preadder is enabled.
c2	Input	If you use preadder, a second input to the third multiplier.	Available if there are at least 3 multiplier and if preadder is enabled, but coefficient ROM not.
coef_sel2	Input	Input to select coefficient from the ROM	Available if there are at least 3 multipliers and if coefficient ROM is enabled.
a3	Input	First input to third multiplier or preadder.	Available if there are 4 multipliers and input scan chain is disabled.
b3	Input	Second input to the third multiplier or preadder.	Available if there are 4 multipliers. and coefficient ROM not enabled, or if preadder is enabled.
c3	Input	If you use preadder, a second input to the third multiplier.	Available if there are 4 multiplier and if preadder is enabled, but coefficient ROM not.
coef_sel3	Input	Input to select coefficient from the ROM	Available if there are 4 multipliers and if coefficient ROM is enabled.
chainin	Input	Input to chain adder. Drive only by a `chainout` output of other `Variable Precision DSP` block.	Available only in non complex modes, when `chianout` adder is available and the input is enabled.

Table 15–64. Inputs and Outputs

Port	Direction	Description	Availability
addnsub1	Input	Controls the operation on the first multiplier pair. 1 - ADD, 0 - SUB	Available if dynamic control of multiplier output operation was activated.
addnsub3	Input	Controls the operation on the second multiplier pair. 1 - ADD, 0 - SUB	Available if you select dynamic control of multiplier output operation.
accum	Input	Controls whether the block performs accumulation. 1 - accumulate, 0 - does not accumulate.	Available if you select accumulation.
negate	Input	Controls the operation on Multipliers-Adders output to accumulation. 1 - value is negated, 0 - no negation	Available if you select the **dynamic control of the operation**.
ena0	Input	Enable inputs	Available if you select **enable signals**, and the block uses the concrete signal.
ena1	Input		
ena2	Input		
aclr0	Input	Asynchronous clear outputs: aclr0 for input registers; aclr1 for output registers.	Available if you enable **user asynchrounous clear signals**.
aclr1	Input		
out	Output	The result of the operation.	The result (not complex mode).
scanout	Output	Output of the scan chain.	Only if you enable input scan chain and select **Use scanout Output** .
chainout	Output	Output of the chainout and accumulation adder.	Available only when chaining is available and chainout output was enabled.

The Boards library supports DSP development platforms for the following development boards:

- Arria II GX FPGA Development Kit

- Arria II GX FPGA Development Kit, 6G Edition

- Cyclone IV GX FPGA Development Kit

- Cyclone IV GX Transceiver Starter Kit

- Stratix IV E FPGA Development Kit

- Stratix IV FPGA Development Kit

- Stratix IV GX/GT 40G/100G Interlaken

- Stratix IV GX/GT PCI Express /SFP+

These development boards provide an economical solution for hardware and software verification that enables you to debug and verify both functionality and design timing.

When combined with DSP intellectual property (IP) from Altera or from the Altera Megafunction Partners Program (AMPP^{SM}), you can solve design problems that formerly required custom hardware and software solutions.

Board Configuration

When targeting a development board, your design must contain the corresponding board configuration block at the top hierarchical level. The configuration block properties allow you to specify from a list of available pins to use for the clock and global reset connections. It also displays details of the hardware device on the board.

The other blocks available for each board provide connections to the controls on each board such as LEDs, push buttons, switches, 7-segment displays, connectors, analog-to-digital converters (ADC), and digital-to-analog converters (DAC). By using these blocks, you do not need to make pin assignments to connect the board components.

Adding PLL Output Clocks

You can manually add PLL blocks to your design and configure them to provide the required output clocks with the Quartus Prime Pinout Assignments block to assign pin locations to the PLL outputs.

Like Simulink, DSP Builder supports native complex signal types. Use complex number notation to simplify the design of applications such as FFT, I-Q modulation, and complex filters.

The Complex Type library contains the following blocks:

- Butterfly
- Complex AddSub
- Complex Conjugate
- Complex Constant
- Complex Delay
- Complex Multiplexer
- Complex Product
- Complex to Real-Imag
- Real-Imag to Complex

☞ When connecting DSP Builder blocks to blocks from the Complex Type library (for example, connecting `AltBus` to `Complex AddSub`), you must use `Real-Imag to Complex` or `Complex to Real-Imag` blocks between the blocks. For an example, refer to Figure 17–2 on page 17–5.

Butterfly

The `Butterfly` block performs the following arithmetic operation on complex signed integer numbers:

$A = a + b \times W$
$B = a - b \times W$

where a, b, W, A, and B are complex numbers (type signed integer) such as:

$a = x + jX$
$b = y + jY$
$W = v + jV$
$A = (x + yv) - YV + j(X + Yv + yV)$
$B = (x - yv) + YV + j(X - Yv - yV)$

This function operates with full bit width precision. The full bit width precision of A and B is:

$2 \times [input\ bit\ width] + 2$.

The **Output Bit Width** and **Output Truncated LSB** parameters specify the bit slice for the output ports A and B. For example, if the input bit width is 16, the output bit width is 16, and the output LSB is 4, then the full precision is 34 bits and the output ports $A[15:0]$ and $B[15:0]$ each contain the bit slice 19:4.

Table 17–1 shows the `Butterfly` block inputs and outputs.

Table 17–1. Butterfly Block Inputs and Outputs

Signal	Direction	Description
a	Input	Data input `a`.
b	Input	Data input `b`.
W	Input	Optional input W.
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
A	Output	Data Output A.
B	Output	Data Output B.

Table 17–2 shows the **Butterfly** block parameters.

Table 17–2. Butterfly Block Parameters

Name	Value	Description
Input Bit Width (a, b, W)	>= 1	Specify the bit width of the complex signed integer inputs *a*, *b*, and *W*.
Number of Pipeline Stages	>= 3	Specify the required number of pipeline stages.
Full Resolution for Output Type	On or Off	When this option is on, full output bit width resolution is enabled. When off, you can separately specify the output bit width and LSB of the output.
Output Bit Width (A, B)	>= 1	Specify the bit width of the complex signed integer outputs A and B. This option is available when **Full Resolution for Output Type** is off.
Output Truncated LSB	>= 0	Specify the LSB of the output bus slice of the full resolution computation. This option is available when **Full Resolution for Output Type** is off.
W is constant	On or Off	When this option is on, you can specify the real and imaginary values for W instead of the W port.
W (real)	User defined	Specify the value of the real part of the constant W
W (imaginary)	User defined	Specify the value of the imaginary part of the constant W.
Dedicated Multiplier Circuitry	Auto, Yes, No	For devices that support multipliers, a value of **Auto** specifies that the choice is based on the width of the multiplier.
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (`aclr`).

Table 17–3 shows the `Butterfly` block I/O formats.

Table 17–3. Butterfly Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
I	I1_{Real([Li].[0])Imag([Li].[0])}	I1Real: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0)	
		I1Imag: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0)	Explicit
	I2_{Real([Li].[0])Imag([Li].[0])}	I2Real: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0)	Explicit
		I2Imag: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0)	Explicit
	I3_{Real([Li].[0])Imag([Li].[0])}	I3Real: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0)	Explicit
		I3Imag: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0)	Explicit
	I4_[1]	I4: in STD_LOGIC	Explicit
	I5_[1]	I5: in STD_LOGIC	Explicit
O	O1_{Real([Lo].[0])Imag([Li].[0])}	O1Real: out STD_LOGIC_VECTOR({Lo - 1} DOWNTO 0)	Explicit
		O1Imag: out STD_LOGIC_VECTOR({Lo - 1} DOWNTO 0)	Explicit
	O2_{Real([Lo].[0])Imag([Li].[0])}	O2Real: out STD_LOGIC_VECTOR({Lo - 1} DOWNTO 0)	Explicit
		O2Imag: out STD_LOGIC_VECTOR({Lo - 1} DOWNTO 0)	Explicit

Notes to Table 17–3:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 17–1 shows an example with the `Butterfly` block.

Figure 17–1. Butterfly Block Example
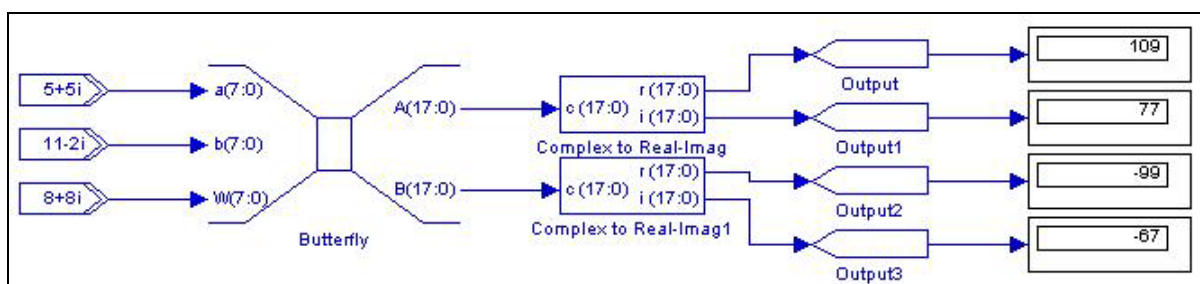
Complex AddSub

The `Complex AddSub` block performs addition or subtraction on a specified number of scalar complex inputs.

Table 17–4 shows the `Complex AddSub` block inputs and outputs.

Table 17–4. Complex AddSub Block Inputs and Outputs

Signal	Direction	Description
+ or -	Input	Complex inputs.
ena	Input	Optional clock enable.

Table 17–4. Complex AddSub Block Inputs and Outputs

Signal	Direction	Description
aclr	Input	Optional asynchronous clear.
R	Output	Result.

Table 17–5 shows the `Complex AddSub` block parameters.

Table 17–5. Complex AddSub Block Parameters

Name	Value	Description
Number of Inputs	>= 2	Specifies the number of input wires to combine.
Add (+) Sub (−)	User defined	Specify addition or subtraction operation for each port with the characters + and −. For example + − + implements +a − b + c for three ports. DSP Builder implements the block as a tree of 2-input adders. Each consecutive pair of inputs are + +, + − or − +. However, none of the input adders can have two consecutive subtractions. Thus, + − − + is valid (as the two input adders are parameterized + − and − +), + − − + + is also valid but + + − − + is not valid. Missing operators are assumed to be +.
Enable Pipeline	On or Off	When this option is on, DSP Builder registers the output from each stage in the adder tree, resulting in a pipeline length that is equal to `ceil(log2(number of inputs))`.
Clock Phase Selection	User Defined	When you enable pipeline, you can specify the phase selection as a binary string, where a 1 indicates the phase in which the block is enabled. For example: `1`—The block is always enabled and captures all data passing through the block (sampled at the rate 1). `10`—The block is enabled every other phase and every other data (sampled at the rate 1) passes through. `0100`—The block is enabled on the second phase of and only the second data of (sampled at the rate 1) passes through. That is, the data on phases 1, 3, and 4 do not pass through the block.
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (`aclr`).

Table 17–6 shows the `Complex AddSub` block I/O formats.

Table 17–6. Complex AddSub Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{Real([L1].[R1])Imag([L1].[R1])}$ ... In$_{Real([Ln].[Rn])Imag([Ln].[Rn])}$ I(n+1)$_{[1]}$ I(n+2)$_{[1]}$	I1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) I1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) ... InReal: in STD_LOGIC_VECTOR({LPn + RPn - 1} DOWNTO 0) InImag: in STD_LOGIC_VECTOR({LPn + RPn - 1} DOWNTO 0) I(n+1): in STD_LOGIC I(n+2): in STD_LOGIC	Implicit Implicit Implicit Implicit
O	O1$_{Real(max(L1,Ln) + 1),(max(RI, Rn) + 1)Imag(max(L1,Ln) + 1),(max(RI,Rn) + 1)}$	O1Real: out STD_LOGIC_VECTOR({max(LI,Ln) + max(RI,Rn)} DOWNTO 0) O1Imag: out STD_LOGIC_VECTOR({max(LI,Ln) + max(RI,Rn)} DOWNTO 0)	Implicit Implicit

Notes to Table 17–6:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 17–2 shows an example with the `Complex AddSub` block.

Figure 17–2. Complex AddSub Block Example
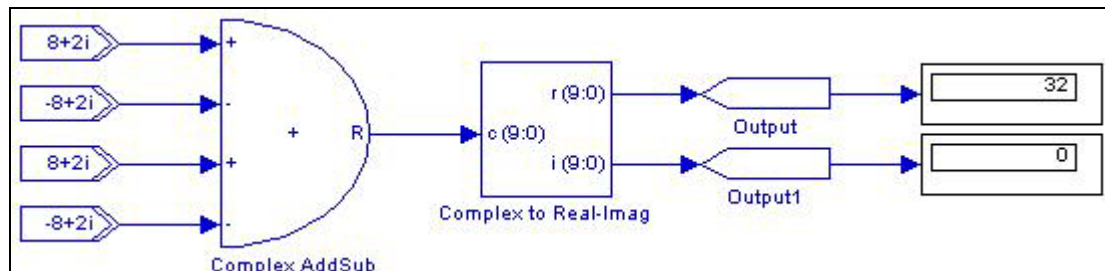
Complex Conjugate

The `Complex Conjugate` block outputs a fixed-point complex conjugate value by performing simple arithmetic operations on the complex inputs. The operation can optionally be conjugate, negative, or negative conjugate. For an input $w = x + iy$, the block returns:

■ Conjugate: $x - iy$

■ Negative: $-x - iy$

■ Negative Conjugate: $-x + iy$

Table 17–7 shows the `Complex Conjugate` block inputs and outputs.

Table 17–7. Complex Conjugate Block Inputs and Outputs

Signal	Direction	Description
w	Input	Complex inputs.
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
c	Output	Fixed point complex conjugate output.

Table 17–8 shows the `Complex Conjugate` block parameters.

Table 17–8. Complex Conjugate Block Parameters

Name	Value	Description
Operation	Conjugate, Negative, Negative Conjugate	Specify the operation to perform.
Register Inputs	On or Off	Turn on to register the inputs and to enable the optional clock enable and asynchronous clear options.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (aclr).

Table 17–9 shows the `Complex Conjugate` block I/O formats.

Table 17–9. Complex Conjugate Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{Real([L1].[R1])Imag([L1].[R1])}$	I1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Implicit
		I1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	
	I2$_{[1]}$	I2: in STD_LOGIC	Implicit
	I3$_{[1]}$	I3: in STD_LOGIC	
O	O1$_{Real([L1] + 1.[R1])Imag([L1] + 1.[R1])}$	O1Real: in STD_LOGIC_VECTOR({LP1 + RP1} DOWNTO 0)	Implicit
		O1Imag: in STD_LOGIC_VECTOR({LP1 + RP1} DOWNTO 0)	Implicit

Notes to Table 17–9:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 17–3 shows an example with `Complex Conjugate` blocks to output conjugate, negative and negative conjugate values.

Figure 17–3. Complex Conjugate Block Example
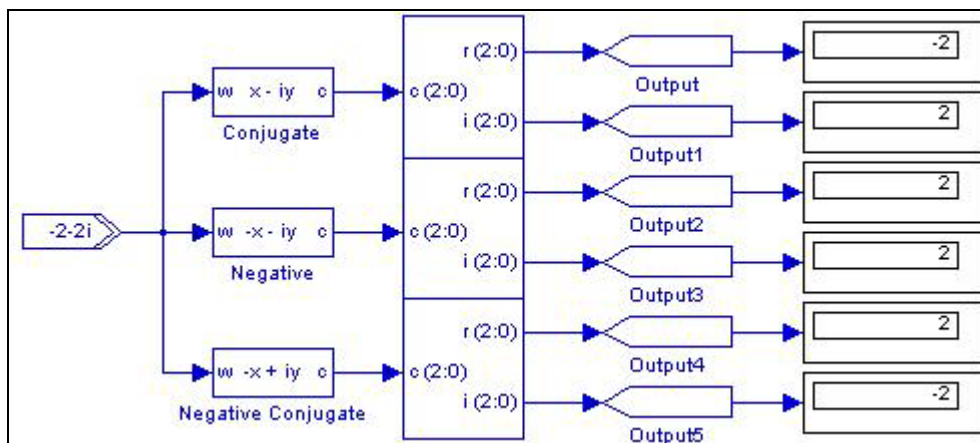
Complex Constant

The `Complex Constant` block outputs a fixed-point complex constant value.

Table 17–10 shows the `Complex Constant` block parameters.

Table 17–10. Complex Constant Block Parameters

Name	Value	Description
Real Part	User Defined	Specify the value of the real part of the constant.
Imaginary Part	User Defined	Specify the value of the imaginary part of the constant.
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Specify the number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined	Specify the clock signal name.

Table 17–11 shows the `Complex Constant` block I/O formats.

Table 17–11. Complex Constant Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
O	O1Real([L1].[R1])Imag([L1].[R1])	O1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) O1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Explicit

Notes to Table 17–11:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 17–4 shows an example with `Complex Constant` blocks as inputs to a `Complex AddSub` block.

Figure 17–4. Complex Constant Block Example
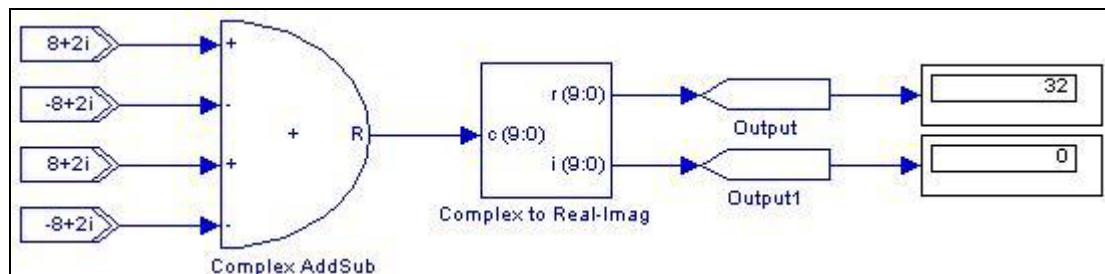
Complex Delay

The `Complex Delay` block delays the incoming data by an amount specified by the **Number of Pipeline Stages** parameter. The input must be a complex number.

Table 17–12 shows the `Complex Delay` block inputs and outputs.

Table 17–12. Complex Delay Block Inputs and Outputs

Signal	Direction	Description
d	Input	Input data.
ena	Input	Optional clock enable.
sclr	Input	Optional synchronous clear.
q	Output	Delayed output data.

Table 17–13 shows the Complex Delay block parameters.

Table 17–13. Complex Delay Block Parameters

Name	Value	Description
Number of Pipeline Stages	>= 1	Specify the delay length of the block.
Clock Phase Selection	User Defined	When you enable pipeline, you can indicate the phase selection with a binary string, where a 1 indicates the phase in which the block is enabled. For example: 1—The block is always enabled and captures all data passing through the block (sampled at the rate 1). 10—The block is enabled every other phase and every other data (sampled at the rate 1) passes through. 0100—The block is enabled on the second phase of and only the second data of (sampled at the rate 1) passes through. That is, the data on phases 1, 3, and 4 do not pass through the block.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear input (sclr).

Table 17–14 shows the Complex Delay block I/O formats.

Table 17–14. Complex Delay Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1Real([L1].[R1])Imag([L1].[R1]) I2[1] I3[1]	I1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) I1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC	Implicit Implicit
O	O1Real([L1].[R1])Imag([L1].[R1])	O1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) O1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Implicit

Notes to Table 17–14:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 17–5 shows an example with the Complex Delay block.

Figure 17–5. Complex Delay Block Example
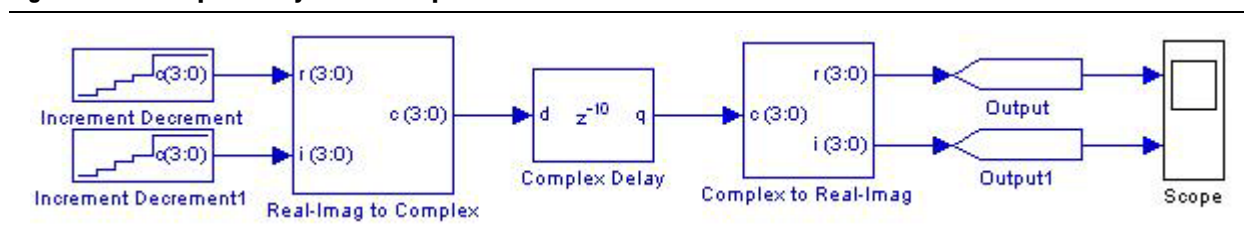
Complex Multiplexer

The Complex Multiplexer block multiplexes *N* complex inputs to one complex output. The select port sel is a non-complex scalar.

Table 17–15 shows the Complex Multiplexer block inputs and outputs.

Table 17–15. Complex Multiplexer Block Inputs and Outputs

Signal	Direction	Description
sel	Input	Non-complex select line.
0 to N–1	Input	Complex inputs.
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
unnamed	Output	Result.

Table 17–16 shows the Complex Multiplexer block parameters.

Table 17–16. Complex Multiplexer Block Parameters

Name	Value	Description
Number of Input Data Lines	>= 2	Number of complex input data lines.
Number of Pipeline Stages	>= 0	Specify the delay length of the block.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (aclr).
One Hot Select Bus	On or Off	Turn on to use one-hot selection for the select signal instead of full binary.

Table 17–17 shows the `Complex Multiplexer` block I/O formats.

Table 17–17. Complex Multiplexer Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1Real([L1].[R1])Imag([L1].[R1]) I2Real([L2].[R2])Imag([L2].[R2]) I3[1] I4[1] I5[1]	I1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) I1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) I2Real: in STD_LOGIC_VECTOR({LP2 + RP2 - 1} DOWNTO 0) I2Imag: in STD_LOGIC_VECTOR({LP2 + RP2 - 1} DOWNTO 0) I3: in STD_LOGIC I4: in STD_LOGIC I5: in STD_LOGIC	Implicit Implicit
O	O1Real(max(L1,L2)),(max(RI,R2)) Imag(max(L1,L2)),(max(RI,R2))	O1Real: in STD_LOGIC_VECTOR({max(LI,L2) + max(RI,R2) - 1} DOWNTO 0) O1Imag: in STD_LOGIC_VECTOR({max(LI,L2) + max(RI,R2) - 1} DOWNTO 0)	Implicit

Notes to Table 17–17:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 17–6 shows an example with the `Complex Multiplexer` block.

Figure 17–6. Complex Multiplexer Block Example
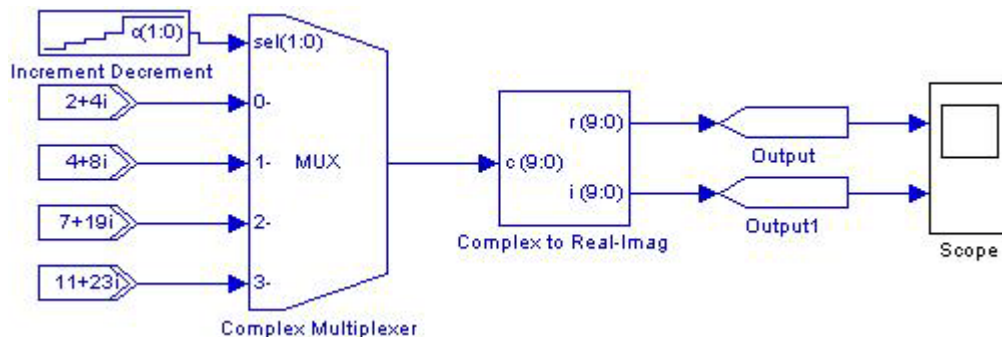
Complex Product

The `Complex Product` block performs output multiplication of two scalar complex inputs. Operand *a* is multiplied by operand *b* and the result output on *r* as the following equation shows:

$$r = a \times b$$

Table 17–18 shows the `Complex Product` block inputs and outputs.

Table 17–18. Complex Product Block Inputs and Outputs

Signal	Direction	Description
a	Input	Complex operand a.
b	Input	Complex operand b.
ena	Input	Optional clock enable.
aclr	Input	Optional asynchronous clear.
r	Output	Result.

Table 17–19 shows the `Complex Product` block parameters.

Table 17–19. Complex Product Block Parameters

Name	Value	Description
Bus Type	Inferred, Signed Integer, Signed Fractional, Unsigned Integer	Specify the bus number format that you want to use. Inferred means that the format is automatically set by the format of the connected signal.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This option applies only to signed fractional formats.
Pipeline Register	No Register, Inputs Only, Multiplier Only, Adder Only, Inputs and Multiplier, Inputs and Adder, Multiplier and Adder, Inputs Multiplier and Adder	Specify the elements that you want pipelined. The clock enable and asynchronous clear ports are available only if the block is registered.
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (`aclr`).
Use Dedicated Circuitry	On or Off	If you target devices that support DSP blocks, turn on to implement the functionality in DSP blocks instead of logic elements.

Table 17–20 shows the Complex Product block I/O formats.

Table 17–20. Complex Product Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1_{Real([L1].[R1])Imag([L1].[R1])} I2_{Real([L2].[R2])Imag([L2].[R2])} I3_[1] I4_[1]	I1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) I1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) I2Real: in STD_LOGIC_VECTOR({LP2 + RP2 - 1} DOWNTO 0) I2Imag: in STD_LOGIC_VECTOR({LP2 + RP2 - 1} DOWNTO 0) I3: in STD_LOGIC I4: in STD_LOGIC	Implicit
O	O1_{Real(2 x max(LI,L2)),(2 x max(RI,R2))} Imag(2 x max(LI,L2)),(2 x max(RI,R2))	O1Real: in STD_LOGIC_VECTOR({(2 x max(LI,L2)) + (2 x max(RI,R2)) -1} DOWNTO 0) O1Imag: in STD_LOGIC_VECTOR({(2 x max(LI,L2)) + (2 x max(RI,R2)) -1} DOWNTO 0)	Implicit

Notes to Table 17–20:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.
(3) I1_{[L].[R]} is an input port. O1_{[L].[R]} is an output port.
(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 17–7 shows an example with the Complex Product block.

Figure 17–7. Complex Product Block Example
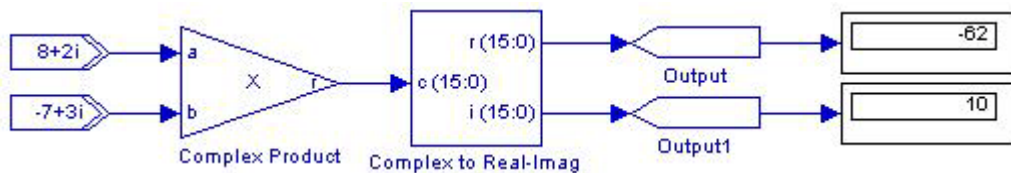
Complex to Real-Imag

The Complex to Real-Imag block constructs a fixed-point real and fixed-point imaginary output from a complex input.

Table 17–21 shows the Complex to Real-Imag block inputs and outputs.

Table 17–21. Complex to Real-Imag Block Inputs and Outputs

Signal	Direction	Description
c	Input	Complex input.
r	Output	Real part output.
i	Output	Imaginary part output.

Table 17–22 shows the `Complex to Real-Imag` block parameters.

Table 17–22. Complex to Real-Imag Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Specify the number format you want to use for the bus.
[number of bits].[]	>= 0 (Parameterizable)	Select the number of data input bits to the left of the binary point, including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Select the number of data input bits to the right of the binary point. This option applies only to signed fractional formats.

Table 17–23 shows the `Complex to Real-Imag` block I/O formats.

Table 17–23. Complex to Real-Imag Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{Real([L1].[R1])}$Imag($_{[L1].[R1]}$)	I1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) I1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Implicit
O	O1$_{Real([L1].[R1])}$ O2$_{Imag([L1].[R1])}$	O1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) O2Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Explicit

Notes to Table 17–23:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 17–8 shows an example with the `Complex to Real-Imag` block.

Figure 17–8. Complex to Real-Imag Block Example
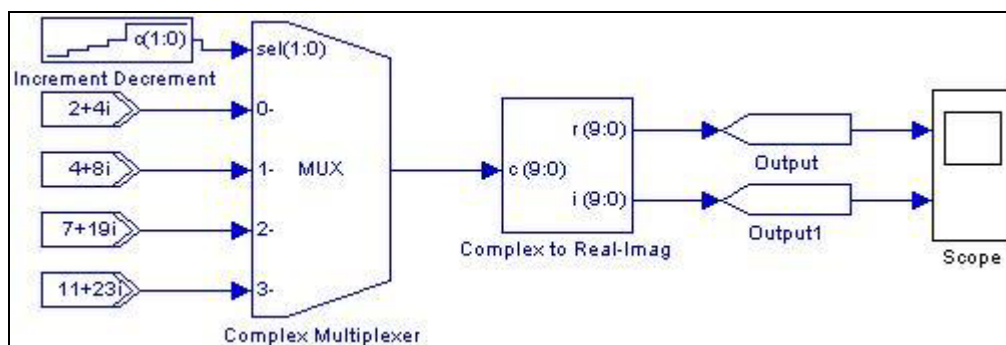
Real-Imag to Complex

The `Real-Imag to Complex` block constructs a fixed-point complex output from real and imaginary inputs.

Table 17–24 shows the `Real-Imag to Complex` block has the inputs and outputs.

Table 17–24. Real-Imag to Complex Block Inputs and Outputs

Signal	Direction	Description
r	Input	Real part input.
i	Input	Imaginary part input.
c	Output	Complex output.

Table 17–25 shows the `Real-Imag to Complex` block parameters.

Table 17–25. Real-Imag to Complex Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Specify the number format you want to use for the bus.
[number of bits].[]	>= 0 (Parameterizable)	Select the number of data input bits to the left of the binary point, including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Select the number of data input bits to the right of the binary point. This option applies only to signed fractional formats.

Table 17–26 shows the `Real-Imag to Complex` block I/O formats.

Table 17–26. Real-Imag to Complex Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1Real([L1].[R1]) I2Imag([L1].[R1])	I1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) I1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Implicit
O	O1Real([L1].[R1])Imag([L1].[R1])	O1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) O1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Explicit

Notes to Table 17–26:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.
(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.
(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 17–9 shows an example with the `Real-Imag to Complex` block.

Figure 17–9. Real-Imag to Complex Block Example
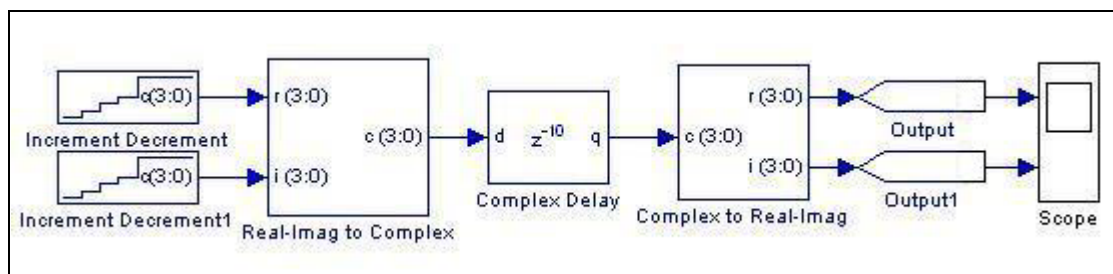
The blocks in the Gate & Control library support gate and other related control functions.

The Gate & Control library contains the following blocks:

■ Binary to Seven Segments

■ Bitwise Logical Bus Operator

■ Case Statement

■ Decoder

■ Demultiplexer

■ Flipflop

■ If Statement

■ LFSR Sequence

■ Logical Bit Operator

■ Logical Bus Operator

■ Logical Reduce Operator

■ Multiplexer

■ Pattern

■ Single Pulse

Binary to Seven Segments

The `Binary to Seven Segments` block converts a 4-bit unsigned input bus to a 7-bit output for connection to a seven-segment displays.

The seven-segment display is set to display the hexadecimal representation of the input number.

Table 18–1 shows the `Binary to Seven Segments` block inputs and outputs.

Table 18–1. Binary to Seven Segments Block Inputs and Outputs

Signal	Direction	Description
(3:0)	Input	4-bit data input.
(6:0)	Output	7-bit data output.

Table 18–2 shows the 4-bit to 7-bit conversion performed by the `Binary to Seven Segments` block.

Table 18–2. Binary to Seven Segments

Input			Output	
Binary	**Decimal**	**Hex**	**Binary**	**Decimal**
0000	0	0	1000000	64
0001	1	1	1111001	121
0010	2	2	0100100	36
0011	3	3	0110000	48
0100	4	4	0011001	25
0101	5	5	0010010	18
0110	6	6	0000010	2
0111	7	7	1111000	120
1000	8	8	0000000	0
1001	9	9	0010000	16
1010	10	A	0001000	8
1011	11	b	0000011	3
1100	12	C	1000110	70
1101	13	d	1000001	33
1110	14	E	0000110	6
1111	15	F	0001110	14

Table 18–3 shows the `Binary to Seven Segments` block I/O formats.

Table 18–3. Binary to Seven Segments Display Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[4].[0]}$	I1: in STD_LOGIC_VECTOR(3 DOWNTO 0)	Explicit
O	$O1_{[7].[0]}$	O1: in STD_LOGIC_VECTOR(6 DOWNTO 0)	Explicit

Notes to Table 18–3:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 18–1 shows an example with the `Binary to Seven Segments` block.

Figure 18–1. Binary to Seven Segments Block Example
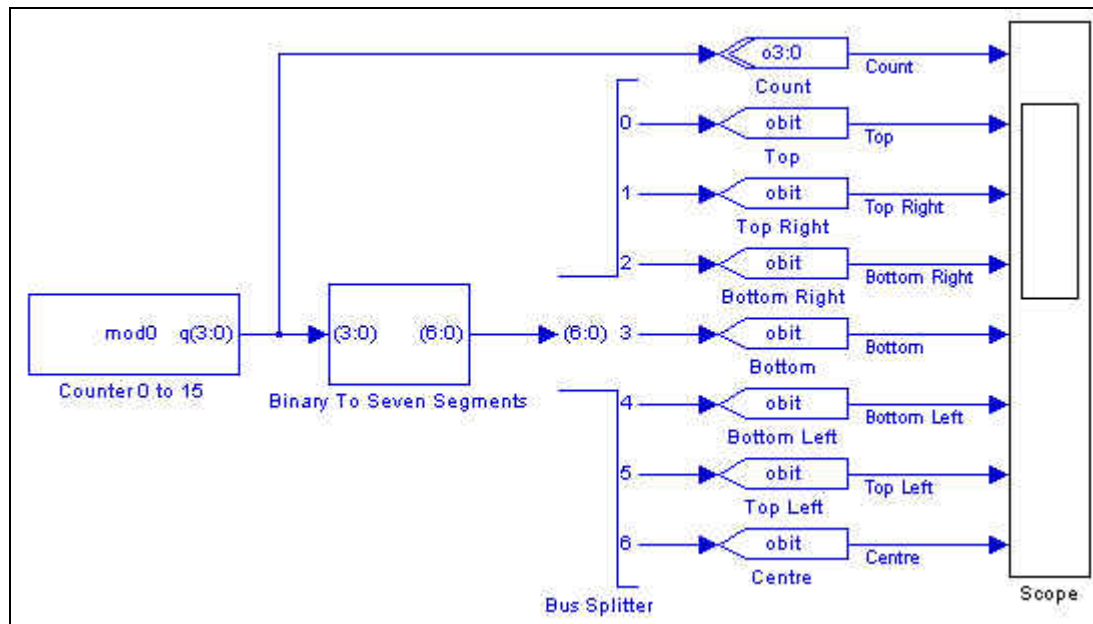
Bitwise Logical Bus Operator

The `Bitwise Logical Bus Operator` block performs bitwise AND, OR, or XOR logical operations on two input buses.

Table 18–4 shows the `Bitwise Logical Bus Operator` block inputs and outputs.

Table 18–4. Bitwise Logical Bus Operator Block Inputs and Outputs

Signal	Direction	Description
a	Input	Data input a.
b	Input	Data input b.
q	Output	Data output.

Table 18–5 shows the `Bitwise Logical Bus Operator` block parameters.

Table 18–5. Bitwise Logical Bus Operator Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Specify the bus number format that you want to use.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point, including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point.
Logic Operation	AND, OR, XOR	Specify the logical operation to perform.

Table 18–6 shows the `Bitwise Logical Bus Operator` block I/O formats.

**Table 18–6. Bitwise Logical Bus Operator Block I/O Formats** [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[L1].[R1]	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
	I2[L1].[R1]	I2: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
O	O1[L1].[R1]	O1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit

Notes to Table 18–6:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 18–2 shows an example with the `Bitwise Logical Bus Operator` block.

Figure 18–2. Bitwise Logical Bus Operator Block Example
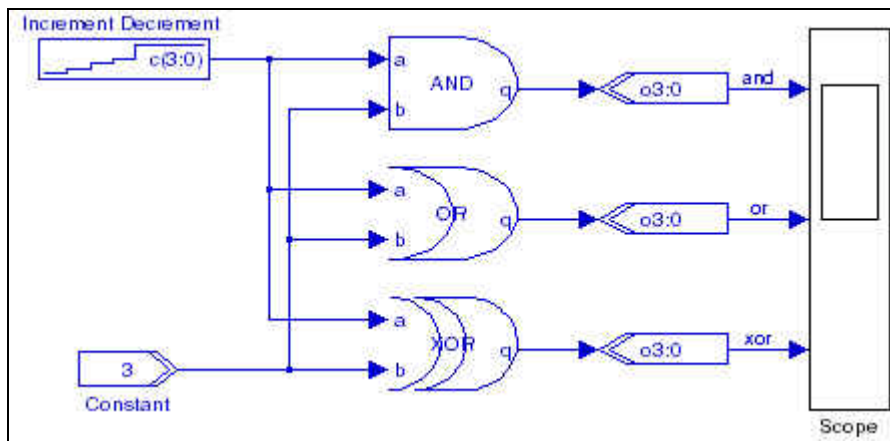
Case Statement

This `Case Statement` block contains boolean operators, which you can use for combinational functions.

The `Case Statement` block compares the input signal (which must be a signed or unsigned integer) with a set of values (or cases). A single-bit output generates for each case. You can implement multiple cases with a comma (,) to separate each case. A comma at the end of the case values is ignored.

You can have multiple conditions for each case with a pipe (|) to separate the conditions. For example, for four cases if the first has two conditions, enter `1|2,3,4,5` in the **Case Values** box.

Table 18–7 shows the `Case Statement` block inputs and outputs.

Table 18–7. Case Statement Block Inputs and Outputs

Signal	Direction	Description
unnamed	Input	Data input.
0 to n	Output	A separate output is provided for each case.

Table 18–8 shows the `Case Statement` block parameters.

Table 18–8. Case Statement Block Parameters

Name	Value	Description
Case Statement	User defined (Parameterizable)	Specify the values with which you want to compare the input. Use a comma between each case and separate conditions by a pipe (|). For example: 1|2|3,4,5|-1,7
Data Bus Type	Signed Integer, Unsigned Integer	Specify the bus number format that you want to use.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point, including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point.
Enable Pipeline	On or Off	Turn on if you want pipeline the output result.
Provide Default Case	On or Off	Turn on if you want the `others` output signal to go high when all the other outputs are false.

Table 18–9 shows the `Case Statement` block I/O formats.

**Table 18–9. Case Statement Block I/O Formats** [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Explicit
O	$O1_{[1]}$ … $Oi_{[1]}$ …. $On_{[1]}$	O1: out STD_LOGIC … Oi: out STD_LOGIC …. On: out STD_LOGIC	Explicit

Notes to Table 18–9:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 18–3 shows an example model with the Case Statement block.

Figure 18–3. Case Statement Block Example
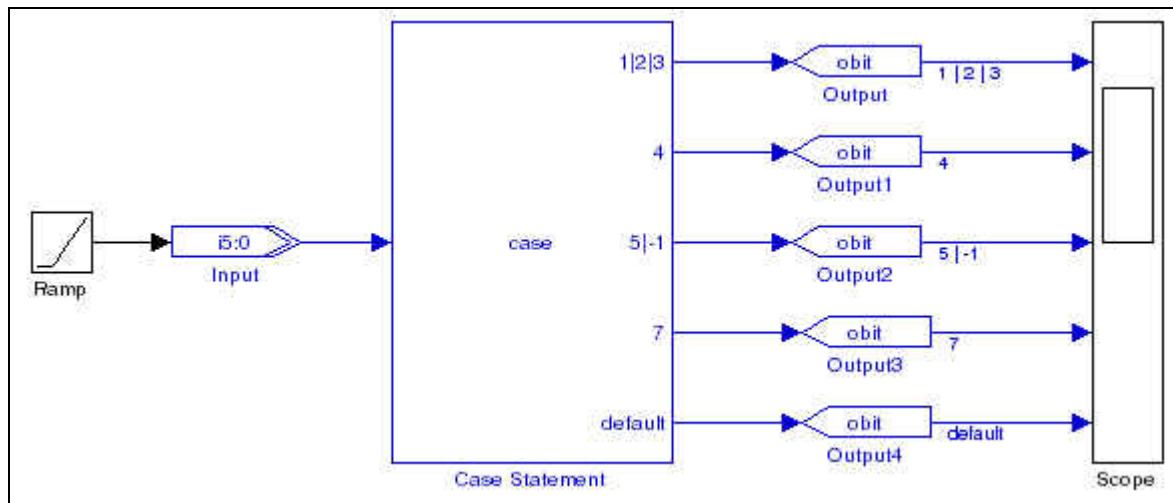
The following VHDL code generates from the model in Figure 18–3:

```
caseproc:process( input )
   begin
      case input is
         when "00000001" | "00000010" | "00000011" =>
               r0 <= '1';
               r1 <= '0';
               r2 <= '0';
               r3 <= '0';
               r4 <= '0';
         when "00000100" =>
               r0 <= '0';
               r1 <= '1';
               r2 <= '0';
               r3 <= '0';
               r4 <= '0';
         when "00000100" | "00000110" =>
               r0 <= '0';
               r1 <= '0';
               r2 <= '1';
               r3 <= '0';
               r4 <= '0';
         when "00000111" =>
               r0 <= '0';
               r1 <= '0';
               r2 <= '0';
               r3 <= '1';
               r4 <= '0';
         when others =>
               r0 <= '0';
               r1 <= '0';
               r2 <= '0';
               r3 <= '0';
               r4 <= '1';
      end case;
end process;
```

☞ The `Case Statement` block output ports in the VHDL are named r<number> where <number> is auto-generated.

Decoder

The `Decoder` block is a bus decoder that compares the input value against the specified decoded value. If the values match, the block outputs a 1, if they do not match it outputs a 0.

If the specified value is not representable in the data type of the input bus, it is truncated to the data type of the input bus. For example: 5 (binary 101) as a 2 bit unsigned integer results in 1 (binary 01).

Table 18–10 shows the `Decoder` block inputs and outputs.

Table 18–10. Decoder Block Inputs and Outputs

Signal	Direction	Description
in	Input	Data input.
match	Output	Data output (1 = match, 0 = mismatch).

Table 18–11 shows the `Decoder` block parameters.

Table 18–11. Decoder Block Parameters

Name	Value	Description
Input Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Specify the bus number format that you want to use.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point for the gain. This option is zero (0) unless Signed Fractional is selected.
Register Output	On or Off	Turn this option on if you want to register the output result.
Decoded Value	User defined (Parameterizable)	Specify the decoded value for matching.

Table 18–12 shows the `Decoder` block I/O formats.

Table 18–12. Decoder Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
O	O1$_{[1].[0]}$	O1: in STD_LOGIC	Explicit

Notes to Table 18–12:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.
(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.
(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 18–4 shows an example with the `Decoder` block.

Figure 18–4. Decoder Block Example
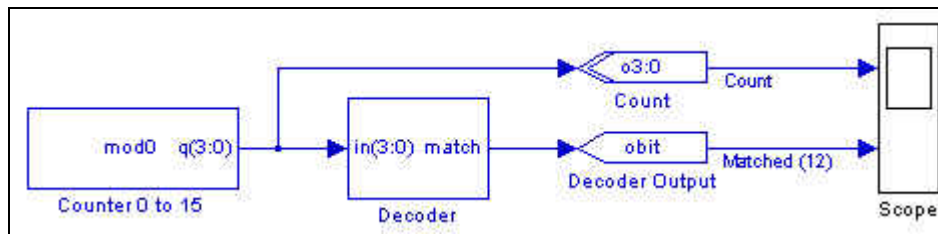
Demultiplexer

The `Demultiplexer` block is a 1-to-*n* demultiplexer that uses full encoded binary values. The value of the input *d* is output to the selected output. All other outputs remain constant.

The `sel` input is an unsigned integer bus.

Table 18–13 shows the `Demultiplexer` block inputs and outputs.

Table 18–13. Demultiplexer Block Inputs and Outputs

Signal	Direction	Description
d	Input	Data input port.
sel	Input	Select control port.
ena	Input	Optional clock enable port.
sclr	Input	Optional synchronous clear port.
0–(n-1)	Output	Output ports.

Table 18–14 describes the parameters for the `Demultiplexer` block.

Table 18–14. Demultiplexer Block Parameters

Name	Value	Description
Number of Output Data Lines	An integer greater than 1 (Parameterizable)	Specify how many outputs you want the demultiplexer to have.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear input (sclr).

Table 18–15 shows the `Demultiplexer` block I/O formats.

**Table 18–15. Demultiplexer Block I/O Formats** [2]

I/O	Simulink [3], [4]	VHDL	Type [5]
I	I1[L].[R]	I1: in STD_LOGIC_VECTOR({L + R1 - 1} DOWNTO 0)	
	I2[L].[R]	I2: in STD_LOGIC_VECTOR({L - 1} DOWNTO 0)	Implicit
	I3[1]	I3: in STD_LOGIC	Implicit
	I4[1]	I4: in STD_LOGIC	
O	O1[L].[R]	O1: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Implicit
	...	...	
	On[L].[R] [1]	On: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Implicit

Notes to Table 18–15:

(1) Where I is the number of outputs to the demultiplexer.

(2) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(3) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(4) I1[L].[R] is an input port. O1[L].[R] is an output port.

(5) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 18–5 shows an example with the `Demultiplexer` block.

Figure 18–5. Demultiplexer Block Example
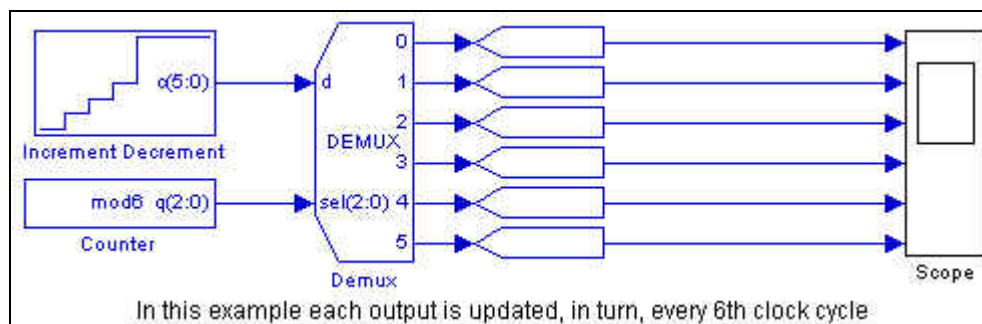
Flipflop

Set the `Flipflop` block as a D-type flipflop with enable (DFFE) or T-type flipflop with enable (TFFE).

If the number of bits is set to more than 1, the block behaves as single-bit flipflops for each bit. For example, for a TFFE flipflop with an n-bit signal, the signal is processed with n 1-bit TFFE flipflops.

Table 18–16 shows the `Flipflop` block inputs and outputs.

Table 18–16. Flipflop Block Inputs and Outputs

Signal	Direction	Description
input	Input	Data or togggle port.
ena	Input	Enable port.

Table 18–16. Flipflop Block Inputs and Outputs

Signal	Direction	Description
aprn	Input	Asynchronous reset port.
aclrn	Input	Asynchronous clear port.
Q	Output	Output port.

DFFE mode:

```
if (0 == aclrn)   Q = 0;
else if (0 == aprn)   Q = 1;
else if (1 == ena)    Q = D
```

TFFE mode:

```
if (0 == aclrn)   Q = 0;
else if (0 == aprn)   Q = 1;
else if (1 == ena) and (1 == T)   Q = toggle
```

☞ DSP Builder does not support (aclrn == 0) and (aprn == 0).

The aclrn port is an active-low asynchronous clear port. When active this sets the output and internal state to 0 for the remainder/duration of the clock cycle.

The aprn port is an active-low asynchronous preset port. When active this sets the output and internal state to 1 for the remainder/duration of the clock cycle.

Table 18–17 shows the Flipflop block parameters.

Table 18–17. Flipflop Block Parameters

Name	Value	Description
Mode	DFFE or TFFE	Specify the type of flipflop to implement.
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, Single Bit	Specify the bus number format that you want to use.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point for the gain. This option is zero (0) unless you select **Signed Fractional**.

Table 18–18 shows the Flipflop block I/O formats.

Table 18–18. Flipflop Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[0]}$	I1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0)	Explicit
	$I2_{[1].[0]}$	I2: in STD_LOGIC	
	$I3_{[1].[0]}$	I3: in STD_LOGIC	
	$I4_{[1].[0]}$	I4: in STD_LOGIC	

Table 18–18. Flipflop Block I/O Formats (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
O	O1[L1].[0]	O1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0)	Explicit

Notes to Table 18–18:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 18–6 shows an example with the Flipflop block.

Figure 18–6. Flipflop Block Example
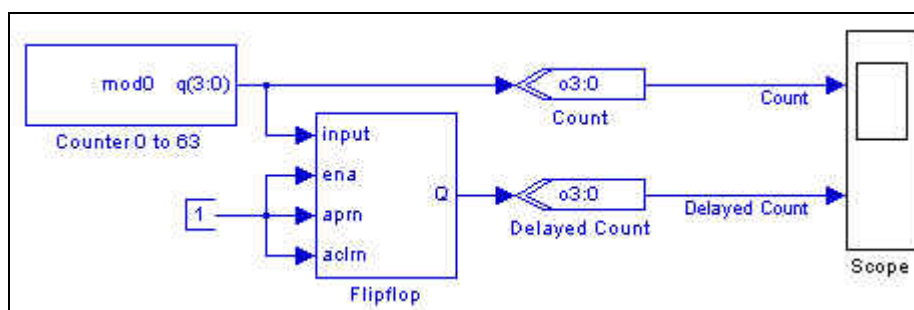
If Statement

The If Statement block outputs a 0 or 1 result based on the IF condition expression.

Table 18–19 shows the If Statement block inputs and outputs.

Table 18–19. If Statement Block Inputs and Outputs

Signal	Direction	Description
a–j	Input	Input ports.
n	Input	Optional ELSE IF input port.
true	Output	Output port (high when true).
false	Output	Optional ELSE output port (high when false).

You can build an IF condition expression with the signal values 0 or 1 and any of the permitted operators given in Table 18–20.

Table 18–20. Supported If Statement Block Operators

Operator	Operation
&	AND
|	OR
$	XOR
=	Equal To
~	Not Equal To

Table 18–20. Supported If Statement Block Operators

Operator	Operation
>	Greater Than
<	Less Than
()	Parentheses

When writing expressions in an `If Statement` block, ensure that the operators are always operating on the same types. That is, bus signals compare with and operate with bus signals; and booleans (the 'true' or 'false' result of such operations) only compare with and operate with booleans. In other words, the types must be the same on either side of an operator.

Treat an `If statement` expression, 0 and 1, as signals rather than as booleans, otherwise you receive an error at HDL generation of the following form:

```
Can't determine definition of operator "<mixed operator>" -- found 0
possible definitions
```

If you receive this error, carefully check the expressions specified in the `If Statement` blocks.

The following examples of bad syntax give errors:

■ `(a>b)&c`, where `a`,`b` and `c` are all input values to the If Statement.

Here `(a>b)` returns a boolean ('true' or 'false') and is `AND`ed with signal `c`. This operation is ill defined and results in the following error:

```
Can't determine definition of operator ""&"" -- found 0 possible
definitions
```

■ `((a>b)~0)`

Again `(a>b)` returns a boolean ('true' or 'false'). `0` is treated as a signal not a boolean, so the hardware generation fails with an error:

```
Can't determine definition of operator ""/="" -- found 0 possible
definitions"
```

where `/=` is the hardware translation of the 'not equal to' operator. Here the `~0` incorrectly means 'not false', and is unnecessary. The correct syntax for this expression is just `(a>b)`.

Table 18–21 shows the `If Statement` block parameters.

Table 18–21. If Statement Block Parameters

Name	Value	Description
Number of Inputs	2–10	Specify the number of inputs to the `If Statement`.
IF Expression	User Defined	Specify the if condition with any of the following operators: &, |, $, =, ~, >, <, or (), the variables a, b, c, d, e, f, g, h, i, or j, and the single digit numerals 0, 1.
Data Bus Type	Signed Integer, Signed Fractional, Unsigned Integer Single Bit, Inferred	Specify the bus number format that you want to use. The selected type must be capable of expressing 0 and 1 exactly.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point.

Table 18–21. If Statement Block Parameters

Name	Value	Description
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point for the gain. This option is zero (0) unless Signed Fractional is selected.
Use ELSE Output Port	On or Off	This option turns on the `false` output, which implements an `ELSE` condition and goes high if the condition evaluated by the `If Statement` block is false.
Use ELSE IF Input Port	On or Off	This option turns on the `else` input, which implements an `ELSE IF` input, when you want to cascade multiple `IF Statement` blocks together or as an enable for the block.

Table 18–22 shows the `If Statement` block I/O formats.

Table 18–22. If Statement Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L1].[R1]}$ …. Ii$_{[LI].[RI]}$ … In$_{[LN].[RN]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) … Ii: in STD_LOGIC_VECTOR({LI + RI - 1} DOWNTO 0) …. In: in STD_LOGIC_VECTOR({LN + RN - 1} DOWNTO 0)	Implicit
O	O1$_{[1]}$ O2$_{[1]}$	O1: out STD_LOGIC O2: out STD_LOGIC	Explicit

Notes to Table 18–22:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 18–7 shows an example of the `If Statement` block, which implements the conditional statement:

```
Quantizer:
if (Input<-4)  Output = -100
else if ((Input>=-4) & (Input<10)) Output = 0
else Output = 100
```

Figure 18–7. If Statement Block Example
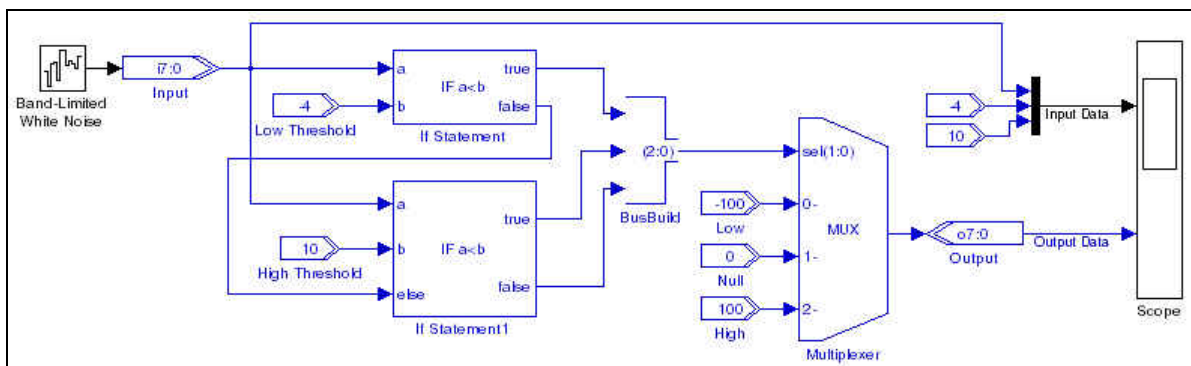
LFSR Sequence

The LFSR Sequence block implements a linear feedback shift register that shifts one bit across L registers. The register output bits shift from LSB to most significant bit (MSB) with the output sout connected to the MSB of the shift register. The register output bits can optionally be XORed or XNORed together.

For example, when choosing an LFSR sequence of length eight, the default polynomial is $x8 + x4 + x3 + x2 + 1$ with the circuitry that Figure 18–8 shows.

Figure 18–8. Default LFSR Sequence Block with Length 8 Circuitry
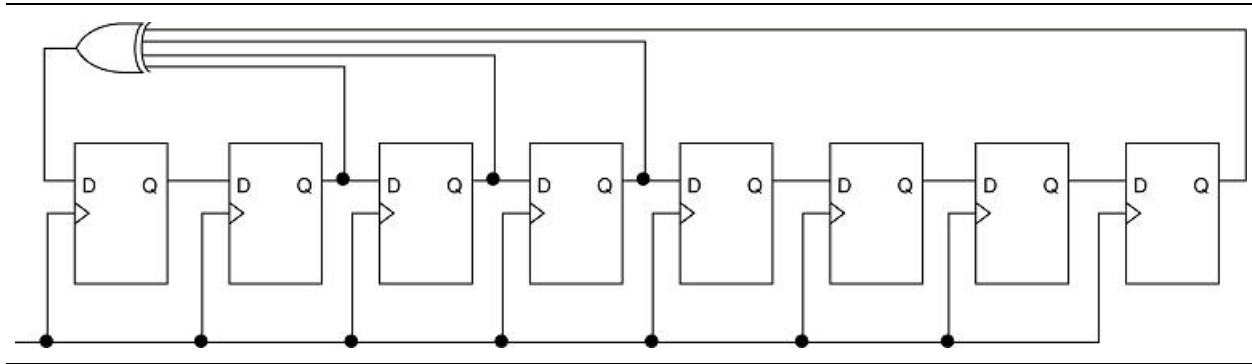
In this default structure:

■ The polynomial is a primitive or maximal-length polynomial

■ All registers are initialized to one

■ The feedback gate type is XOR

■ The feedback structure is an external n-input gate or many to one

You can modify the implemented LFSR sequence by changing the parameter values.

For example, after changing the feedback structure to an internal two-inputs gate, DSP Builder implements the circuitry (Figure 18–9).

Figure 18–9. Internal 2-Input Gate Circuitry
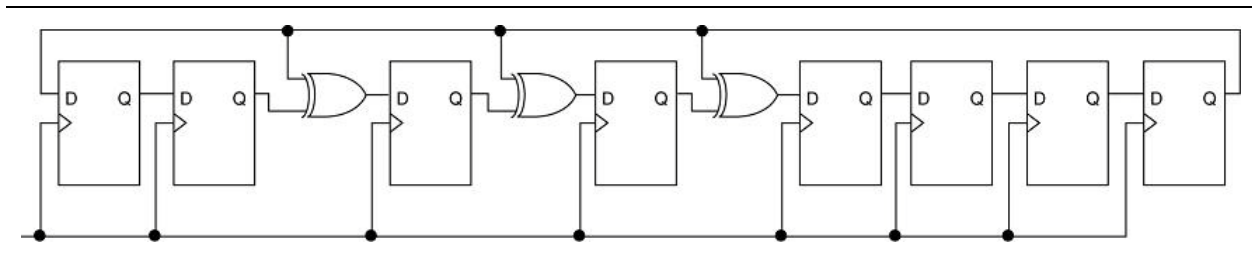
This circuitry changes the sequence from:

```
1 1 1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0 1
```

to:

```
1 1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0 0
```

Table 18–23 shows the `LFSR Sequence` block inputs and outputs.

Table 18–23. LFSR Sequence Block Inputs and Outputs

Signal	Direction	Description
ena	Input	Optional clock enable port.
rst	Input	Optional reset port.
sout	Output	Serial output port for MSB of the LFSR.
pout	Output	Optional parallel output port for LFSR unsigned value.

Table 18–24 shows the `LFSR Sequence` block parameters.

Table 18–24. LFSR Sequence Block Parameters

Name	Value	Description
LFSR Length	User Defined (Parameterizable)	Specify the LFSR length as an integer.
Feedback Structure	External n-inputs gate, Internal two-inputs gate	Specify whether you want an external n-inputs gate (many-to-one) or internal two-inputs gate (one-to-many) structure.
Feedback Gate Type	XOR or XNOR	Specify the type of feedback gate to implement.
Initial Register Value (Hex)	Any Hexadecimal Number (Parameterizable)	Specify the initial values in the register. If this value is larger than is represented in the shift register (set by LFSR Length) the unrepresentable bits are truncated.
Primitive Polynomial Tap Sequence	User-Defined Array of Polynomial Coefficients (Parameterizable)	Specify where the taps occur in the shift register, 1 denotes the LSB and the LFSR length denotes the MSB. There must be a minimum of 2 taps. The numbers should be enclosed in square brackets. For example, [0 3 10].
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined (Parameterizable)	Specify the name of the clock signal.
Use Parallel Output	On or Off	Turn on to use the parallel output (pout).
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear input (sclr).

Table 18–25 shows the `LFSR Sequence` block I/O formats.

Table 18–25. LFSR Sequence Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type
I	I1$_{[1].[0]}$	I1: in STD_LOGIC	—
	I2$_{[1].[0]}$	I2: in STD_LOGIC	—
O	O1$_{[1].[0]}$	O1: out STD_LOGIC	—
	O2$_{[L].[0]}$	O2: out STD_LOGIC_VECTOR(L-1 DOWNTO 0)	—

Notes to Table 18–25:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

Figure 18–10 shows an example with the LFSR Sequence block.

Figure 18–10. LFSR Sequence Block Example
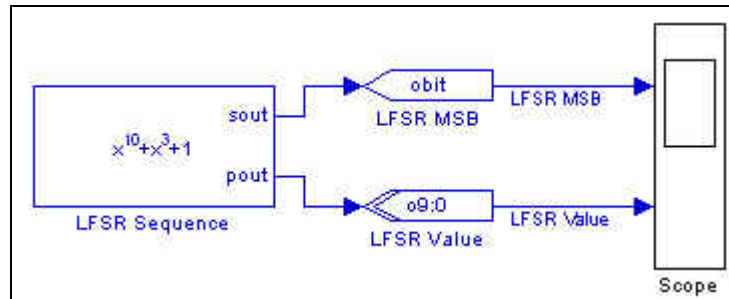
Logical Bit Operator

The Logical Bit Operator block performs logical operations on single-bit inputs. You can specify a variable number of inputs. If the integer is positive, it is interpreted as a boolean 1, otherwise it is interpreted as 0. The number of inputs is variable.

Table 18–26 shows the Logical Bit Operator block parameters.

Table 18–26. Logical Bit Operator Block Parameters

Name	Value	Description
Logical Operator	AND, OR, XOR, NAND, NOR, NOT	Specify the operator you want to use.
Number of Inputs	1–16 (Parameterizable)	Specify the number of inputs. This parameter defaults to 1 if the NOT logical operator is selected.

Table 18–27 shows the Logical Bit Operator block I/O formats.

Table 18–27. Logical Bit Operator Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[1]}$ … $Ii_{[1]}$ …. $In_{[1]}$	I1: in STD_LOGIC … Ii: in STD_LOGIC …. In: in STD_LOGIC	Explicit
O	$O1_{[1]}$	O1: out STD_LOGIC	Explicit

Notes to Table 18–27:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 18–11 shows an example with the `Logical Bit Operator` block.

Figure 18–11. Logical Bit Operator Block Example
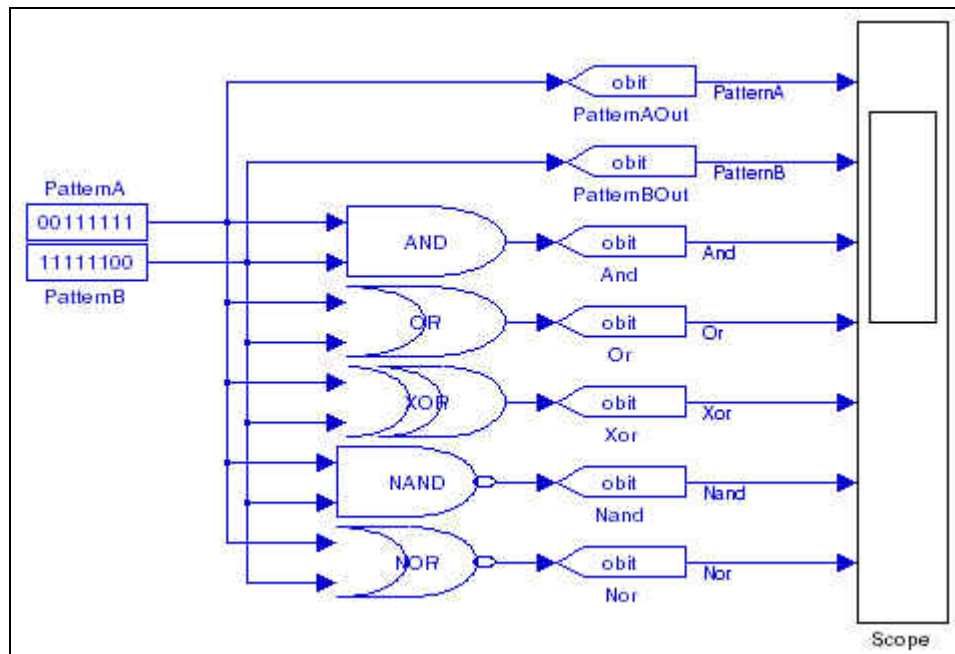
Logical Bus Operator

The `Logical Bus Operator` block performs logical operations on a bus such as AND, OR, XOR, and invert. You can perform masking by entering a mask value in decimal notation, or a shift (rotate) operation by entering the number of bits. By default, a right shift operation preserves the input data sign (for signed inputs).

Table 18–28 shows the `Logical Bus Operator` block inputs and outputs.

Table 18–28. Logical Bus Operator Block Inputs and Outputs

Signal	Direction	Description
d	Input	Input data.
q	Output	Output data.

Table 18–29 shows the `Logical Bus Operator` block parameters.

Table 18–29. Logical Bus Operator Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Specify the bus number format that you want to use.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point, including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point.

Table 18–29. Logical Bus Operator Block Parameters

Name	Value	Description
Logical Operation	AND, OR, XOR, Invert, Shift Left, Shift Right, Rotate Left, Rotate Right	Specify the logical operation to perform.
Mask Value	Integer (Parameterizable)	Specify the mask value for an AND, OR, or XOR operation as an unsigned integer representing the required mask, which must have the same number of bits as the input.
Number of Bits to Shift	User Defined (Parameterizable)	Specify how many bits you want to shift when you chose a shift or rotate operation.
Sign Extend	On or Off	Turn on to preserve the input data sign when right shifting signed data.

Table 18–30 shows the Logical Bus Operator block I/O formats.

Table 18–30. Logical Bus Operator Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
I	I1$_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
O	O1$_{[L1].[R1]}$	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit

Notes to Table 18–30:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 18–12 shows an example with the `Logical Bus Operator` block.

Figure 18–12. Logical Bus Operator Block Example
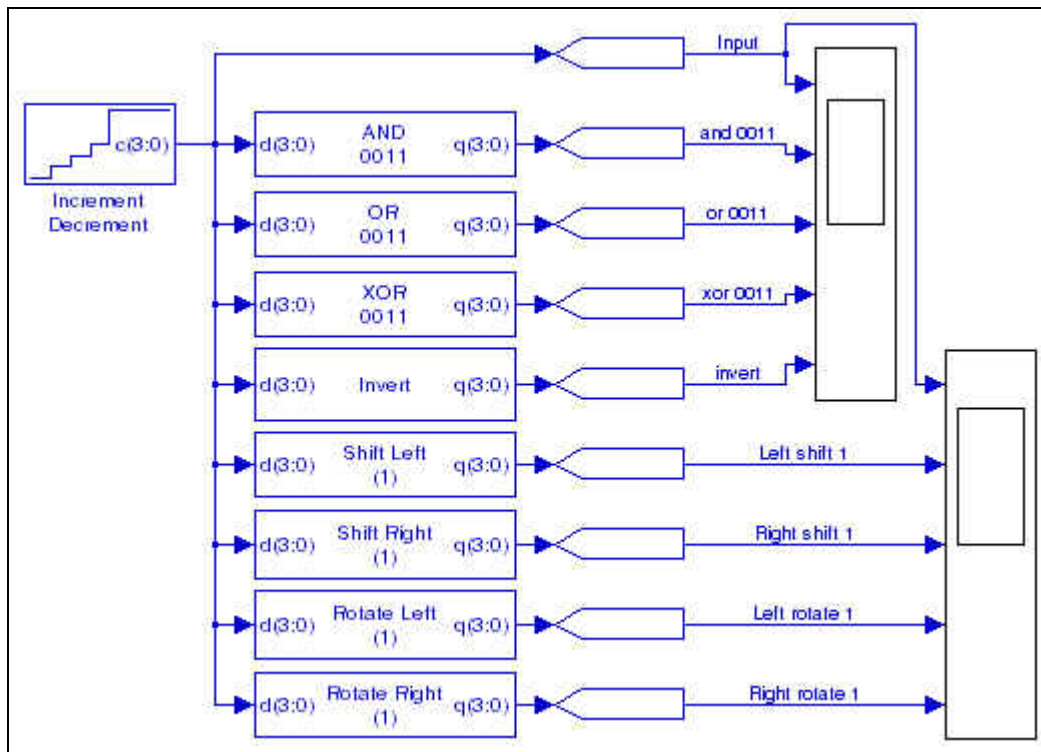
Logical Reduce Operator

The `Logical Reduce Operator` block performs logical reduction operations on a bus such as AND, OR, XOR. The logical operation is applied bit-wise to the input bus to give a single bit result.

Table 18–31 shows the `Logical Reduce Operator` block inputs and outputs.

Table 18–31. Logical Reduce Operator Block Inputs and Outputs

Signal	Direction	Description
d	Input	Input data.
q	Output	Output result.

Table 18–32 shows the `Logical Reduce Operator` block parameters.

Table 18–32. Logical Reduce Operator Block Parameters

Name	Value	Description
Bus Type	Inferred, Signed Integer, Signed Fractional, Unsigned Integer	Specify the bus number format that you want to use.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits to the left of the binary point, including the sign bit.

Table 18–32. Logical Reduce Operator Block Parameters

Name	Value	Description
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point.
Logical Reduction Operation	AND, OR, XOR, NAND, NOR	Specify the logical operation to perform.

Table 18–33 shows the Logical Reduce Operator block I/O formats.

Table 18–33. Logical Reduce Operator Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
O	$O1_{[1]}$	O1: out STD_LOGIC	Explicit

Notes to Table 18–30:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.
(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.
(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 18–13 shows an example with the Logical Reduce Operator block.

Figure 18–13. Logical Reduce Operator Block Example
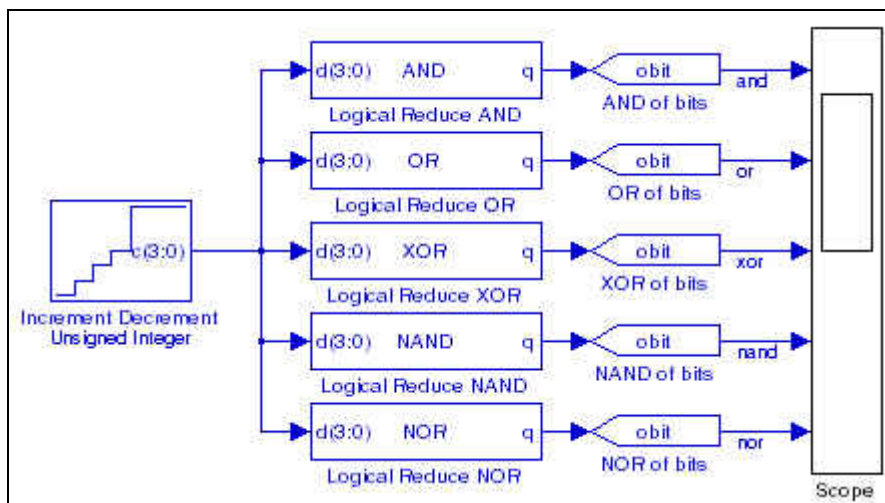
Multiplexer

The Multiplexer block operates as either a n-to-1 one-hot or full-binary bus multiplexer with one select control. The output width of the multiplexer is equal to the maximum width of the input data lines. The block works on any data type and sign extends the inputs if there is a bit width mismatch.

Table 18–34 shows the `Multiplexer` block inputs and outputs.

Table 18–34. Multiplexer Block Inputs and Outputs

Signal	Direction	Description
sel	Input	Select control port.
0-(n-1)	Input	Data input ports.
ena	Input	Optional enable port.
aclr	Input	Optional asynchronous clear port.
<unnamed>	Output	Output port.

Table 18–35 shows the `Multiplexer` block parameters.

Table 18–35. Multiplexer Block Parameters

Name	Value	Description
Number of Input Data Lines	An integer greater than 1 (Parameterizable)	Specify how many inputs the multiplexer has.
Number of Pipeline Stages	>= 0 (Parameterizable)	Specify the number of pipeline stages.
One Hot Select Bus	On or Off	Turn on to use one-hot selection for the bus select signal instead of full binary.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena). This option is available only when the number of pipeline stages is greater than 0.
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (aclr). This option is available only when the number of pipeline stages is greater than 0.

Table 18–36 shows the `Multiplexer` block I/O formats.

Table 18–36. Multiplexer Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[LS].[0]}$ (select input) $I2_{[L2].[R2]}$ …. $Ii_{[Li].[Ri]}$ … $In_{[Ln].[Rn]}$ $In+1_{[1]}$ $In+2_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L2 + R2 - 1} DOWNTO 0) … Ii: in STD_LOGIC_VECTOR({Li + Ri - 1} DOWNTO 0) …. In: in STD_LOGIC_VECTOR({Ln + Rn - 1} DOWNTO 0) In+1: STD_LOGIC In+2: STD_LOGIC	Implicit
O	$O1_{[max(Li)].[max(Ri)]}$ with (0 < I < i + 1)	O1: out STD_LOGIC_VECTOR({max(Li)) + max(Ri) - 1} DOWNTO 0)	Implicit

Notes to Table 18–36:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 18–14 shows an example with the `Multiplexer` block.

Figure 18–14. Multiplexer Block Example
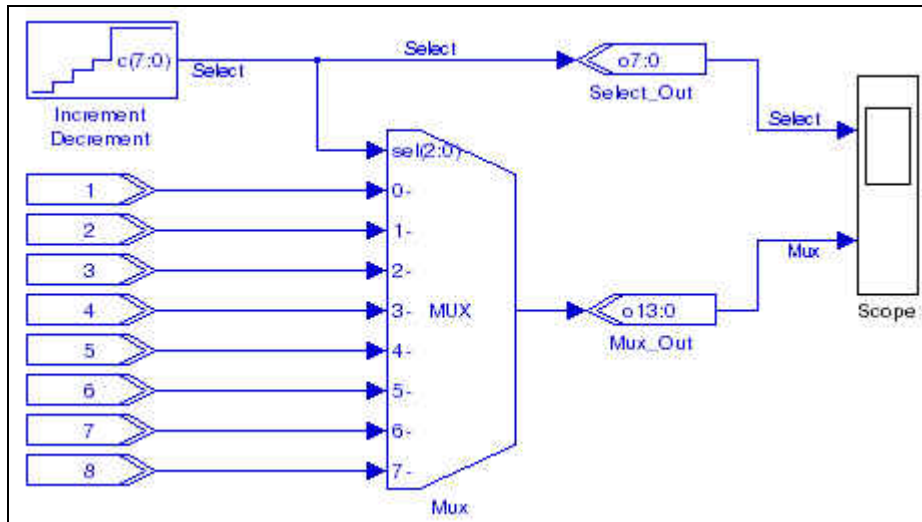
Pattern

The `Pattern` block generates a repeating periodic bit sequence in time. You can enter the required pattern as a binary sequence.

For example, the pattern `01100` outputs the repeating pattern:

> `01100011000110001100011000110001100011000` `1100`

You can change the output data rate for a registered block by feeding the clock enable input with the output of the `Pattern` block.

☞ With a sequence of length 1, the `Pattern` block acts as a constant, holding its output to the specified value at all times. There is no artificial limit to the pattern length.

Table 18–37 shows the `Pattern` block inputs and outputs.

Table 18–37. Pattern Block Inputs and Outputs

Signal	Direction	Description
`ena`	Input	Optional clock enable port.
`sclr`	Input	Optional synchronous clear port.
`<unnamed>`	Output	Output data port.

Table 18–38 shows the `Pattern` block parameters.

Table 18–38. Pattern Block Parameters

Name	Value	Description
Binary Sequence	User Defined	Specify the sequence that you want to use.
Specify Clock	On or Off	Turn on to explicitly specify the clock name.

Table 18–38. Pattern Block Parameters

Name	Value	Description
Clock	User defined (Parameterizable)	Specify the name of the required clock signal.
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear input (`sclr`).

Table 18–39 shows the `Pattern` block I/O formats.

Table 18–39. Pattern Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[1]	I1: in STD_LOGIC	Explicit - optional
	I2[1]	I2: in STD_LOGIC	Explicit - optional
O	O1[1]	O1: out STD_LOGIC	Explicit

Notes to Table 18–39:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 18–15 shows an example with the `Pattern` block.

Figure 18–15. Pattern Block Example
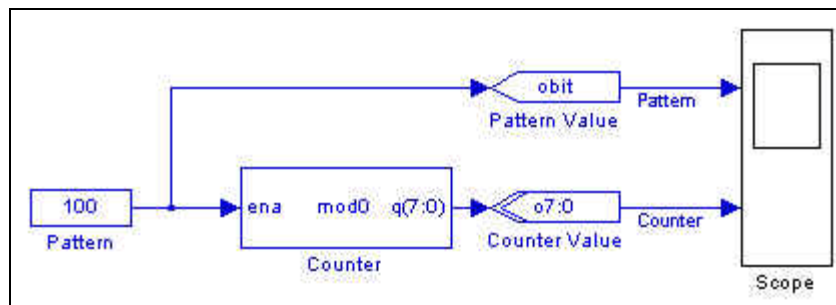
Single Pulse

The `Single Pulse` block generates a single pulse output signal. The output signal is a single bit that takes only the values 1 or 0. The signal generation type can be an impulse, a step up (0 to 1), or a step down (1 to 0).

The output of a impulse starts at 0 changing to 1 after a specified delay and changing to 0 again after a specified length. The output of a step up starts at 0 changing to 1 after a specified delay. The output of a step down starts at 1 changing to 0 after a specified delay.

Table 18–40 shows the Single Pulse block inputs and outputs.

Table 18–40. Single Pulse Block Inputs and Outputs

Signal	Direction	Description
ena	Input	Optional clock enable port.
sclr	Input	Optional synchronous clear port.
<unnamed>	output	Output port.

Table 18–41 shows the Single Pulse block parameters.

Table 18–41. Single Pulse Block Parameters

Name	Value	Description
Signal Generation Type	Step Up, Step Down, Impulse	Specify the type of single pulse.
Impulse Length	Integer (Parameterizable)	Specify the number of clock cycles for which the output signal is transitional from 0 to 1 for an Impulse type output.
Delay	Integer (Parameterizable)	Specify the number of clock cycles that occur before the pulse transition.
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined (Parameterizable)	Specify the name of the required clock signal.
Use Enable Port	On or Off	Turn on to use the clock enable input (ena).
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear input (sclr).

Table 18–42 shows the Single Pulse block I/O formats.

Table 18–42. Single Pulse Block I/O Formats

I/O	Simulink [1]	VHDL	Type
I	I1[1]	I1: in STD_LOGIC	Optional trigger
	I2[1]	I2: in STD_LOGIC	Optional reset
O	O1[1]	O1: out STD_LOGIC	—

Notes to Table 18–42:

(1) I1[1] is an input port. O1[1] is an output port.

Figure 18–16. shows an example of a Single Pulse block.

Figure 18–16. Single Pulse Output Signal Types
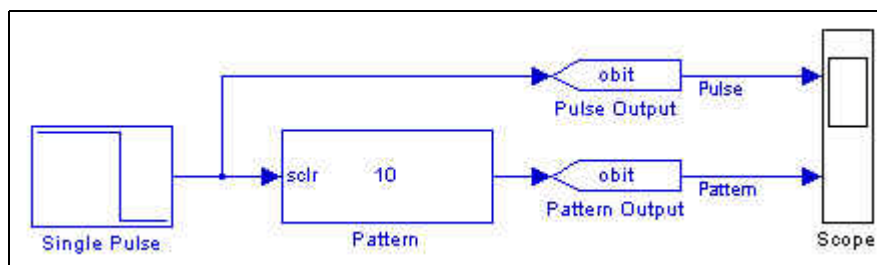
Use the blocks in the Interfaces library to build custom logic blocks that support the Avalon® Memory-Mapped (Avalon-MM) and Avalon Streaming (Avalon-ST) interfaces.

The Interfaces library contains the following blocks:

■ Avalon-MM Master

■ Avalon-MM Slave

■ Avalon-MM Read FIFO

■ Avalon-MM Write FIFO

■ Avalon-ST Packet Format Converter (Deprecated)

■ Avalon-ST Sink

■ Avalon-ST Source

Avalon Memory-Mapped Blocks

The Avalon-MM blocks automate the process of specifying master and slave ports that are compatible with the Avalon-MM bus.

After you build a model of your DSP Builder peripheral, you can add the following blocks to control the peripheral's inputs and outputs:

■ Configurable master and slave blocks that contain the ports required to connect peripherals that use the Avalon-MM bus.

■ Wrapped versions of the Avalon-MM slave that implement an Avalon-MM read FIFO buffer and Avalon-MM write FIFO.

For more information about the Avalon-MM interface, refer to the *Avalon Interface Specifications*.

After you synthesize your model and compile it in the Quartus II software, use Qsys to add it to your Nios II system.

Your design automatically appears under the `DSP Builder` category in the Qsys component browser peripherals listing if the MDL file is in the same directory as the SOPC file.

A file **mydesign.mdl** creates a component `mydesign_interface` in Qsys.

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

Avalon-MM Master

The `Avalon-MM Master` block defines a collection of ports for connection to an Qsys system when your design functions as an Avalon-MM master interface.

Table 19–1 lists the signals supported by the `Avalon-MM Master` block.

Table 19–1. Signals Supported by the Avalon-MM Master Block

Signal	Direction	Description
waitrequest	Input	This signal forces the master port to wait until you are ready to proceed with the transfer.
address	Output	The address signal represents a byte address but is asserted on word boundaries only.
read	Output	Available with **Read** or **Read/Write** address type. Read request signal. Not required if there are no read transfers. If used, also use `readdata`.
readdata	Input	Available when **Read** or **Read/Write** address type is chosen. Data lines for read transfers. Not required if there are no read transfers. If used, also use `read`.
write	Output	Available when **Write** or **Read/Write** address type is chosen. Write request signal. Not required if there are no write transfers. If used, also use `writedata`.
writedata	Output	Available when **Write** or **Read/Write** address type is chosen. Data lines for write transfers. Not required if there are no write transfers. If used, also use `write`.
byteenable	Output	Available when **Write** or **Read/Write** address type is chosen and the bit width is greater than 8. Enables specific byte lane(s) during write transfers to memories of width greater than 8 bits. All `byteenable` lines must be enabled during read transfers.
endofpacket	Input	Available when **Allow Flow Control** is on. Indicates an end-of-packet condition.
readdatavalid	Input	Available when **Allow Pipeline Transfers** is on. Use for pipelined read transfers with latency. Indicates that valid data is present on the `readdata` lines.
flush	Output	Available when **Allow Pipeline Transfers** and **Use Flush Signal** are on. Can be asserted to clear any pending transfers in the pipeline.
burstcount	Output	Available when **Allow Burst Transfers** is on. Indicates the number of transfers in a burst.
irq	Input	Available when **Receive IRQ** is on. Indicates when one or more ports have requested an interrupt.
irqnumber	Input	Available when **Receive IRQ** is on and IRQ mode is set to Prioritized. Indicates the interrupt priority. Lower value means higher priority.

☞ The direction in Table 19–1 refers to the direction in respect of the DSP Builder block interface.

Table 19–2 shows the `Avalon-MM Master` block parameters.

Table 19–2. Avalon-MM Master Block Parameters

Name	Value	Description
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined	Specifies the clock signal name.
Address Width	1–32	Specifies the number of address bits.
Address Type	Read, Write, Read/Write	The address type for the bus.
Data Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. Read and write buses must have the same number of bits.

Table 19–2. Avalon-MM Master Block Parameters

Name	Value	Description
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Allow Byte Enable	On or Off	Turn on to use the `Byte Enable` signal. This option is available when the address type is set to **Write** or **Read/Write** and the bit width is greater than 8.
Allow Flow Control	On or Off	Turn on to enable flow control. Flow control allows a slave port to regulate incoming transfers from a master port, so that a transfer only begins when the slave port indicates that it has valid data or is ready to receive data.
Allow Pipeline Transfers	On or Off	Turn on to allow pipeline transfers. Pipeline transfers increase the bandwidth for synchronous slave peripherals that require several cycles to return data for the first access, but can return data every cycle thereafter. This option is available when the address type is **Read** or **Read/Write**.
Use Flush Signal	On or Off	Turn on to clear any pending transfers in the pipeline. This option is available when **Allow Pipeline Transfers** is on.
Allow Burst Transfers	On or Off	Turn on to allow burst transfers. A burst executes multiple transfers as a unit, and maximize the throughput for slave ports that achieves the greatest efficiency when handling multiple units of data from one master port at a time.
Maximum Burst Size	2–32	Specifies the maximum width of a burst transfer. This option is available when **Allow Burst Transfers** is on.
Receive IRQ	On or Off	Turn on to enable interrupt requests from the slave port.
IRQ Mode	Prioritized, Individual Signals	The interrupt request mode. This option is available when **Receive IRQ** is on.

Figure 19–1 shows an `Avalon-MM Master` block with all signals enabled.

Figure 19–1. Avalon-MM Master Block with All Signals Enabled
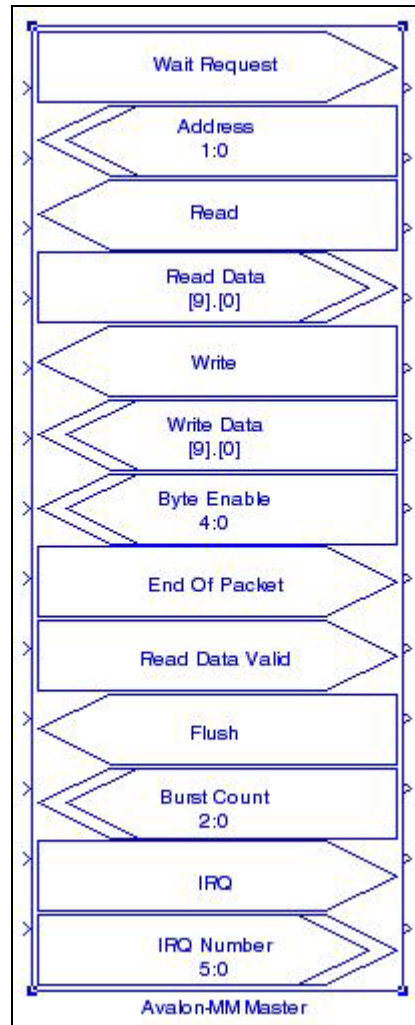
Avalon-MM Slave

The `Avalon-MM Slave` block defines a collection of ports for connection to an Qsys system when your design functions as an Avalon-MM slave interface.

Table 19–3 lists the signals supported by the `Avalon-MM Slave` block.

Table 19–3. Signals Supported by the Avalon-MM Slave Block

Signal	Direction	Description
address	Output	Address lines to the slave port. Specifies a word offset into the slave address space.
read	Output	Available when **Read** or **Read/Write** address type is chosen. Read-request signal. Not required if there are no read transfers. If used, also use `readdata`.
readdata	Input	Available when **Read** or **Read/Write** address type is chosen. Data lines for read transfers. Not required if there are no read transfers. If used, also use `read`.
write	Output	Available when **Write** or **Read/Write** address type is chosen. Write-request signal. Not required if there are no write transfers. If used, also use `writedata`.
writedata	Output	Available when **Write or Read/Write** address type is chosen. Data lines for write transfers. Not required if there are no write transfers. If used, also use `write`.
byteenable	Output	Available when **Allow Byte Enable** is on and the bit width is greater than 8. Byte-enable signals to enable specific byte lane(s) during write transfers to memories of width greater than 8 bits. If used, also use `writedata`.
readyfordata	Input	Available when **Write** or **Read/Write** access is chosen and **Allow Flow Control** is on. Indicates that the peripheral is ready for a write transfer.
dataavailable	Input	Available when **Read** or **Read/Write** access is chosen and **Allow Flow Control** is on. Indicates that the peripheral is ready for a read transfer.
endofpacket	Input	Available when **Allow Flow Control** is on. Indicates an end-of-packet condition.
readdatavalid	Input	Available when **Allow Pipeline Transfers** is on and variable read latency is chosen. Marks the rising clock edge when `readdata` asserts.
waitrequest	Input	Available when variable wait-state format is chosen. Use to stall the interface when the slave port cannot respond immediately.
beginbursttransfer	Output	Available when **Allow Burst Transfers** is on. Asserted for the first cycle of a burst to indicate when a burst transfer is starting.
burstcount	Output	Available when **Allow Burst Transfers** is on. Indicates the number of transfers in a burst. If used, also use `waitrequest`.
irq	Input	Available when **Output IRQ** is on. Interrupt request. Asserted when a port needs to be serviced.
begintransfer	Output	Available when **Receive Begin Transfer** is on. Asserted during the first cycle of every transfer.
chipselect	Output	Available when **Use Chip Select** is on. The slave port ignores all other Avalon-MM signal inputs unless `chipselect` is asserted.

☞ The direction in Table 19–3 refers to the direction in respect of the DSP Builder block interface.

Table 19–4 shows the `Avalon-MM Slave` block parameters.

Table 19–4. Avalon-MM Slave Block Parameters

Name	Value	Description
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined	Specifies the clock signal name.
Address Width	1–32	Specifies the number of address bits.

Table 19–4. Avalon-MM Slave Block Parameters

Name	Value	Description
Address Alignment	Native, Dynamic	Use native address alignment or dynamic bus sizing.
Address Type	Read, Write, Read/Write	The address type for the bus.
Data Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. Read and write buses must have the same number of bits.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Allow Byte Enable	On or Off	Turn on to use the `Byte Enable` signal. This option is available only when the address type is set to **Write** or **Read/Write**.
Allow Flow Control	On or Off	Turn on to enable flow control. Flow control allows a slave port to regulate incoming transfers from a master port, so that a transfer only begins when the slave port indicates that it has valid data or is ready to receive data.
Allow Pipeline Transfers	On or Off	Turn on to allow pipeline transfers. Pipeline transfers increase the bandwidth for synchronous slave peripherals that require several cycles to return data for the first access, but can return data every cycle thereafter. This option is available only when the address type is set to **Read** or **Read/Write**.
Wait-State Format	Fixed, Variable	The required wait-state format.
Read Wait-State Cycles	0–255	Specifies the number of read wait-state cycles. This option is available only when the wait-state format is set to Fixed.
Write Wait-State Cycles	0–255	Specifies the number of write wait state cycles. This option is available only when the wait-state format is set to Fixed.
Read Latency Format	Fixed, Variable	The required read latency format. This option is available only when **Allow Pipeline Transfers** is on.
Read Latency Cycles	0–8	Specifies the pipeline read latency. Latency determines the length of the data phase, independently of the address phase. For example, a pipelined slave port (with no wait-states) can sustain one transfer per cycle, even though it may require several cycles of latency to return the first unit of data. This option is available only when **Allow Pipeline Transfers** is on and **Fixed** read latency format is set.
Allow Burst Transfers	On or Off	Turn on to allow burst transfers. A burst executes multiple transfers as a unit, and maximize the throughput for slave ports that achieves the greatest efficiency when handling multiple units of data from one master port at a time.
Maximum Burst Size	4–2^{32}	Specifies the maximum width of a burst transfer. This option is available only when **Allow Burst Transfer** is on.
Output IRQ	On or Off	Turn on to enable interrupt requests from the slave port.
Receive BeginTransfer	On or Off	Turn on to receive `begintransfer` signals.
Use Chip Select	On or Off	Turn on to enable the `chipselect` signal.

Figure 19–2 shows an `Avalon-MM Slave` block with all signals enabled.

Figure 19–2. Avalon-MM Slave Block with All Signals Enabled
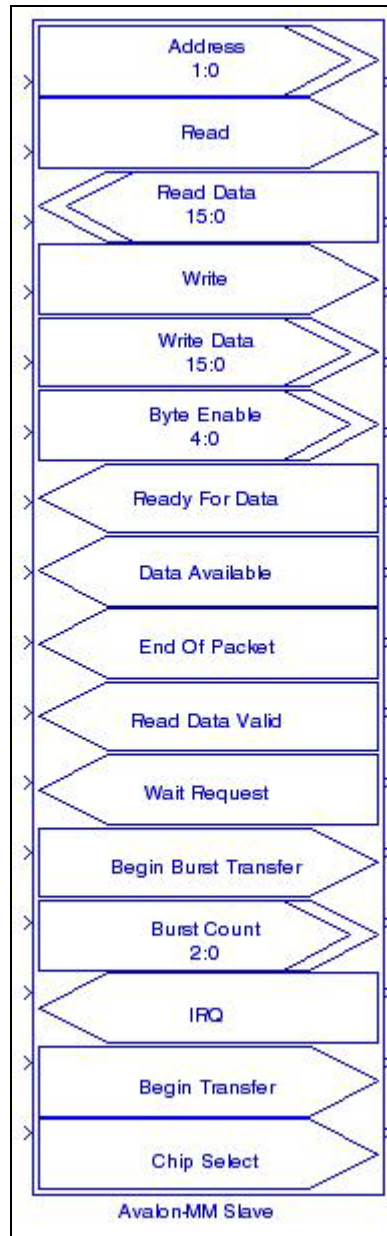
> For general information about Avalon-MM blocks refer to "Avalon Memory-Mapped Blocks" on page 19–1.

Avalon-MM Read FIFO

The `Avalon-MM Read FIFO` block is essentially an `Avalon-MM Slave` block configured to implement a read FIFO. It is accessed by other Avalon-MM peripherals to obtain data when connected in Qsys.

For information about the `Avalon-MM Slave` block, refer to "Avalon-MM Slave" on page 19–4.

Table 19–5 lists the signals supported by the `Avalon-MM Read FIFO` block.

Table 19–5. Signals Supported by the Avalon-MM Read FIFO Block

Signal	Direction	Description
Stall	Input	This port must be connected to Simulink blocks. It simulates stall conditions of the Avalon-MM bus and hence back pressure to the SOPC component. For any simulation cycle where the `Stall` signal is asserted, no Avalon-MM reads take place and the internal FIFO buffer fills. When full, the `Ready` output is de-asserted so that no data is lost.
Data	Input	This port should be connected to DSP Builder blocks and should be connected to outgoing data from the user design.
DataValid	Input	This port should be connected to DSP Builder blocks and should be asserted whenever the signal on the `Data` port corresponds to real data.
TestDataOut	Output	This port should be connected to Simulink blocks and corresponds to the data received over the Avalon-MM bus.
TestDataValid	Output	This port should be connected to Simulink blocks and is asserted whenever `TestDataOut` corresponds to real data.
Ready	Output	When asserted, indicates that the block is ready to receive data.

Table 19–6 shows the `Avalon-MM Read FIFO` block parameters.

Table 19–6. Avalon-MM Read FIFO Block Parameters

| Name	Value	Description
Data Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
FIFO Depth	> 2	Specifies the depth of the FIFO.

Figure 19–3 shows an `Avalon-MM Read FIFO` block.

Figure 19–3. Avalon-MM Read FIFO

Figure 19–4 shows the content of the `Avalon-MM Read FIFO` block.

Figure 19–4. Avalon-MM Read FIFO Content
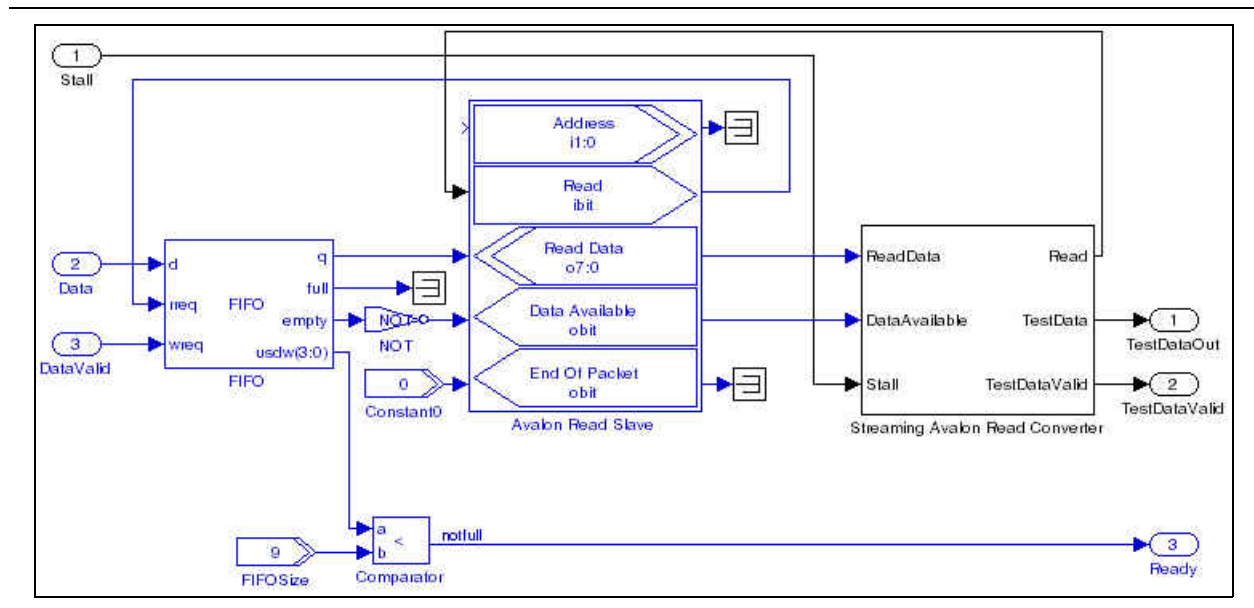
Avalon-MM Write FIFO

The `Avalon-MM Write FIFO` block is essentially an `Avalon-MM Slave` block configured to implement a write FIFO.

For information about the `Avalon-MM Slave` block, refer to "Avalon-MM Slave" on page 19–4.

Table 19–7 lists the signals supported by the `Avalon-MM Write FIFO` block.

Table 19–7. Signals Supported by the Avalon-MM Write FIFO Block

Signal	Direction	Description
TestData	Input	This port must be connected to Simulink blocks. It provides simulation data to the Avalon-MM write FIFO. The data is passed to the `DataOut` port one cycle after the `Ready` input port is asserted.
Stall	Input	This port must be connected to Simulink blocks. It simulates stall conditions of the Avalon-MM bus and hence underflow to the SOPC component. For any simulation cycle where `Stall` is asserted, the test data is cached by the Avalon-MM write converter and released in order, one sample per clock, when stall is de-asserted.
Ready	Input	This port must be connected to DSP Builder blocks. It indicates that the downstream hardware is ready for data.
DataOut	Output	This port should be connected to DSP Builder blocks and corresponds to the oldest unsent data sample received on the `TestData` port.
DataValid	Output	This port should be connected to DSP Builder blocks and is asserted whenever `DataOut` corresponds to real data.

Table 19–8 shows the Avalon-MM Write FIFO block parameters.

Table 19–8. Avalon-MM Write FIFO Block Parameters

Name	Value	Description
Data Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
FIFO Depth	> 2	Specifies the depth of the FIFO buffer.

Figure 19–5 shows an Avalon-MM Write FIFO block.

Figure 19–5. Avalon-MM Write FIFO

Figure 19–6 shows the content of the Avalon-MM Write FIFO block.

Figure 19–6. Avalon-MM Write FIFO Content
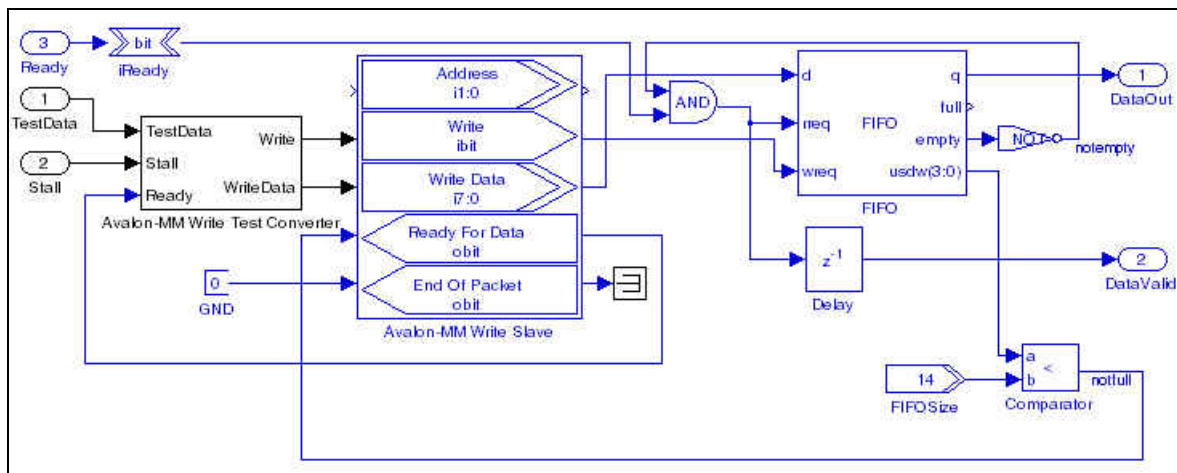
Avalon Streaming Blocks

The Avalon Streaming blocks automate the process of specifying ports that are compatible with an Avalon-ST interface. The blocks include an Avalon-ST Packet Format Converter (Deprecated), Avalon-ST Sink and Avalon-ST Source.

For information about the Avalon-ST interface, refer to the *Avalon Interface Specifications*.

Avalon-ST Packet Format Converter (Deprecated)

This block is deprecated. Use Avalon-ST interfacing blocks available in DSP Builder Standard or Advanced Blocksets to implement custom transformations.

The `Avalon-ST Packet Format Converter` (PFC) block transforms packets received from one block to a different packet format required by another block.

The PFC takes packet data from one or more input interfaces, and provides field reassignment in time and space to one or more output packet interfaces. You specify the input packet format and the desired output packet format, then the appropriate control logic automatically generates.

The PFC operates on a single clock domain, and supports multicast data, where an input field is broadcast copied to multiple output fields. The ready latency of the PFC block is zero and it can only connect to other Avalon-ST interfaces with a ready latency of zero.

☞ Verilog HDL generates for the PFC block and you must therefore have a license that supports Verilog HDL when simulating in ModelSim.

Figure 19–7 shows the basic operation of the PFC.

Figure 19–7. Basic Packet Format Converter
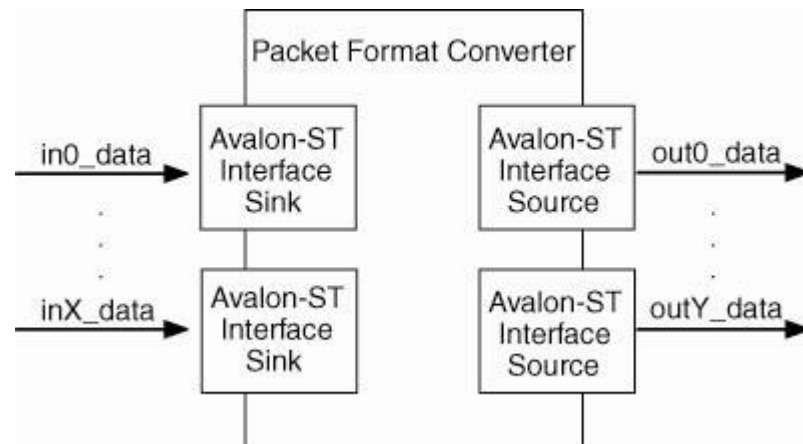
The PFC performs data mapping on a packet by packet basis, so that there is exactly one input packet on each input interface for one output packet on each output interface. The interface with the longest packet limits the packet rate of the converter.

When the PFC has multiple output interfaces, the packets on each output interface are aligned so that the `startofpacket` signal is presented on the same clock cycle.

If each interface supports fixed-length packets, you can select a **Multi-Packet Mapping** option. The PFC can then map fields from multiple input packets to multiple output packets. The PFC does not support bursts or blocks on its output interfaces.

Use the **Split Data** option to split the input or output data signals across additional ports named `data0` through `dataN`.

Each input interface consists of the `ready`, `valid`, `startofpacket`, `endofpacket`, `empty`, and `data` signals. Each output interface has an additional `error` signal that asserts to indicate a frame delineation error.

For more information about these signal types, refer to the *Avalon Interface Specifications*.

☞ The PFC block does not support Avalon-ST bursts or blocks on its output interfaces.

Table 19–9 lists the signals supported by the `Avalon-ST Packet Format Converter` block.

Table 19–9. Signals Supported by the Avalon-ST Packet Format Converter Block

Signal	Direction	Description
reset_n	Input	Active-low reset signal.
inX_dataN	Input	Data input bus for sink interface *X*.
inX_empty	Input	Indicates the number of empty symbols for sink interface *X* during cycles that mark the end of a packet.
inX_endofpacket	Input	This signal marks the active cycle containing the end of the packet for sink interface *X*.
inX_startofpacket	Input	This signal marks the active cycle containing the start of the packet for sink interface *X*.
inX_valid	Input	Indicates DSP Builder can accept data for sink interface *X*.
outY_ready	Input	Indicates that the sink driven by the source interface *Y* is ready to accept data.
aclr	Input	Optional asynchronous clear port.
inX_ready	Output	Indicates that sink interface *X* is ready to output data.
outY_dataN	Output	Data output bus for source interface *Y*.
outY_empty	Output	Indicates the number of empty symbols for source interface *Y* during cycles that mark the end of a packet.
outY_endofpacket	Output	This signal marks the active cycle containing the end of the packet for source interface *Y*.
outY_startofpacket	Output	This signal marks the active cycle containing the start of the packet for source interface *Y*.
outY_valid	Output	Indicates that valid data is available on source interface *Y*.
outYerror	Output	Indicates an error condition when asserted high.

Table 19–10 shows the `Avalon-ST Packet Format Converter` block parameters.

Table 19–10. Avalon-ST Packet Format Converter Block Parameters

Name	Value	Description
Number of Sinks	1–16	Specifies the number of sink interfaces *X*.
Number of Sources	1–16	Specifies the number of source interfaces *Y*.

Table 19–10. Avalon-ST Packet Format Converter Block Parameters

Name	Value	Description
Split Data	On or Off	When on, the data signals on the sink and source interface are split into signals named `data0` through `dataN` with widths corresponding to the specified symbol width.
Multi-Packet Mapping	On or Off	When off, one input packet is matched to one output packet and the input and output packets must have the name number of instances in each field. When on, the PFC maps the input packets to output packets such that all instances of every data field are accounted for.
Symbol Width	>= 1	Specifies the number of bits per symbol that all the PFC sink and source interfaces use.
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear input (`aclr`).
Sink Format X	string	A quoted string or MATLAB variable that describes the packet format for sink interface *X*.
Sink X Symbols Per Beat	1–32	Specifies the number of symbols per beat for sink interface *X*.
Source Format Y	string	A quoted string or MATLAB variable that describes the packet format for source interface *Y*.
Source Y Symbols Per Beat	1–32	Specifies number of symbols per beat for source interface *Y*.

PFC Data Flow

The PFC spools data into a FIFO-like memory as it arrives, and spools it out in a different order as it leaves. DSP Builder can provide the data at the output of each interface as soon as it writes it into the memory and all previous output data is transferred. When the PFC has multiple output interfaces, the `startofpacket` signal for all the interfaces is asserted at the same time.

The PFC stops data input on input interfaces by deasserting the `ready` signal whenever there is a risk of overwriting data that it has not yet output. If a downstream block pauses output data by deasserting the `ready` signal to the PFC, the PFC accepts data until it risk overwriting unsent data. At this point, the PFC deasserts the `ready` signals on its own input interface, causing the upstream block to stop sending data.

In a similar way, if the upstream block starves the PFC of data by deasserting the `valid` signals to the PFC, then the PFC output interface continues to send data until the memory drains. It then stops sending data by deasserting the output `valid` signals.

For multiple interface PFC blocks, back pressuring an output interface or starving an input interface affects all other interfaces. When an output interface is back pressured, the input interfaces are back pressured, causing the other outputs to be starved of data. Likewise, if an input interface is starved of data, the output interfaces eventually stop, causing the other input interfaces to be back pressured.

Packet Format Description

For each input and output interface, the number of symbols per beat and the packet description describe the basic format of the packet.

The number of symbols per beat defines, for each interface, the number of symbols that present in parallel on every active cycle. The packet description is a string that describes the fields in the packet.

A basic packet description is a comma-separated list of field names, where a field name includes any of the characters a-z, A-Z, _, or 0-9 but must start with a letter. For example: Field1, Red, Green, Blue, and DestinationAddress. Field names are case sensitive. Do not use whitespace in a packet description.

If fields repeat in a packet, parentheses delineate the repeated group (of one or more fields), and a positive integer follows the group to indicate the number of repeats. The following examples describe the parenthesis further:

■ Dest,Source,(Data)128,(CRC)4 indicates a packet that has destination and source address symbols followed by 128 data symbols and 4 CRC symbols.

■ (Red,Green,Blue)100 refers to a frame with 100 repetitions of a symbol of Red, followed by a symbol of Green, followed by a symbol of Blue.

■ Nest repeats, so that (F1,(F2)3,F3)2,F4 is equivalent to (F1,F2,F2,F2,F3)2,F4 or F1,F2,F2,F2,F3,F1,F2,F2,F2,F3,F4.

Use a + instead of a positive integer, such as (Red,Green,Blue)+, to repeat a group an unspecified number of times in a packet. However, such a group must compose the entire packet. Therefore, none of the following examples are valid: A,(B,C)+, (A,B)+,C, or((A)+)2.

Table 19–11 summarizes the packet description syntax for the PFC.

Table 19–11. Packet Description Syntax

Packet Descriptor:	Group | (Group)+ where + indicates that the preceding Group is repeated an unknown number of times
Group	repeatedGroup | simpleGroup
repeatedGroup	(Group)N where N is a positive integer indicating the number of times the preceding group is repeated
simpleGroup	FieldName[,Group]

Table 19–12 shows some example packets. All these examples use the convention *<packet description>* / *<symbols per beat>*, so that R,G,B/2 refers to an interface where the packet description is R,G,B and the number of symbols per beat is 2.

Table 19–12. Packet Description Examples
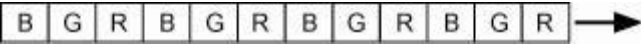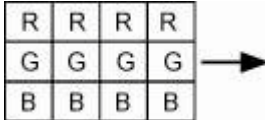
Packet Description / Symbols Per Beat	Example Packets
(R,G,B)4	
(R,G,B)4/3	

Table 19–12. Packet Description Examples

Packet Description / Symbols Per Beat	Example Packets
(Y,Cr,Y,Cb)/2	
((A)2,B,C,(A)2,B,D)3/4	
((((A)2,B)2,C)2,D)2	

Figure 19–8 shows an example of the packet formats for a PFC with two input and two output interfaces.

Figure 19–8. Example of a Packet Format Converter with Two Input and Two Output Interfaces
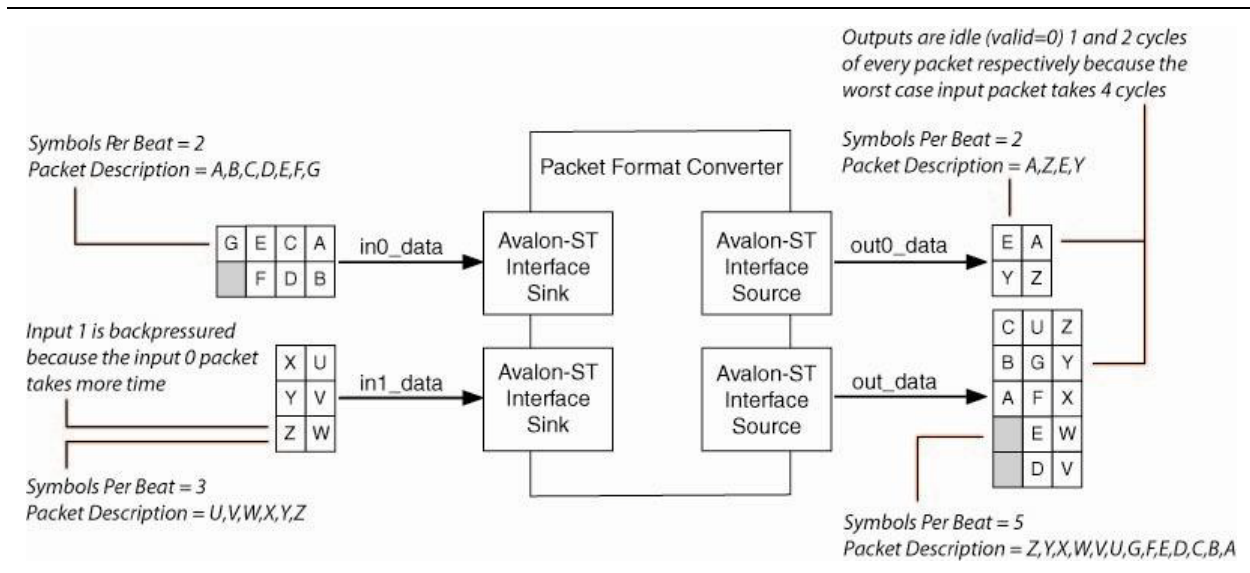
Packet Mapping

Packet mapping is the process of determining where the data for each field in each output interface is coming from (as an {input interface, position} pair).

To achieve packet mapping, compare the field name strings. For example, the source of data for the `Red` field in a given output interface is the field on an input interface with the name `Red`. It is not valid for any field name to exist on multiple-input interfaces; no two input interfaces may have a `Red` field. It is valid, however, for multiple-output interfaces to have the same field; you may copy the `Red` data to two or more output interfaces.

A single input or output interface can have multiple instances of the same field. For example, `Red,Green,Red,Blue` represents a packet with two red symbols per packet. The PFC matches the *nth* instance of a field on an input interface to the *nth* instance of the same field on an output interface. If an output interface has `Blue,Green,Red,Red`, the data for the first `Red` field is taken from the first `Red` field in the input packet.

Each output interface may or may not use a given input field, but unless you set the **Multi-Packet Mapping** option (and if the input field is used) there must be the same number of instances of the field in each output as there is in the input. For example, `Green` and `Red,Red,Green` are both valid, but `Red,Green` is not.

Multi-Packet Mapping

Set the **Multi-Packet Mapping** option, so that the PFC is not limited to mapping a single input packet on each port to a single output packet on each port. It can map multiple input packets to multiple output packets.

For example, `(Red,Green,Blue)2` maps to `(Red,Green,Blue)3` by using three input packets for every two output packets.

The ratio of input fields to output fields must be constant.

For example, `Red,Red,Green,Blue` does not map to `(Red,Green,Blue)2` because each output packet requires one input packet for `Red`, but two input packets for `Green` and `Blue`.

DSP Builder supports multiple interfaces but the packet ratio must be constant across all {input interface, output interface} pairs.

For example, two input interfaces with the formats `(Red,Green)2` and `Blue` map to output interface `(Red)6,Blue(3),Green(6)` because three input packets are required for two output packets for all input and output pairs. The same inputs do not map to `(Red)3,Blue(3),Green(3)`, because to make two output packets, three of the first input's packets and six of the second input's packets are required.

☞   DSP Builder does not support packets of unknown length.

Error Handling

The PFC contains internal counters that keep track of the current position in the packet for each input and uses these counters to detect frame delineation errors. Every time a `startofpacket` or `endofpacket` signal asserts on an input interface, the PFC uses its knowledge of the frame structure to ensure that the assertion is on a valid cycle. For PFC variants where the packet size is known, the PFC also checks that the `startofpacket` and `endofpacket` signals assert when they should do, and are not missed.

The PFC only has a single output error bit to report frame delineation errors. The output error bit asserts on all outputs as soon as DSP Builder detects an error, and it asserts for each output interface independently until an endofpacket asserts for that output interface.

After the endofpacket asserts, the PFC presents no more data to that output interface. When all output interfaces stop, the PFC resets and resumes normal operation. The PFC stops independently on the endofpacket signal for each output, and components downstream of the PFC should never see partial frames.

While errors assert to the output interfaces and the core is reset, the input interfaces are not back pressured. This action prevents loss of any synchronization between input interfaces by uneven back pressuring during error conditions.

When the PFC starts again, it waits until it sees a startofpacket signal for each input interface before accepting data for that interface. It is not possible to guarantee synchronization of output interfaces when frame delineation errors are present.

The PFC does not support relaying errors from an upstream component to a downstream component.

When simulating the PFC block, connect the reset port to a pulse generator (such as the Single Pulse block in the DSP Builder Gate & Control library) that is configured to output an initial 0, then a 1 for the remainder of the simulation.

Avalon-ST Sink

The Avalon-ST Sink block defines a collection of ports for connection to an Qsys system when your design functions as an Avalon-ST sink.

For information about the Avalon-ST interface, refer to the *Avalon Interface Specifications*.

Table 19–13 lists the signals supported by the Avalon-ST Sink block.

Table 19–13. Signals Supported by the Avalon-ST Sink Block

| Signal	Direction	Description
DataIn	Input	Data input bus.
Valid	Input	Data valid signal that indicates the validity of the input data signals.
Ready	Output	Data input ready signal. Indicates that the sink can accept data.
startofpacket	Input	This signal is available when **Use startofpacket** is on and marks the active cycle containing the start of the packet.
endofpacket	Input	This signal is available when **Use endofpacket** is on and marks the active cycle containing the end of the packet.
empty	Input	This signal is available when **Use empty** is turned on and the bit width is greater than the symbol width. It specifies how many of the symbols in a packet are empty. For example, a 32-bit wide bus with 8-bit symbols can have an empty value from 0 to 3.

Table 19–14 shows the `Avalon-ST Sink` block parameters.

Table 19–14. Avalon-ST Sink Block Parameters

Name	Value	Description
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined	Specifies the clock signal name.
Data Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. Read and write buses must have the same number of bits.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Symbol Width	>= 1	Specifies the symbol width in bits.
Use endofpacket	On or Off	When this option is on, the `endofpacket` port is available on the `Avalon-ST Sink` block.
Use startofpacket	On or Off	When this option is on, the `startofpacket` port is available on the `Avalon-ST Sink` block.
Use empty	On or Off	When this option is on and the bit width is greater than the symbol width, the `empty` port is available on the `Avalon-ST Sink` block.
Ready Latency	0 or 1	Defines the relationship between assertion or deassertion of the `Ready` signal and cycles the ones ready for data transfer separately for each interface.

Figure 19–9 shows an `Avalon-ST Sink` block with all signals enabled.

Figure 19–9. Avalon-ST Sink Block with All Signals Enabled
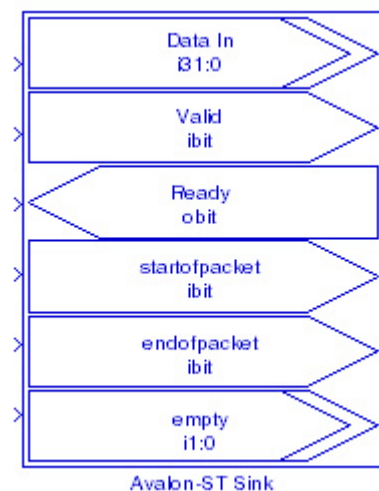
Avalon-ST Source

The `Avalon-ST Source` block defines a collection of ports for connection to an Qsys system when your design functions as an Avalon-ST source.

For information about the Avalon-ST interface, refer to the *Avalon Interface Specifications*.

Table 19–15 lists the signals supported by the `Avalon-ST Source` block.

Table 19–15. Signals Supported by the Avalon-ST Source Block

Signal	Direction	Description
DataOut	Output	Data input bus.
Valid	Output	Data valid signal that indicates the validity of the output data signals.
Ready	Input	Data output ready signal. Indicates that the source can accept data.
startofpacket	Output	This signal is available when the **Use startofpacket** parameter is on and marks the active cycle containing the start of the packet.
endofpacket	Output	This signal is available when the **Use endofpacket** parameter is on and marks the active cycle containing the end of the packet.
empty	Output	This signal is available when **Use empty** is turned on and the bit width is greater than the symbol width. It specifies how many of the symbols in a packet are empty. For example, a 32-bit wide bus with 8-bit symbols can have an empty value from 0 to 3.

Table 19–16 on page 19–19 shows the `Avalon-ST Source` block parameters.

Table 19–16. Avalon-ST Source Block Parameters

Name	Value	Description
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined	Specifies the clock signal name.
Data Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. Read and write buses must have the same number of bits.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Symbol Width	1–512	Specifies the symbol width in bits.
Use endofpacket	On or Off	When this option is on, the `endofpacket` port is available on the `Avalon-ST Source` block.
Use startofpacket	On or Off	When this option is on, the `startofpacket` port is available on the `Avalon-ST Source` block.
Use empty	On or Off	When this option is on and the bit width is greater than the symbol width, the `empty` port is available on the `Avalon-ST Sink` block.
Ready Latency	0 or 1	Defines the relationship between assertion/deassertion of the `Ready` signal and cycles the ones ready for data transfer separately for each interface.

Figure 19–10 shows an Avalon-ST Source block with all signals enabled.

Figure 19–10. Avalon-ST Source Block with All Signals Enabled
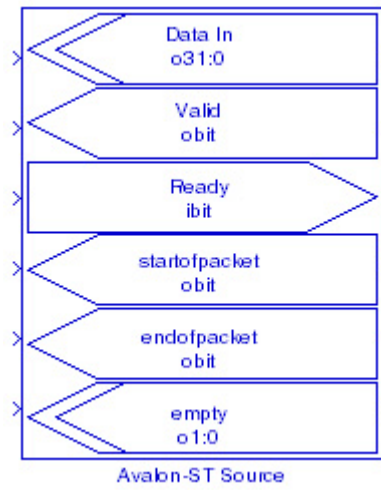
The blocks in the IO & Bus library manipulate signals and buses to perform operations such as truncation, saturation, bit extraction, or bus format conversion.

The IO & Bus library contains the following blocks:

- AltBus
- Binary Point Casting
- Bus Builder
- Bus Concatenation
- Bus Conversion
- Bus Splitter
- Constant
- Extract Bit
- Global Reset
- GND
- Input
- Non-synthesizable Input
- Non-synthesizable Output
- Output
- Round
- Saturate
- VCC

AltBus

The AltBus block modifies the bus format of a DSP Builder signal. Only use this block as an internal node in a system, not as an input to or output from the system. If the specified bit width is wider than the input bit width, the bus is sign extended to fit. If it is smaller than the input bit width, you can specify to either truncate or saturate the excess bits.

Table 20–1 shows the `AltBus` block parameters.

Table 20–1. AltBus Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, Single Bit	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Saturate Output	On or Off	When this option is on, if the output is greater than the maximum positive or negative value to be represented, the output is forced (or saturated) to the maximum positive or negative value, respectively. When off, the MSB is truncated.

Table 20–2 shows the `AltBus` block I/O formats.

Table 20–2. AltBus Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit - Optional
O	$O1_{[LP].[RP]}$	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 20–2:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Table 20–3 and Figure 20–1 on page 20–3 illustrate how a floating-point number (4/3 = 1.3333) is cast into signed binary fractional format with three different binary point locations.

Table 20–3. Floating-Point Numbers Cast to Signed Binary Fractional

Bus Notation	Input	Simulink	VHDL
[4].[1]	4/3	1.00	2
[2].[3]	4/3	1.25	10
[1].[4]	4/3	-0.6875 [1]	-11

Note to Table 20–3:

(1) In this case, more bits are needed to represent the integer part of the number.

Figure 20–1. Floating-Point Conversion
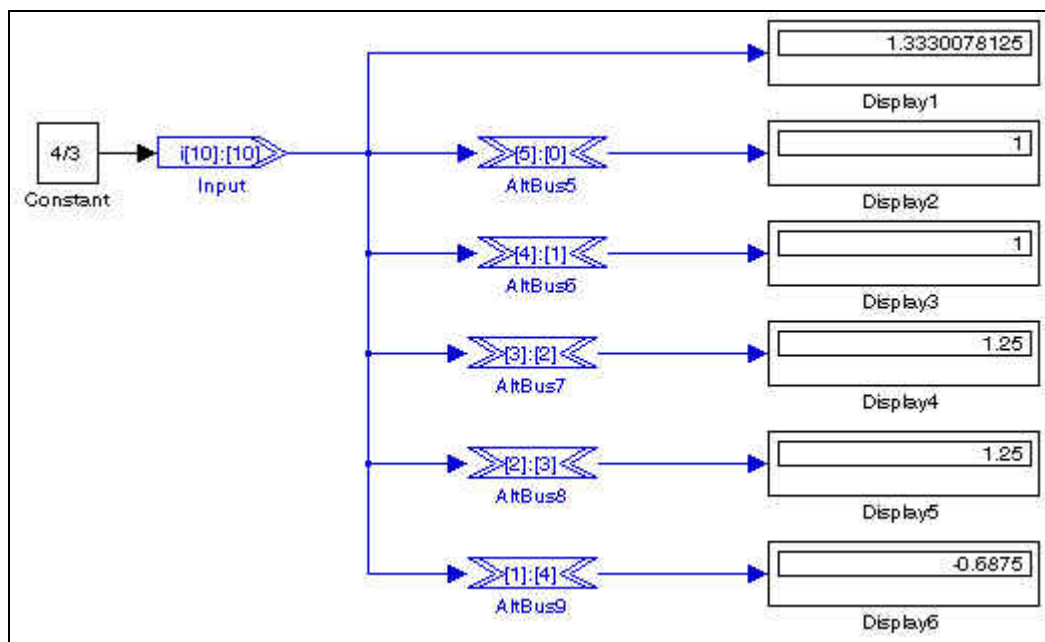
Figure 20–2 illustrates the usage of `AltBus` to convert a 20-bit bus with a ([10].[10]) signed binary fractional format to a 4-bit bus with a [2].[2] signed binary fractional format.

In VHDL, this results in extracting a 4-bit bus (`AltBus(3 DOWNTO 0)`) from a 20-bit bus (`AltBus(19 DOWNTO 0)`) with the assignment:

```
AltBus3(3 DOWNTO 0)) <= AltBus(11 DOWNTO 8))
```

Figure 20–2. Internal Format Conversion
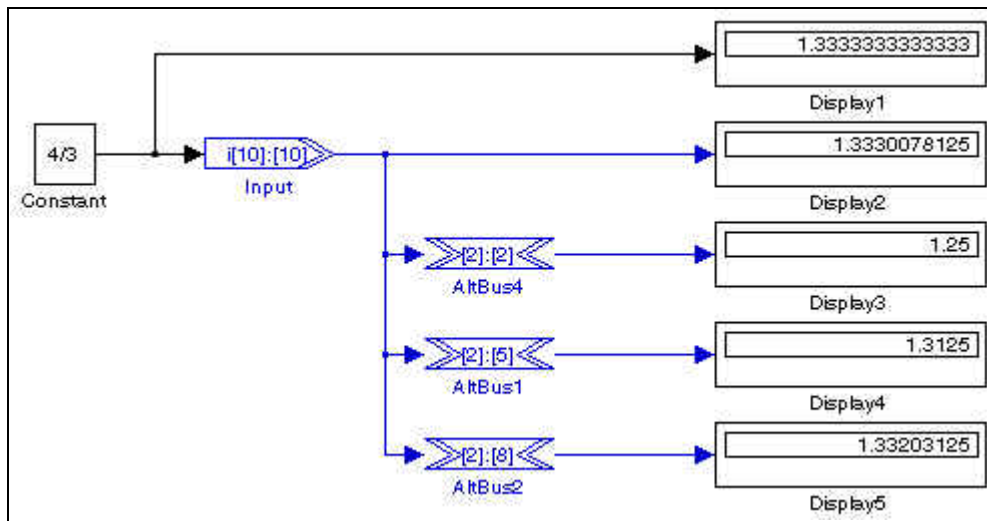
Figure 20–3 shows AltBus blocks for sign extension.

Figure 20–3. Sign Extension
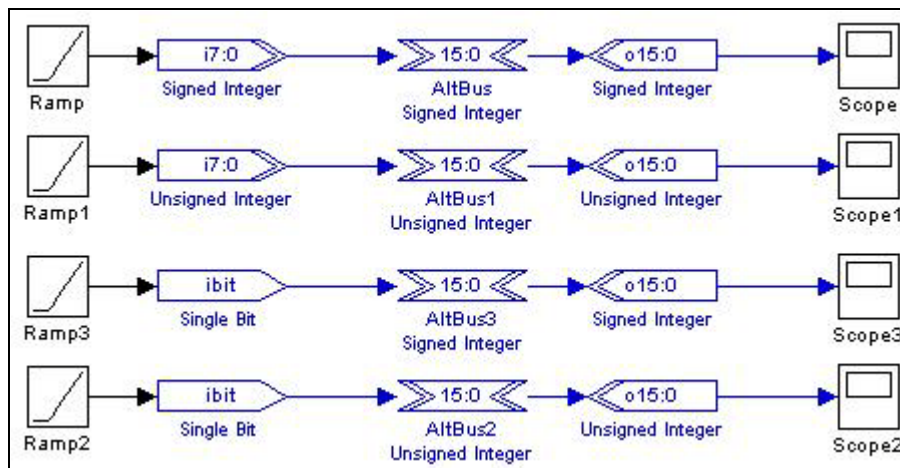
☞ You can also perform additional internal bus manipulation with the Altera Bus Conversion, Extract Bit, or Bus Builder blocks.

Binary Point Casting

The Binary Point Casting block changes the binary point position for a signed fractional bus type, or converts an integer to a fractional bus type.

The output bit width remains equal to the input bit width.

Table 20–4 shows the `Binary Point Casting` block parameters.

Table 20–4. Binary Point Casting Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Output Binary Point Position	>= 0 (Parameterizable)	Specifies the binary point location of the output.

Table 20–5 shows the `Binary Point Casting` block I/O formats.

Table 20–5. Binary Point Casting Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[Li].[Ri]}$	I1: in STD_LOGIC_VECTOR({Li + Ri - 1} DOWNTO 0)	Explicit
O	O1$_{[LO].[RO]}$	O1: out STD_LOGIC_VECTOR({LO + RO - 1} DOWNTO 0)	Explicit

Notes to Table 20–5:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 20–4 shows a design example with the `Binary Point Casting` block.

Figure 20–4. Binary Point Casting Block Example
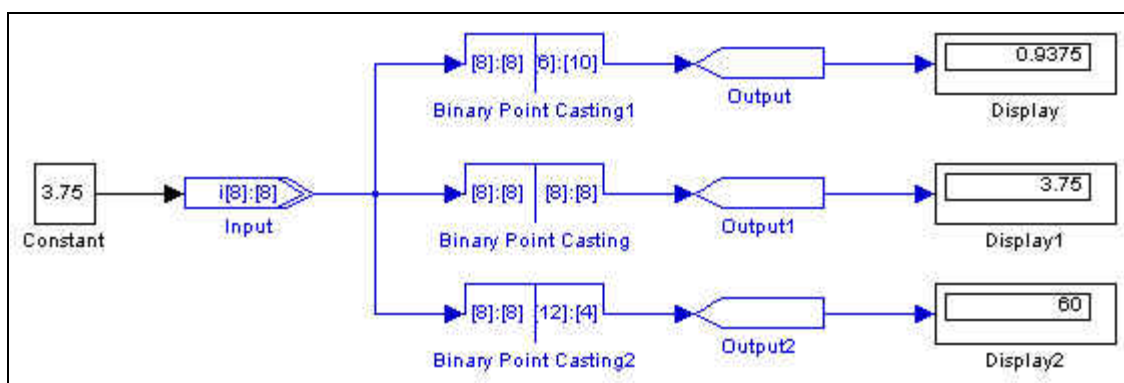
Bus Builder

The `Bus Builder` block constructs an output bus from single-bit inputs. The output bus is signed integer, unsigned integer, or signed binary fractional format. You can specify the number of bits in each case.

The HDL mapping of the `Bus Builder` block is a simple wire.

The input MSB is at the bottom left of the symbol and the input LSB displays at the top left of the symbol.

☞ The `Bus Builder` block does not support sign extension. Instead use a an AltBus block (Figure 20–3 on page 20–4).

Table 20–6 shows the `Bus Builder` block parameters.

Table 20–6. Bus Builder Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed binary fractional buses.

Table 20–7 shows the `Bus Builder` block I/O formats.

Table 20–7. Bus Builder Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[1]}$	I1: in STD_LOGIC	Explicit
	…	…	...
	$Ii_{[1]}$	Ii: in STD_LOGIC	Explicit
	….	….	...
	$In_{[1]}$	In: in STD_LOGIC	Explicit
O	$O1_{[LP].[RP]}$ with LP + RP = n where n is the number of inputs	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 20–7:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 20–5 shows a design example with the `Bus Builder` block.

Figure 20–5. Bus Builder Block Example
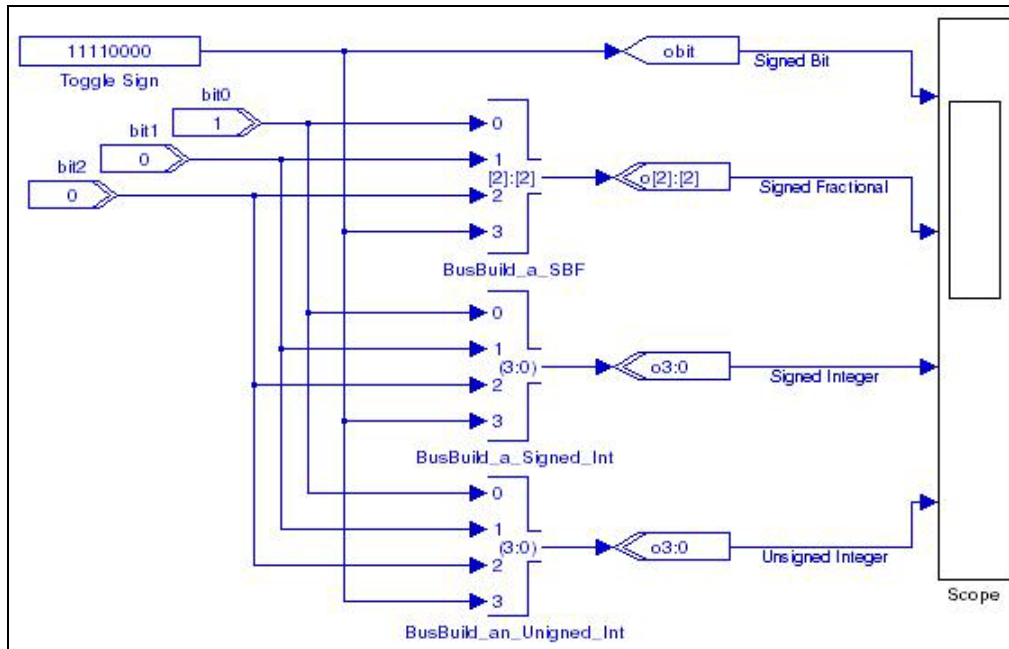
Bus Concatenation

The `Bus Concatenation` block concatenates two buses.

The block has two inputs, a and b. These may be signed integer or unsigned integer. The output width is width(a) + width(b).

Input a becomes the MSB part of the output, input b becomes the LSB part.

Table 20–8 shows the `Bus Concatenation` block parameters.

Table 20–8. Bus Concatenation Block Parameters

Name	Value	Description
Output Is Signed	On or Off	Turn on if the output bus is signed.
Width of Input a	>= 1 (Parameterizable)	Specifies the width of the first bus to concatenate.
Width of Input b	>= 1 (Parameterizable)	Specifies the width of the second bus to concatenate.

Table 20–9 shows the `Bus Concatenation` block I/O formats.

Table 20–9. Bus Concatenation Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[N1]}$ I2$_{[N2]}$	I1: in STD_LOGIC_VECTOR({N1 - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({N2 - 1} DOWNTO 0)	Explicit

Table 20–9. Bus Concatenation Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
O	O1[N1 + N2]	O1: out STD_LOGIC_VECTOR({N1 + N2 - 1} DOWNTO 0)	Explicit

Notes to Table 20–9:

(1) For signed integers, the MSB is the sign bit.

(2) [N] is the number of bits.

(3) I1[N] is an input port. O1[N] is an output port.

(4) Explicit means that the port bit width information is a block parameter.

Figure 20–6 shows an example with the `Bus Concatenation` block.

Figure 20–6. Bus Concatenation Block Example
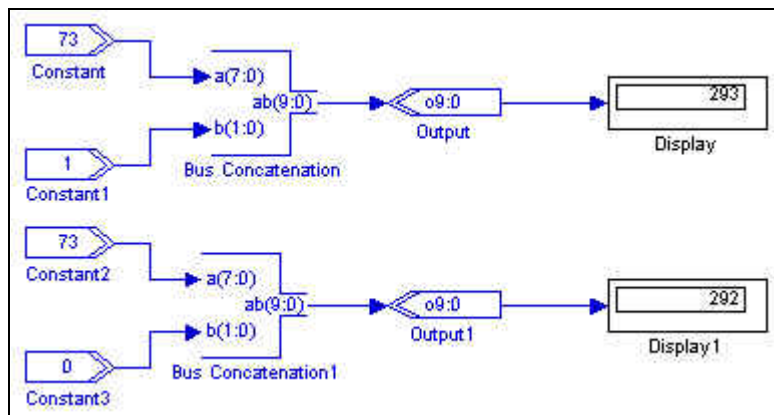
Bus Conversion

The `Bus Conversion` block extracts a subsection of a bus including bus type and width conversion. If the input is in signed binary fractional format, you should specify a left bit width (number of integer bits) and a right bit width (number of fractional bits) for the output bus. If the input is an integer, specify the input bit to connect to the output LSB.

☞ If **Input Bit Connected To Output LSB** is on, the input bit indexing starts from 0. Do not use this option with signed fractional type or with rounding.

Table 20–10 shows the `Bus Conversion` block parameters.

Table 20–10. Bus Conversion Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The input bus type for the simulator, VHDL or both.
Input [number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point including the sign bit.
Input [].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed binary fractional buses.

Table 20–10. Bus Conversion Block Parameters

Name	Value	Description
Output [number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point.
Output [].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bit on the right side of the binary point. This parameter applies only to signed binary fractional buses.
Input Bit Connected to Output LSB	>= 0 (Parameterizable)	Specifies the slice of the input bus to use. This parameter designates the start point of the slice that is transferred to the output LSB and applies to signed or unsigned integer buses only.
Round	On or Off	Turn on to round the output away from zero. When this option is off, the LSM is truncated: <int>(input +0.5).
Saturate	On or Off	When this option is on, if the output is greater than the maximum positive or negative value to be represented, the output is forced (or saturated) to the maximum positive or negative value, respectively. If off, the MSB is truncated.

Table 20–11 shows the `Bus Conversion` block I/O formats.

Table 20–11. Bus Conversion Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[LPi].[RPi]}$	I1: in STD_LOGIC_VECTOR({LPi + RPi - 1} DOWNTO 0)	Explicit
O	O1$_{[LPO].[RPO]}$	O1: out STD_LOGIC_VECTOR({LPO + LPO - 1} DOWNTO 0)	Explicit

Notes to Table 20–11:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.
(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.
(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 20–7 shows a design example with the `Bus Conversion` block.

Figure 20–7. Bus Conversion Block Example
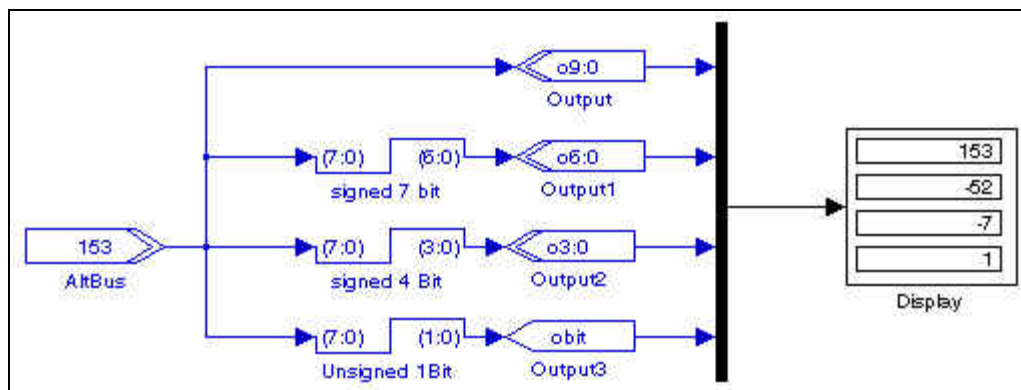
Bus Splitter

The `Bus Splitter` block splits a bus into single-bit outputs.

The output ports are numbered from LSB to MSB. You can specify the bus type that you want to use, and specify the number of bits on either side of the binary point.

Table 20–12 shows the `Bus Splitter` block parameters.

Table 20–12. Bus Splitter Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed binary fractional buses.

Table 20–13 shows the `Bus Splitter` block I/O formats.

Table 20–13. Bus Splitter Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[LP].[RP]}$ with LP + RP = n where n is the number of inputs	I1: in STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit
O	$O1_{[1]}$ … $On_{[1]}$	O1: in STD_LOGIC … On: in STD_LOGIC	Explicit ... Explicit

Notes to Table 20–7:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 20–8 shows a design example with the `Bus Splitter` block.

Figure 20–8. Bus Splitter Block Example
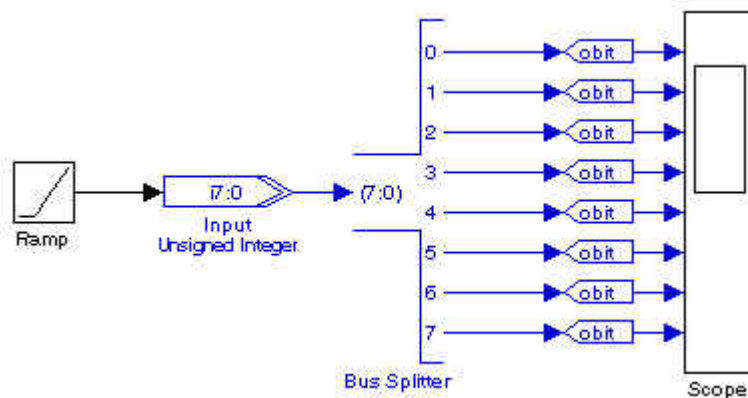
Constant

The `Constant` block specifies a constant bus. The options available depend on the selected bus type.

Table 20–14 shows the `Constant` block parameters.

Table 20–14. Constant Block Parameters

Name	Value	Description
Constant Value	Double (Parameterizable)	Specifies the constant value that is formatted with the specified bus type.
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, Single Bit	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Rounding Mode	Truncate, Round Towards Zero, Round Away From Zero, Round To Plus Infinity, Convergent Rounding	The rounding mode. Refer to the description of the Round block for more information about the rounding modes.
Saturation Mode	Wrap, Saturate	The saturation mode.
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined (Parameterizable)	Specifies the name of the required clock signal.

Table 20–15 shows the `Constant` block I/O formats.

Table 20–15. Constant Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
O	O1[LP].[RP]	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 20–15:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 20–9 shows an example with the Constant block.

Figure 20–9. Constant Block Example
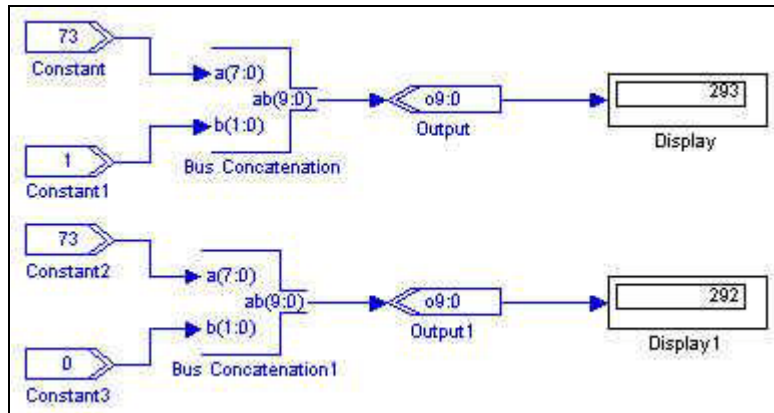
Extract Bit

The Extract Bit block reads a Simulink bus in the specified format and outputs the single bit specified.

The selected bit is indexed starting from zero for the LSB and increasing to (total bit width - 1) for the MSB.

Table 20–16 shows the Extract Bit block parameters.

Table 20–16. Extract Bit Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Specifies the number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point.
Select the Bit to be Extracted From the Bus	>= 0 (Parameterizable)	Specifies the input bit to extract.

Table 20–17 shows the `Extract Bit` block I/O formats.

Table 20–17. Extract Bit Block I/O Formats *(1)*

I/O	Simulink *(2)*, *(3)*	VHDL	Type *(4)*
I	I1$_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
O	O1$_{[1]}$	O1: out STD_LOGIC	Explicit

Notes to Table 20–17:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 20–10 shows a design example with the `Extract Bit` block.

Figure 20–10. Extract Bit Block Example
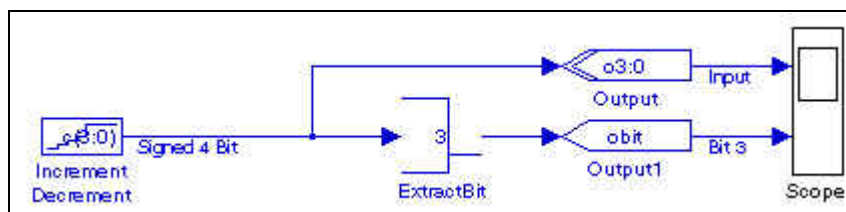
Global Reset

The `Global Reset` (or SCLR) block provides a single bit reset signal. All signals driven by the block are connected to the global reset for that clock domain. In simulation, this block outputs a constant 0.

Table 20–18 shows the `Global Reset` block parameters.

Table 20–18. Global Reset Block Parameters

Name	Value	Description
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined (Parameterizable)	Specifies the name of the required clock signal.

Table 20–19 shows the `Global Reset` block I/O formats.

Table 20–19. Global Reset Block I/O Formats *(1)*

I/O	Simulink *(2)*, *(3)*	VHDL	Type *(4)*
O	O1$_{[1].[0]}$	O1: out STD_LOGIC	Explicit

Notes to Table 20–19:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

GND

The GND block is a single bit that outputs a constant 0. Table 20–20 shows the GND block parameters.

Table 20–20. GND Block Parameters

Name	Value	Description
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined (Parameterizable)	Specifies the name of the required clock signal.

Table 20–21 shows the GND block I/O formats.

Table 20–21. GND Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
O	O1$_{[1].[0]}$	O1: out STD_LOGIC	Explicit

Notes to Table 20–21:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 20–11 shows a design example with the GND block.

Figure 20–11. GND Block Example
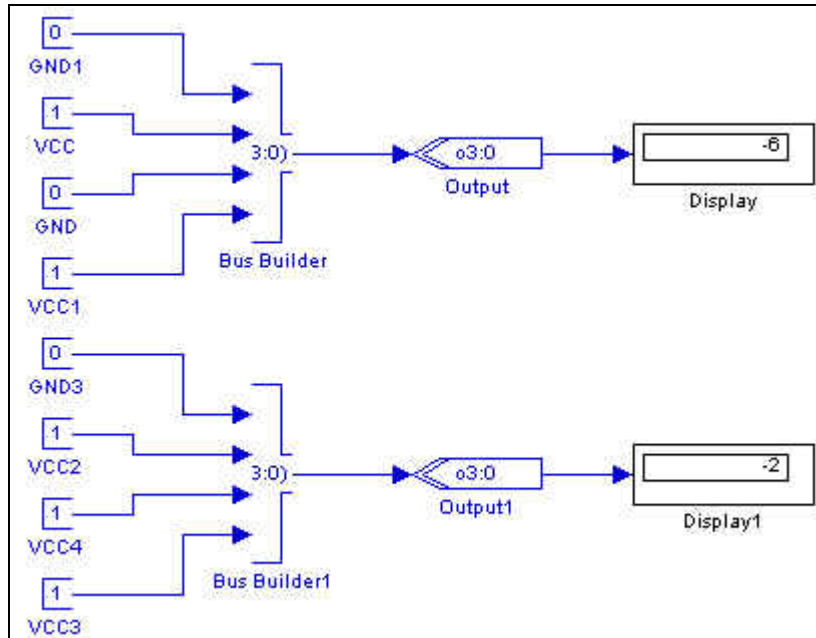
Input

The `Input` block defines the input boundary of a hardware system and casts floating-point Simulink signals (from generic Simulink blocks) to signed binary fractional format (feeding DSP Builder blocks).

Table 20–22 shows the `Input` block parameters.

Table 20–22. Input Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, Single Bit	Specifies the number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined (Parameterizable)	Specifies the name of the required clock signal.

Table 20–23 on page 20–15 shows the `Input` block I/O formats.

Table 20–23. Input Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L1].[R1]}$	I1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit - Optional
O	O1$_{[LP].[RP]}$	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 20–23:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.
(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.
(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Non-synthesizable Input

The `Non-synthesizable Input` block marks an entry point to a non-synthesizable DSP Builder system. Use a corresponding Non-synthesizable Output block to mark the exit point. Because DSP Builder registers its own type with Simulink, this block is required when the DSP Builder blocks are not intended to be synthesized.

Table 20–24 shows the `Non-synthesizable Input` block parameters.

Table 20–24. Non-synthesizable Input Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, Single Bit	Specifies the number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined (Parameterizable)	Specifies the name of the required clock signal.

Table 20–25 shows the `Non-synthesizable Input` block I/O formats.

Table 20–25. Non-synthesizable Input Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L1].[R1]}$	I1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit - Optional
O	O1$_{[LP].[RP]}$	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 20–23:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Non-synthesizable Output

The `Non-synthesizable Output` block marks an exit point from a non-synthesizable DSP Builder system. Use a corresponding Non-synthesizable Input block to mark the entry point. Because DSP Builder registers its own type with Simulink, this block is required when the DSP Builder blocks are not intended to be synthesized. You can also use this block to create an non-synthesizable output from a synthesizable system.

You can optionally specify the external Simulink type. If set to `Simulink Fixed Point Type`, the bit width is the same as the DSP Builder input type. If set to `Double`, the width may be truncated if the bit width is greater than 52.

Table 20–26 shows the `Non-synthesizable Output` block parameters.

Table 20–26. Non-synthesizable Output Block Parameters

Name	Value	Description
Bus Type	Inferred, Signed Integer, Unsigned Integer, Signed Fractional, Single Bit	Specifies the number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
External Type	Inferred, Simulink Fixed Point Type, Double	Specifies whether the external type is inferred from the Simulink block it is connected to or explicitly set to either Simulink Fixed Point or Double type. The default is Inferred.

Table 20–27 shows the `Non-synthesizable Output` block I/O formats.

Table 20–27. Non-synthesizable Output Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$	I1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit - Optional
O	$O1_{[LP].[RP]}$	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 20–29:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Output

The `Output` block defines the output boundary of a hardware system and casts signed binary fractional format (from DSP Builder blocks) to floating-point Simulink signals (feeding generic Simulink blocks).

`Output` blocks map to output ports in VHDL and mark the edge of the generated system. You normally connect these blocks to Simulink simulation blocks in your testbench. Their outputs should not be connected to other Altera blocks.

You can optionally specify the external Simulink type. If set to `Simulink Fixed Point Type`, the bit width is the same as the input. If set to `Double`, the width may be truncated if the bit width is greater than 52.

Table 20–28 shows the Output block parameters.

Table 20–28. Output Block Parameters

Name	Value	Description
Bus Type	Inferred, Signed Integer, Unsigned Integer, Signed Fractional, Single Bit	The number format of the bus.
[number of bits].[]	>= 0 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
External Type	Inferred, Simulink Fixed Point Type, Double	Specifies whether the external type is inferred from the Simulink block it is connected to or explicitly set to either Simulink Fixed Point or Double type. The default is Inferred.

Table 20–29 shows the Output block I/O formats.

Table 20–29. Output Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L1].[R1]}$	I1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit - Optional
O	O1$_{[LP].[RP]}$	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 20–29:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Round

The Round block rounds the input to the closest possible representation in the specified output bus format. If the nearest two possibilities are equidistant, you can specify from the available rounding modes:

- **Truncate**: Remove discarded bits without changing the other bits; effectively, specify the lower value. This is the simplest and fastest mode to implement in hardware.

- **Round Towards Zero**: Specify the value closer to zero.

- **Round Away From Zero**: Specify the value further from zero (round downwards for negative values, upwards for positive values). This was the rounding behavior in DSP Builder version 7.0 and before. When using this mode —the maximum positive value overflows the available representation. For example, when rounding from an 8-bit signed input to a 6-bit signed output, 01111111 (127) becomes 100000 (-32). If you use this mode, it is best to use saturation logic to prevent this from happening.

- **Round To Plus Infinity**: Specify the higher value.

■ **Convergent Rounding**: Specify the even value. For a large sample of random input values there is no bias —on average the same number of values round upwards as downwards.

☞ When using Simulink fixed-point types, MATLAB supports the following rounding options: **Zero**, **Nearest** (equivalent to **Round Away From Zero**), **Ceiling**, **Floor** (equivalent to **Truncate**), and **Simplest**. The MATLAB **Zero** and **Ceiling** modes round all intermediate values up or down and have no DSP Builder equivalent. This is because the DSP Builder modes (except **Truncate**) always specify the nearest representable value and the rounding mode applies only to values that are equidistant from two representable values. For example, 0.9 rounds to 1 (for all modes except **Truncate**) but the MATLAB **Zero** mode rounds 0.9 to 0. Similarly 0.1 rounds to 0 but the MATLAB **Ceiling** mode rounds 0.1 to 1.

Table 20–30 shows the `Round` block parameters.

Table 20–30. Round Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format of the bus.
[number of bits].[]	>= 2 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Number of LSB Bits to Remove	>= 0 (Parameterizable)	Specifies how many bits to remove.
Rounding Mode	Truncate, Round Towards Zero, Round Away From Zero, Round To Plus Infinity, Convergent Rounding	The rounding mode.
Enable Pipeline	On or Off	Turn on if you want to pipeline the function.
Use Enable Port [1]	On or Off	Turn on to use the clock enable input (`ena`).
Use Asynchronous Clear Port [1]	On or Off	Turn on to use the asynchronous clear input (`aclr`).

Note to Table 20–30:

(1) These ports are available only when you enable pipeline.

Table 20–31 shows the `Round` block I/O formats.

Table 20–31. Round Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L1].[R1]}$ I2$_{[1]}$ I3$_{[1]}$	I1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC	Explicit

Table 20–31. Round Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
O	O1[LP].[RP]	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to **Table 20–31**:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 20–12 shows a design example with the `Round` block.

Figure 20–12. Round Block Example
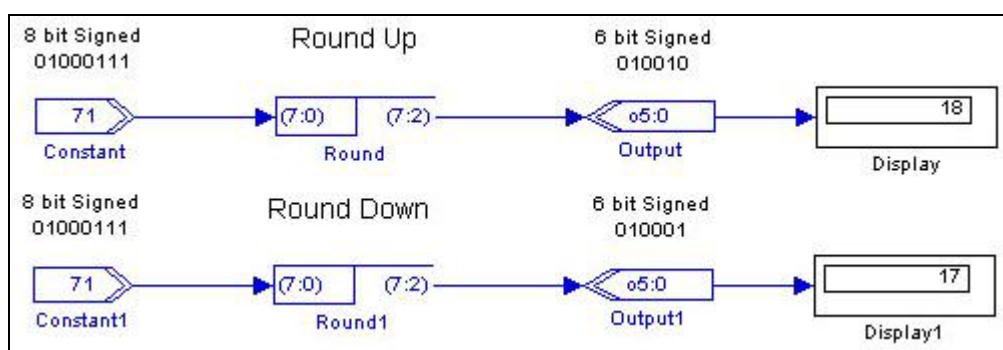
Saturate

The `Saturate` block limits output to a maximum value. If the output is greater than the maximum positive or negative value to be represented, the output is forced (or saturated) to the maximum positive or negative value, respectively. Alternatively, you can truncate the MSB.

Table 20–32 shows the `Saturate` block parameters.

Table 20–32. Saturate Block Parameters

Name	Value	Description
Input Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The number format of the bus.
[number of bits].[]	>= 2 (Parameterizable)	Specifies the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
[].[number of bits]	>= 0 (Parameterizable)	Specifies the number of bits to the right of the binary point. This parameter applies only to signed fractional buses.
Number of MSB Bits to Remove	>= 0 (Parameterizable)	Specifies how many bits to remove.
Saturation Type	Saturate, Truncate MSB, Enter Saturation Limits	Saturate, truncate, or specify the saturation limits for the output.

Table 20–32. Saturate Block Parameters

Name	Value	Description
Upper Saturation Limit	Integer (Parameterizable)	Specifies the upper saturation limit when Saturation Type is set to Enter Saturation Limits.
Lower Saturation Limit	Integer (Parameterizable)	Specifies the lower saturation limit when Saturation Type is set to Enter Saturation Limits.
Enable Pipeline	On or Off	Turn on if you want to pipeline the function.
Use Saturation Occurred Port	On or Off	Turn on to use the saturation occurred input (`sat_flag`).
Use Enable Port [1]	On or Off	Turn on to use the clock enable input (`ena`).
Use Asynchronous Clear Port [1]	On or Off	Turn on to use the asynchronous clear input (`aclr`).

Note to Table 20–30:

(1) These ports are available only when you enable pipeline.

Table 20–33 shows the `Saturate` block I/O formats.

Table 20–33. Saturate Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$ $I2_{[1]}$ $I3_{[1]}$ $I4_{[1]}$	I1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC I4: in STD_LOGIC	Explicit
O	$O1_{[LP].[RP]}$	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 20–33:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.
(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.
(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 20–13 shows a design example with the `Saturate` block.

Figure 20–13. Saturate Block Example
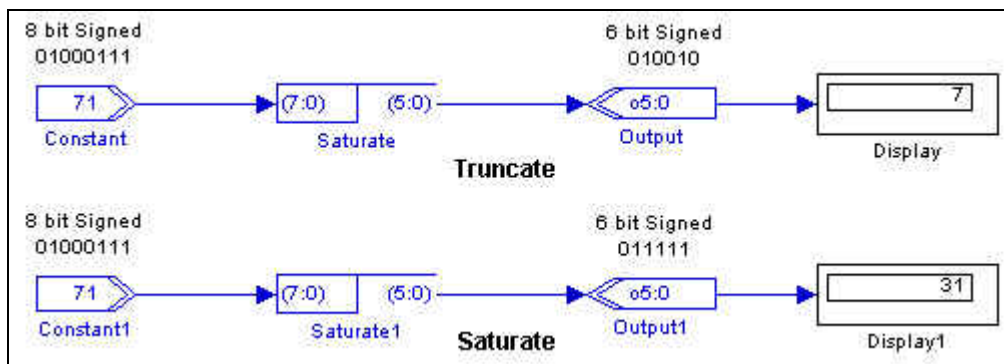
VCC

The VCC block outputs a single-bit constant 1.

Table 20–34 shows the VCC block parameters.

Table 20–34. VCC Block Parameters

Name	Value	Description
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock	User defined (Parameterizable)	Specifies the name of the required clock signal.

Table 20–35 shows the VCC block I/O formats.

Table 20–35. VCC Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
O	O1[1]	O1: out STD_LOGIC	Explicit

Notes to Table 20–35:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 20–14 shows a design example with the VCC block.

Figure 20–14. VCC Block Example

21. MegaCore Functions Library
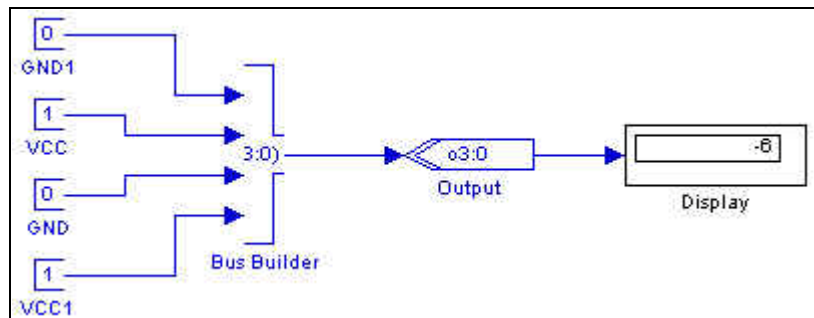
The MegaCore Functions library contains blocks that represent parameterizable IP cores, which install with the Quartus Prime software. The MegaCore Functions library blocks do not support designs targeting Arria 10 devices.

For Arria 10 designs, use the **MegaCore Import** block.

DSP Builder supports the following Altera DSP IP:

- CIC—implements a cascaded integrator-comb) filter.

  For more information, refer to the *CIC MegaCore Function User Guide*.

- FFT—implements a high performance fast Fourier transform or inverse FFT processor.

  For more information, refer to the *FFT MegaCore Function User Guide*.

- NCO—implements a customized numerically controlled oscillator.

  For more information, refer to the *NCO MegaCore Function User Guide*.

- Reed-Solomon—implements a forward error correction encoder or decoder.

  For more information, refer to the *Reed-Solomon Compiler User Guide*.

- Viterbi—Implements a high performance Viterbi decoder.

  For more information, refer to the *Viterbi Compiler User Guide*.

The Rate Change library contains the following blocks that allow you to control the clock assignment to registered DSP Builder blocks, such as `Delay` or `Increment Decrement` blocks:

■ Clock

■ Clock_Derived

■ Dual-Clock FIFO

■ Multi-Rate DFF

■ PLL

■ Tsamp

For information about the Clock and Clock_Derived blocks, refer to Chapter 14, AltLab Library. For information about the Dual-Clock FIFO block, refer to Chapter 25, Storage Library.

Multi-Rate DFF

The `Multi-Rate DFF` block implements a D-type flipflop and typically specifies sample rate transitions.

Simulation of the `Multi-Rate DFF` block may not match hardware because of limitations in the way DSP Builder simulates multiclock designs. Typically, differences may occur when moving from a slow to a fast clock domain. In such cases, an error message of the following form issues in the MATLAB command window:

```
Warning: simulation will not match hardware
```

If your design allows, increasing the latency of the `Multi-Rate DFF` block to at least one slow clock period should result in correct simulation results.

If the clocks are asynchronous, simulations do not match hardware. Do not use a `Multi-Rate DFF` block to cross asynchronous clock domains, otherwise data is corrupted or lost. Use a Dual-Clock FIFO block instead to guarantee correct data transfer.

Table 22–1 shows the `Multi-Rate DFF` block inputs and outputs.

Table 22–1. Multirate DFF Block Inputs and Outputs

Signal	Direction	Description
d	Input	Input data port.
q	Output	Output data port.
ena	Input	Optional clock enable port.
sclr	Input	Optional synchronous clear port.

Table 22–2 shows the `Multi-Rate DFF` block parameters.

Table 22–2. Multi-Rate DFF Block Parameters

Name	Value	Description
Number of Pipeline Stages	>= 1 (Parameterizable)	Adds more pipeline stages to the block. Increased delay reduces the likelihood of metastability.
Use Base Clock	On or Off	Turn on to use the base clock.
Clock Name	User specified	Specify the name of the clock signal.
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear input (`sclr`).

Table 22–3 shows the `Multi-Rate DFF` block I/O formats.

Table 22–3. Multi-Rate DFF Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L].[R]}$ $I2_{[1]}$ $I3_{[1]}$	I1: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC	Implicit
O	$O1_{[L].[R]}$	O1: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Implicit

Notes to Table 22–3:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 22–1 shows an design example with the `Multi-Rate DFF` block.

Figure 22–1. Multirate DFF Block Example
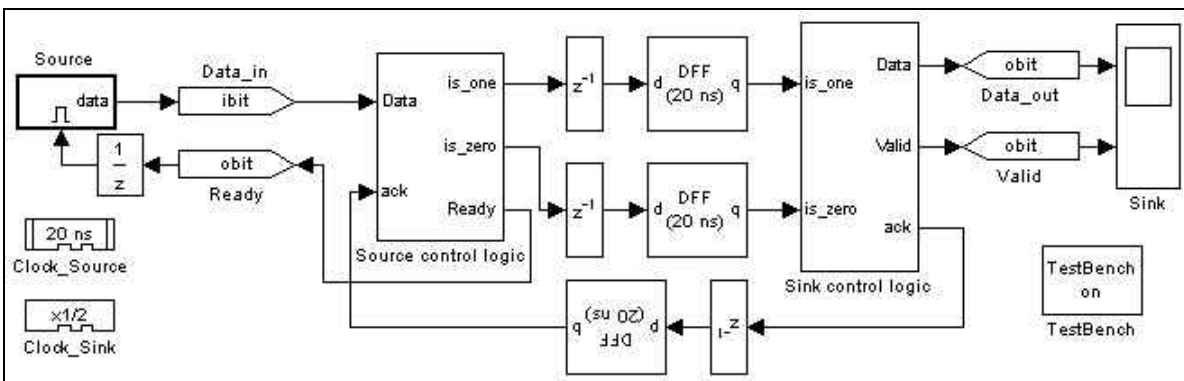
PLL

The `PLL` block generates a clock signal that is based on a reference clock.

Phase-locked loops (PLL) have become an important building block of most high-speed digital systems today. Their use ranges from improving timing as zero delay lines to full-system clock synthesis. The Arria, Cyclone, and Stratix series device families offer advanced on-chip PLL features that were previously offered only by the most complex discrete devices.

Each PLL has multiple outputs that can source any of the 40 system clocks in the devices to give you complete control over your clocking needs. The PLLs offer full frequency synthesis capability (the ability to multiply up or divide down the clock period) and phase shifting for optimizing I/O timing. Additionally, the PLLs have high-end features such as programmable bandwidth, spread spectrum, and clock switchover.

The PLL block generates internal clocks with frequencies that are multiples of the frequency of the system clock. PLLs on the FPGA can simultaneously multiply and divide the reference clock. The PLL block checks the validity of the parameters.

☞ If you use a PLL block to define clock signals when there is no Clock block in your design, the PLL-derived clocks might not pass the derived period correctly to the blocks referencing the PLL-derived clock. Always explicitly include a Clock block with a PLL block.

The number of PLL internal clock outputs supported by each device family depends on the specific device packaging.

👣 For information about the built-in PLLs, refer to the device handbook for the device family you target.

The following restrictions apply when you use a PLL block:

■ Your design may contain more than one PLL block but they must be at the top level.

■ Each output clock of the PLL has a zero degree phase shift and 50% duty cycle.

Table 22–4 shows the PLL block parameters.

Table 22–4. PLL Block Parameters

Name	Value	Description
Input Clock:	User specified	Specify the name of the input clock signal.
Use Base Clock	On or Off	Turn on to use the base clock.
Number of Output Clocks	1–9	The number of PLL clock outputs.
Output Clocks	*<PLL block name>*_clk0 to *<PLL block name>*_clk8	Select the PLL clock that you want to set frequency multiplier and divider factors for.
Period Multiplier	*(1)*	Multiply the reference clock period by this value.
Period Divider	*(1)*	Divide the reference clock period by this value.
Export As Output Pin	On or Off	Turn on to export this clock as an output pin.

Note to Table 22–4:

(1) Refer to the device documentation for the device family you target.

Tsamp

The `Tsamp` block sets the clock domain inherited by all downstream blocks.

☞ When you use the `Tsamp` block, you must select a variable step solver in the Simulink configuration parameters. Unless the downstream clock is an exact, slower multiple of the upstream clock, the simulation results may not match ModelSim; in this case it is better to use a Multi-Rate DFF block.

Table 22–5 shows the `Tsamp` block inputs and outputs.

Table 22–5. Tsamp Block Inputs and Outputs

Signal	Direction	Description
<unnamed>	Input	Input data port.
<unnamed>	Output	Output data port.

Table 22–6 shows the `Tsamp` block parameters.

Table 22–6. Tsamp Block Parameters

Name	Value	Description
Specify Clock	On or Off	Turn on to explicitly specify the clock name.
Clock Name	User specified	Specify the name of the Clock block that specifies the clock signal.

Table 22–7 shows the `Tsamp` block I/O formats.

Table 22–7. Tsamp Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L].[R]}$	I1: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Implicit
O	O1$_{[L].[R]}$	O1: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Implicit

Notes to Table 22–7:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 22–2 on page 22–5 shows an design example with the `Tsamp` block.

Figure 22–2. Tsamp Block Example
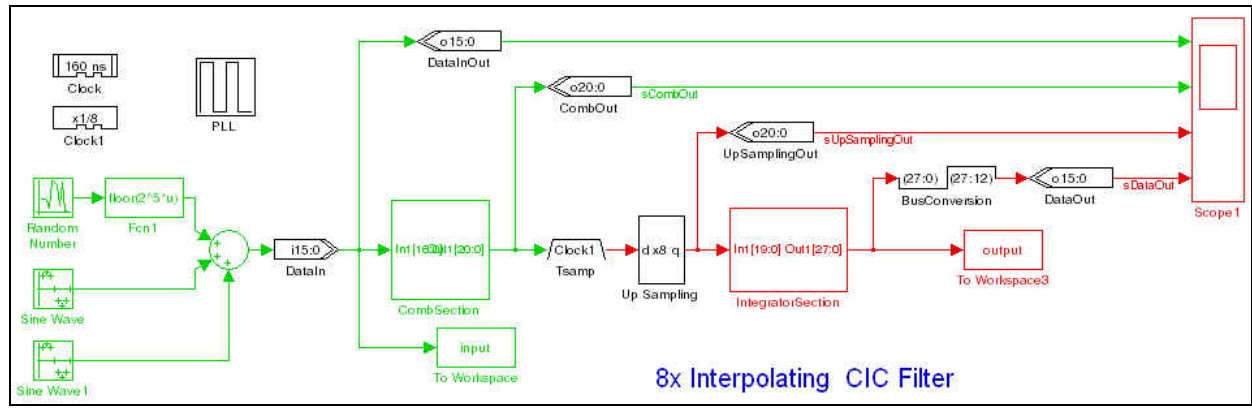
This design example is available in the *<DSP Builder install path>***DesignExamples\Demos\Filters\Filters\CicFilter** directory.

The Simulation library contains the following simulation-only blocks that do not synthesize to HDL when Signal Compiler runs:

■ External RAM

■ Multiple Port External RAM

External RAM

The `External RAM` block is a simulation model of an external RAM. The `External RAM` block stores and retrieves data from a range of addresses and is compatible with the Avalon-MM interface.

👣 For information about the Avalon-MM interface, refer to *Avalon Interface Specifications*.

This block is not cycle-accurate and a warning issues if you use it in a gate level (cycle-accurate) simulation.

☞ If 64 or 128 bit data width is specified, the block attempts to use a Simulink fixed-point license. If you do not have a Simulink fixed-point license., you can only use 8, 16 or 32 bit data widths.

👣 For information about fixed-point licenses, refer to the Simulink Help.

This is a simulation only block, and does not generate any HDL when Signal Compiler is run.

Table 23–1 shows the `External RAM` block inputs and outputs.

Table 23–1. External RAM Block Inputs and Outputs

Signal	Direction	Description
WriteData	Input	Data lines for write transfers. Not required if there are no write transfers. If used, also use `Write`.
WriteAddress	Input	Address lines for write transfers.
ReadAddress	Input	Address lines for read transfers.
Read	Input	Read request signal. Not required if there are no read transfers. If used, also use `ReadData`.
Write	Input	Write request signal. Not required if there are no write transfers. If used, also use `WriteData`.
ReadData	Output	Data lines for read transfers. Not required if there are no read transfers. If used, also use `Read`.
WriteWaitRequest	Output	Stalls the interface when the Avalon-MM interface cannot respond immediately to a write request.

Table 23–1. External RAM Block Inputs and Outputs

Signal	Direction	Description
ReadWaitRequest	Output	Stalls the interface when the Avalon-MM interface cannot respond immediately to a read request.
ReadDataValid	Output	Marks the rising clock edge when ReadData is asserted. Indicates that valid data is present on the ReadData lines.

Table 23–2 shows the External RAM block parameters.

Table 23–2. External RAM Block Parameters

Name	Value	Description
Data Width	8, 16, 32, 64, or 128	Specifies the number of bits for the data. No other values are supported. 64 and 128 bit data widths require a Simulink fixed-point license.
Address Width	1–32	Specifies the number of bits n for the address.
Wait States Per Write	0–10	Specifies a fixed number of wait states for each write transfer.
Maximum Latency	1–255	Specifies the latency for pipelined read transfers.
Size	$1-2^n$ [1]	Specifies the total size of the RAM in bytes (the number of addresses when you use a range of addresses).
Offset	$1-2^n$ [1]	Specifies an offset for the RAM start address (the start address when you use a range of addresses.

Notes to Table 23–2

(1)   The size added to the offset must be less than 2^n where n is the address width.

Figure 23–1 shows an design example with the External RAM block.

Figure 23–1. External RAM Block Example
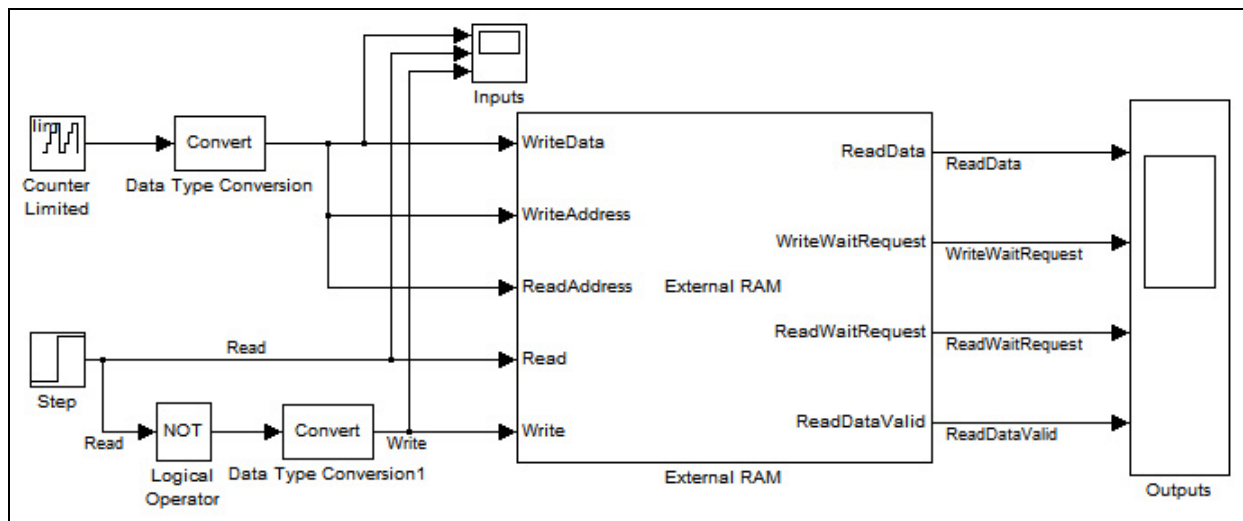
Multiple Port External RAM

The Multiple Port External RAM block is a simulation model of a multiple port external RAM block. It stores and retrieves data from a range of addresses and is compatible with the Avalon-MM interface.

For information about the Avalon-MM interface, refer to *Avalon Interface Specifications*.

This block is not cycle-accurate and a warning issues if you use it in a gate level (cycle-accurate) simulation.

☞ If 64 or 128 bit data width is specified, the block attempts to use a Simulink fixed-point license. If you do not have a Simulink fixed-point license., you can only use 8, 16 or 32 bit data widths.

For information about fixed-point licenses, refer to the Simulink Help.

This block is simulation only and does not generate any HDL when you run Signal Compiler.

The ports on the block symbol update when you change the number of write or read interfaces. However, the port names do not automatically show on the block symbol. To display the updated block symbol correctly, perform the following steps:

1. Click on the block, point to **Link Options** in the popup menu and click **Break Link**.

2. While the block is still selected, run the following command in MATLAB:

   `alt_dspbuilder_update_external_RAM`

   Table 23–3 shows the `Multiple Port External RAM` block inputs and outputs.

Table 23–3. Multiple Port External RAM Block Inputs and Outputs

Signal	Direction	Description
WriteDataN	Input	Data lines for write transfers on port *N*.
WriteAddressN	Input	Address lines for write transfers on port *N*.
WriteEnableN	Input	Write enable for transfers on port *N*.
WriteBurstCountN	Input	Write burst count for transfers on port *N*.
ReadAddressN	Input	Address lines for read transfers on port *N*.
ReadEnableN	Input	Read enable for transfers on port *N*.
ReadBurstCountN	Input	Read burst count for transfers on port *N*.
WriteWaitRequestN	Output	Stalls the interface when the Avalon-MM interface cannot respond immediately to a write request on port *N*.
ReadDataN	Output	Data lines for read transfers on port *N*.
ReadDataValidN	Output	Marks the rising clock edge when `ReadDataN` is asserted. Indicates that valid data is present on the `ReadDataN` lines.
ReadWaitRequestN	Output	Stalls the interface when the Avalon-MM interface cannot respond immediately to a read request on port *N*.

Table 23–4 shows the `Multiple Port External RAM` block parameters.

Table 23–4. Multiple Port External RAM Block Parameters

Name	Value	Description
Number of Write Interfaces	0–5	Specifies the number of write ports.
Number of Read Interfaces	0–5	Specifies the number of read ports.

Table 23–4. Multiple Port External RAM Block Parameters

Name	Value	Description
Data Width	8, 16, 32, 64, or 128	Specifies the number of bits for the data. No other values are supported. 64 and 128 bit data widths require a Simulink fixed-point license.
Address Width	1–32	Specifies the number of bits n for the address.
Wait States Per Write	0–10	Specifies a fixed number of wait states for each write transfer.
Maximum Latency	1–255	Specifies the latency for pipelined read transfers.
Size	$1–2^n$ [1]	Specifies the total size of the RAM in bytes (the number of addresses when you use a range of addresses).
Offset	$1–2^n$ [1]	Specifies an offset for the RAM start address (the start address when you use a range of addresses.

Notes to Table 23–4

(1)   The size added to the offset must be less than 2^n where n is the address width.

The State Machine Functions library contains the following block:

■ State Machine Editor

■

State Machine Editor

The `State Machine Editor` block provides access to the Quartus® II state machine editor, which allows you to create graphic representations of state machines for use in your design.

A state machine is a very efficient means to specify complex control logic that you can then use to generate a HDL description and Simulink interface to the simulation model.

You can define a state machine graphically by adding states and transitions directly on the diagram, or by using a wizard interface to enter all the properties for the state machine. When you use the wizard interface, a graphical state diagram view is created with the states and transitions automatically placed for optimum readability.

Figure 24–1 shows the state machine that is created when you use the default options in the wizard.

Figure 24–1. Default State Machine Diagram View
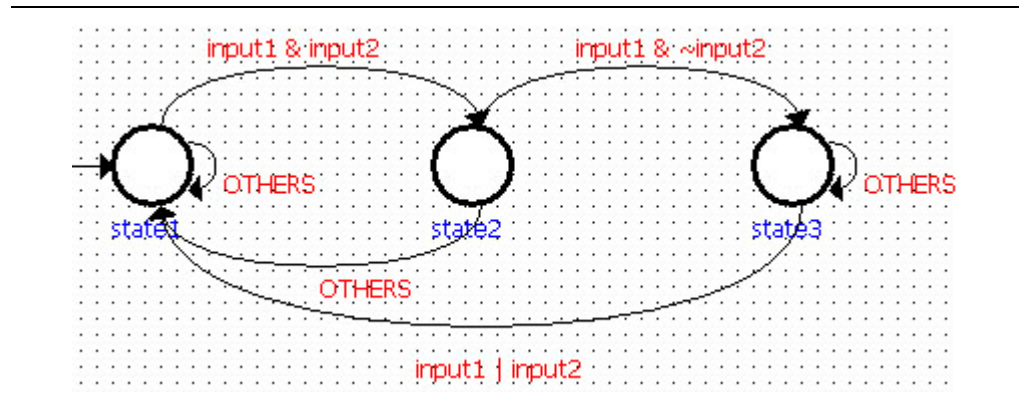
Table 24–1 shows the parameters that you can e set in the State Machine wizard.

Table 24–1. State Machine Wizard Parameters

Name	Value	Description
Which reset mode do you want to use	Synchronous, Asynchronous	Specifies whether the state machine has a synchronous or asynchronous reset.
Reset is active-high	On, Off	Turn on to uses an active-high reset or off if you want an active-low reset.
Register the output ports	On, Off	Turn on to register the state machine output ports.

Table 24–1. State Machine Wizard Parameters

Name	Value	Description
States	user specified	You can specify any number of state names that must be valid HDL identifiers.
Input ports	user specified	You can specify any number of input port names that must be valid HDL identifiers.
State transitions	user specified	You can specify any number of conditional statements for the transitions between source and destination states.
Transition to source state if not specified	On, Off	Turn on to always transition to the source state if not all transition conditions are specified.
Output ports	user specified	You can specify any number of output port names that must be valid HDL identifiers.
Action conditions	user specified	You can specify actions assigned to each output port.

Use Verilog HDL syntax to specify the conditional statements that you specify for state transitions and output actions. Table 24–2 shows the operators you can use to define a conditional expression.

Table 24–2. State Machine Editor Operators

Operator	Description	Priority	Example
~ (unary)	Negative	1	~in1
(...)	Brackets	1	(1)
==	Numeric equality	2	in1==5
!=	Not equal to	2	in1!=5
>	Greater than	2	in1>in2
>=	Greater than or equal to	2	in1>=in2
<	Less than	2	in1<in2
<=	Less than or equal to	2	in1<=in2
&	AND	2	(in1==in2)&(in3>=4)
|	OR	2	(in1==in2)|(in1>in2)

A conditional statement consists of a source state, a condition that causes a transition to take place, and the destination state to which the state machine transitions. The source state and destination state values must be valid state names, which you can select from a drop down list in the wizard.

The state machine description is saved in a *<block name>*.**smf** file when you close the state machine wizard.

The syntax of each conditional statement is automatically checked on entry and the completed state machine is validated when you generate HDL to ensure that the state machine is functionally correct.

For more information including procedures for drawing a graphical state machine, refer to the *About the State Machine Editor* topic in the Quartus Prime Help.

When you exit from the State Machine Editor, the generated HDL is compiled in the Quartus Prime software and the ports updated on the block in your Simulink model.

Figure 24–2 shows an example of the default state machine that the State Machine Editor wizard creates and includes in a simple Simulink model.

Figure 24–2. Example With the State Machine Editor Block
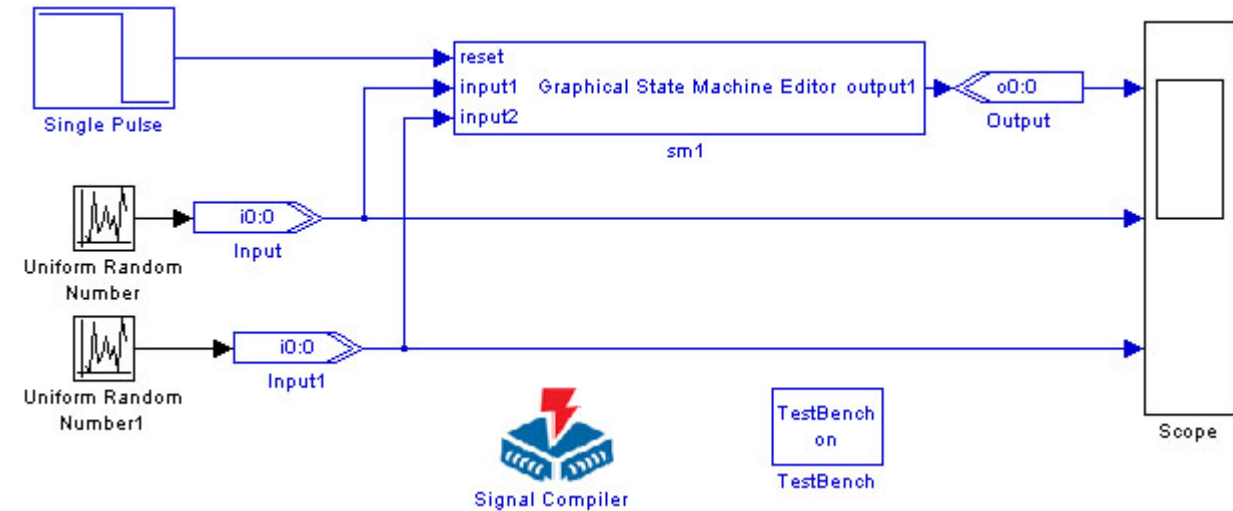
> For more information, refer to the *Using the State Machine Editor Block* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

The Storage library contains the following blocks, which support storage and associated control functions:

■ Delay

■ Down Sampling

■ Dual-Clock FIFO

■ Dual-Port RAM

■ FIFO Buffer

■ LUT (Look-Up Table)

■ Memory Delay

■ Parallel To Serial

■ ROM

■ Serial To Parallel

■ Shift Taps

■ Single-Port RAM

■ True Dual-Port RAM

■ Up Sampling

Delay

The `Delay` block delays the incoming data by an amount specified by the number of pipeline stages. The block accepts any data type as inputs.

Table 25–1 shows the `Delay` block inputs and outputs.

Table 25–1. Delay Block Inputs and Outputs

Signal	Direction	Description
`<unnamed>`	Input	Input data port.
`ena`	Input	Optional clock enable port.
`sclr`	Input	Optional synchronous clear port.
`<unnamed>`	Output	Output data port.

Table 25–2 shows the `Delay` block parameters.

Table 25–2. Delay Block Parameters

Name	Value	Description
Number of Pipeline Stages	User Defined (Parameterizable)	Specify the pipeline length of the block. The delay must be greater than or equal to 1.
Clock Phase Selection	User Defined	Specify the phase selection with a binary string, where a 1 indicates the phase in which the `Delay` block is enabled. For example: `1`—The block is always enabled and captures all data passing through the block (sampled at the rate 1). `10`—The block is enabled every other phase and every other data (sampled at the rate 1) passes through. `0100`—The block is enabled on the second phase of and only the second data of (sampled at the rate 1) passes through. That is, the data on phases 1, 3, and 4 do not pass through the delay block.
Use Enable Port	On or Off	Turn on to use the clock enable input (`ena`).
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear input (`sclr`).
Reset To Constant (Non-Zero) Value	On or Off	Turn on to specify a non-zero reset value. Specifying a reset value increases the hardware resources.
Reset Value	User Defined (Parameterizable)	Specify the reset value.

Table 25–3 shows the `Delay` block I/O formats.

Table 25–3. Delay Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L1].[R1]}$ I2$_{[1]}$ I3$_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC	Implicit
O	O1$_{[L1].[R1]}$	O1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit

Notes to Table 25–3:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 25–1 shows an example with the `Delay` block.

Figure 25–1. Delay Block Example
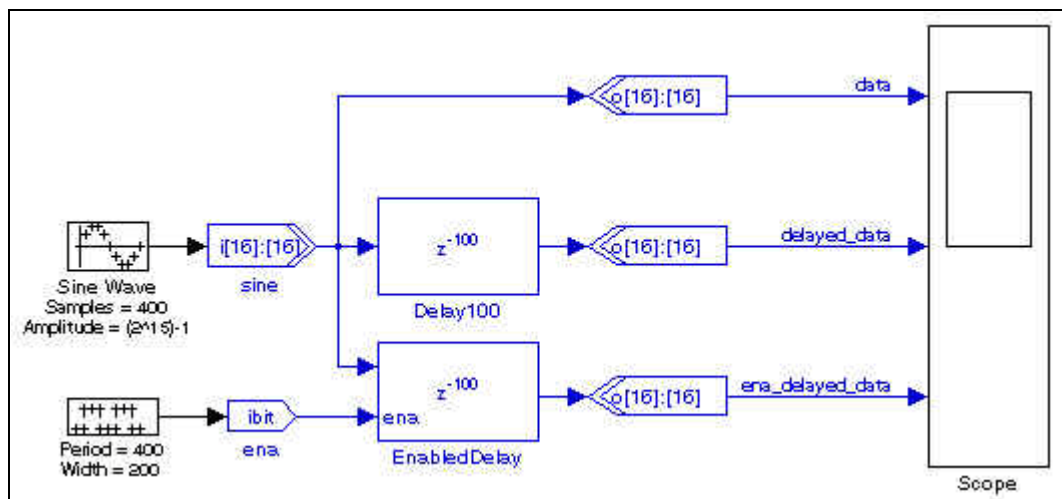
Down Sampling

The `Down Sampling` block decreases the output sample rate from the input sample rate. The output data is sampled at every *N*th cycle where *N* is the down sampling rate. The output data is then held constant for the next *N* input cycles.

Table 25–4 shows the `Down Sampling` block inputs and outputs.

Table 25–4. Down Sampling Block Inputs and Outputs

Signal	Direction	Description
d	Input	Input data port.
q	Output	Output data port.

Table 25–5 shows the `Down Sampling` block parameters.

Table 25–5. Down Sampling Block Parameters

Name	Value	Description
Down Sampling Rate	An integer greater than 1 (Parameterizable)	Specify the down sampling rate.

Table 25–6 shows the `Down Sampling` block I/O formats.

Table 25–6. Down Sampling Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
I	I1$_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit

Table 25–6. Down Sampling Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
O	O1$_{[L1].[R1]}$	O1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit

Notes to **Table 25–6**:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 25–2 shows an example with the `Down Sampling` block.

Figure 25–2. Down Sampling Block Example
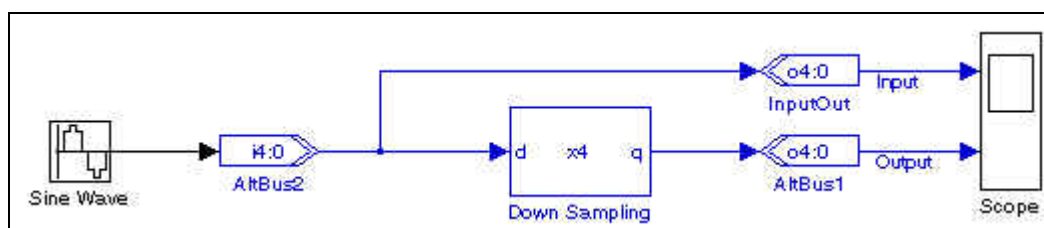
Dual-Clock FIFO

The `Dual-Clock FIFO` block implements a parameterized, dual-clock FIFO buffer controlled by separate read-side and write-side clocks.

☞ The `Dual-Clock FIFO` block simulation in Simulink is functionally equivalent to hardware, but not cycle-accurate.

Table 25–7 shows the `Dual-Clock FIFO` block inputs and outputs.

Table 25–7. Dual-Clock FIFO Block Inputs and Outputs

Signal	Direction	Description
d	Input	Data input to the FIFO buffer.
wrreq	Input	Write request control. The `d[]` port is written to the FIFO buffer.
rdreq	Input	Read request control. The oldest data in the FIFO buffer goes to the `q[]` port.
aclr	Input	Optional asynchronous clear input, which flushes the FIFO.
q	Output	Data output from the FIFO buffer.
rdfull	Output	Optional output synchronized to the read clock. Indicates that the FIFO buffer is full and disables the `wrreq` port.
rdempty	Output	Optional output synchronized to the read clock. Indicates that the FIFO buffer is empty and disables the `rdreq` port.
rdusedw	Output	Optional output synchronized to the read clock. Indicates the number of words that are in the FIFO buffer.
wrfull	Output	Optional output synchronized to the write clock. Indicates that the FIFO buffer is full and disables the `wrreq` port.

Table 25–7. Dual-Clock FIFO Block Inputs and Outputs

Signal	Direction	Description
`wrempty`	Output	Optional output synchronized to the write clock. Indicates that the FIFO buffer is empty and disables the `rdreq` port.
`wrusedw`	Output	Optional output synchronized to the write clock. Indicates the number of words that are in the FIFO buffer.

Table 25–8 shows the `Dual-Clock FIFO` block parameters.

Table 25–8. Dual-Clock FIFO Block Parameters

Name	Value	Description
Number of Words in the FIFO	Integer (Parameterizable)	Specify the FIFO depth
Input Bus Type	Signed Integer, Unsigned Integer, Signed Fractional	The bus type format.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits stored on the left side of the binary point.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This option applies only to signed fractional formats.
Memory Block Type	AUTO, M512, M4K, M9K, MLAB, M144K	The FPGA RAM type. Some memory types are not available for all device types.
Use Base Clock for Read Side	On or Off	Turn on to use the base clock signal for the read-side clock.
Read-Side Clock	User defined	Specify the read-side clock signal when not using the base clock.
Use Base Clock for Write Side	On or Off	Turn on to use the base clock signal for the write-side clock.
Write-Side Clock	User defined	Specify the write-side clock signal when not using the base clock.
Use Read-Side Synchronized EMPTY Port	On or Off	Turn on to use the read-side empty port (`rdempty`).
Use Read-Side Synchronized FULL Port	On or Off	Turn on to use the read-side full port (`rdfull`).
Use Read-Side Synchronized USEDW Port	On or Off	Turn on to use the read-side words port (`rdusedw`).
Use Write-Side Synchronized EMPTY Port	On or Off	Turn on to use the write-side empty port (`wrempty`).
Use Write-Side Synchronized EMPTY Port	On or Off	Turn on to use the write-side empty port (`wrfull`).
Use Write-Side Synchronized USEDW Port	On or Off	Turn on to use the write-side words port (`wrusedw`).
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear port (`aclr`).
Register Output	On or Off	Turn on to register the output ports. This mode is faster but larger.
Implement FIFO with logic Cells Only	On or Off	Turn on to implement the FIFO buffer with logic cells only.
Use Show-Ahead Mode of Read Request	On or Off	Turn on to use the show-ahead mode of read-request.

Table 25–9 shows the `Dual-Clock FIFO` block I/O formats.

Table 25–9. Dual-Clock FIFO Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[L1].[R1]	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
	I2[1]	I2: in STD_LOGIC	Explicit
	I3[1]	I3: in STD_LOGIC	Explicit
O	O1[L1].[R1]	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
	O2[1]	O2: out STD_LOGIC	Explicit
	O3[1]	O3: out STD_LOGIC	Explicit
	O4[1]	O4: out STD_LOGIC	Explicit
	O5[1]	O5: out STD_LOGIC	Explicit
	O6[L2].[0]	O6: out STD_LOGIC_VECTOR({L2 - 1} DOWNTO 0)	Explicit-optional
	O7[L2].[0]	O7: out STD_LOGIC_VECTOR({L2 - 1} DOWNTO 0)	Explicit-optional

Notes to Table 25–9:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 25–3 shows an example with the `Dual-Clock FIFO` block.

Figure 25–3. Dual-Clock FIFO Block Example
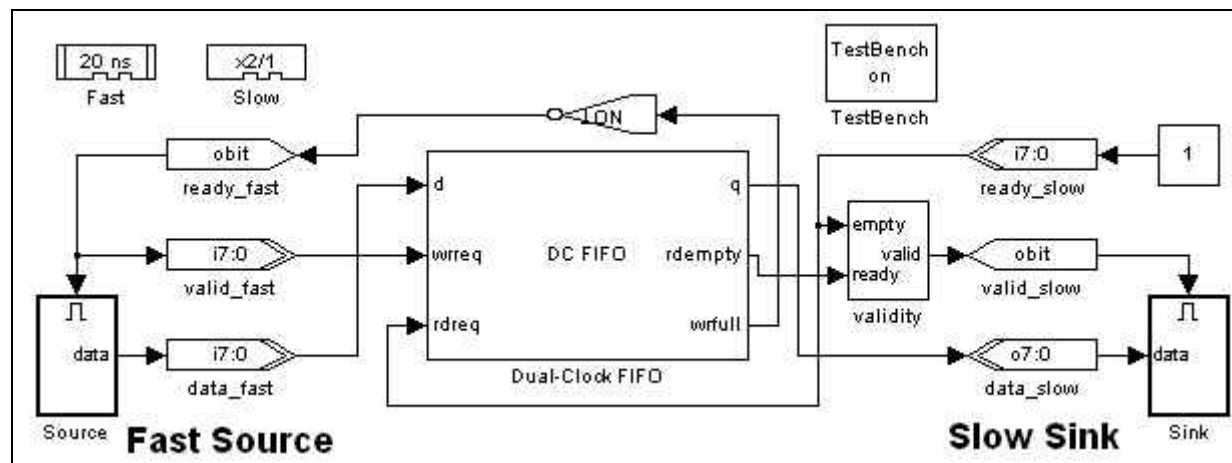
Dual-Port RAM

The `Dual-Port RAM` block maps data to an embedded RAM (embedded array block, EAB; or embedded system block, ESB) in Altera devices. The read and write ports are separate.

The `Dual-Port RAM` block accepts any data type as input. The input port always registers and the output port can optionally be registered.

☞ The input address bus must be unsigned. The clock enable signal (ena) bypasses any output register.

Turning on **DONT_CARE** may give a higher f_{MAX} for your design, especially if the memory implements as a MLAB. When this option is on, the output is not double-registered (and therefore, in the case of MLAB implementation, uses fewer external registers), and you gain an extra half-cycle on the output. The default is off, which outputs old data for read-during-write.

👣 For more information about this option, refer to the *Read-During-Write Output Behavior* section in the *RAM Megafunction User Guide.*

The contents of the RAM are pre-initialized to zero by default. Use an Intel Hexadecimal (.**hex**) file or MATLAB array to specify them. Use the Quartus Prime software to generate a.**hex** file that must be in your DSP Builder working directory.

The data in a standard .**hex** file is formatted in multiples of eight and the output bit width should also be in multiples of eight. The Quartus Prime software does allow you to create non-standard .**hex** files but pads 1's to the front for negative numbers to make them multiples of eight. Thus, large numbers with less bits may be treated as negative numbers. A warning issues if you specify a non-standard .**hex** file. If you require a different bit width, you should set the output bit width to the same as that in the .**hex** file but use an AltBus block to convert to the required bit width. DSP Builder supports 32-bit addressing with extended linear address records in the .**hex** file.

👣 For instructions on creating this file, refer to *Creating a Memory Initialization File or Hexadecimal (Intel-Format) File* in the Quartus II Help.

The MATLAB array parameter must be a one dimensional MATLAB array with a length less than or equal to the number of words. Specify the array from the MATLAB workspace or directly in the **MATLAB Array** box.

Table 25–10 shows the Dual-Port RAM block inputs and outputs.

Table 25–10. Dual-Port RAM Block Inputs and Outputs

Signal	Direction	Description
d	Input	Input data port.
rd_add	Input	Read address bus.
wr_add	Input	Write address bus.
wren	Input	Write enable.
ena	Input	Optional clock enable port
q_a	Output	Output data port.

Table 25–11 shows the `Dual-Port RAM` block parameters.

Table 25–11. Dual-Port RAM Block Parameters

Name	Value	Description
Number of words	>= 1 (Parameterizable)	Specify the address width in words.
Data Type	Inferred, Signed Integer, Unsigned Integer, Signed Fractional	The input data type format.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits stored on the left side of the binary point.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This option applies only to signed fractional formats.
Memory Block Type	AUTO, M512, M4K, M-RAM, M9K, MLAB, M144K	The FPGA RAM memory block type. Some RAM memory types are not available for all device types. If you specify M-RAM, the RAM is always initialized to `unknown` in the hardware and simultaneous read/writes to the same address also give `unknown` in hardware. Simulink does not modify the `unknowns`, and comparisons with ModelSim shows differences.
Use DONT_CARE when reading from and writing to the same address	On or Off	If the memory block type is set to **AUTO**, setting DONT_CARE gives more flexibility in RAM block placement. If the implementation is set to **MLAB**, the design uses fewer external registers, because the output is not double registered, and the resulting memory block can often be run at a higher f_{Max}. However, the output in hardware when reading from and writing to the same address is unpredictable. In ModelSim simulation, unknowns (X) are output when reading from and writing to the same address. The Simulink simulation is unchanged whether or not you use this option, but a warning message issues on every simultaneous read/write to the same address. If you compare the simulation results to ModelSim, you see mismatches associated with any read/write to the same address events. When this option is set, ensure that the same address is not read from and written to at the same time or that your design does not depend on the read output in these circumstances. By default this option is off, and data is always read before write.
Initialization	Blank, From HEX file, From MATLAB array	Specify the initialization. If `Blank` is selected, the contents of the RAM are pre-initialized to zero.
Input HEX File	User defined	Specify the name of a **.hex** file, which must be in your DSP Builder working directory. For example: `input.hex`. DSP Builder supports 32-bit addressing with extended linear address records in the .hex file.
MATLAB Array	User defined (Parameterizable)	Specify a one-dimensional MATLAB array with a length less than or equal to the number of words. For example: `[0:1:15]`
Register output Port	On or Off	Turn on to register the output port.

Table 25–11. Dual-Port RAM Block Parameters

Name	Value	Description
Use Enable Port	On or Off	Turn on to use the optional clock enable input (`ena`).
Clock Phase Selection	User Defined	Specify the phase selection with a binary string, where a 1 indicates the phase in which the block enables. For example: `1`—The block is always enabled and captures all data passing through the block (sampled at the rate 1). `10`—The block is enabled every other phase and every other data (sampled at the rate 1) passes through. `0100`—The block is enabled on the second phase of and only the second data of (sampled at the rate 1) passes through. That is, the data on phases 1, 3, and 4 do not pass through the block.

Table 25–12 shows the `Dual-Port RAM` block I/O formats.

Table 25–12. Dual-Port RAM Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$ $I2_{[L2].[0]}$ $I3_{[L2].[0]}$ $I4_{[1]}$ $I5_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L2 - 1} DOWNTO 0) I3: in STD_LOGIC_VECTOR({L3 - 1} DOWNTO 0) I4: in STD_LOGIC I5: in STD_LOGIC	Explicit
O	$O1_{[L1].[R1]}$	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit

Notes to Table 25–12:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 25–4 shows an example with the Dual-Port RAM block.

Figure 25–4. Dual-Port RAM Block Example
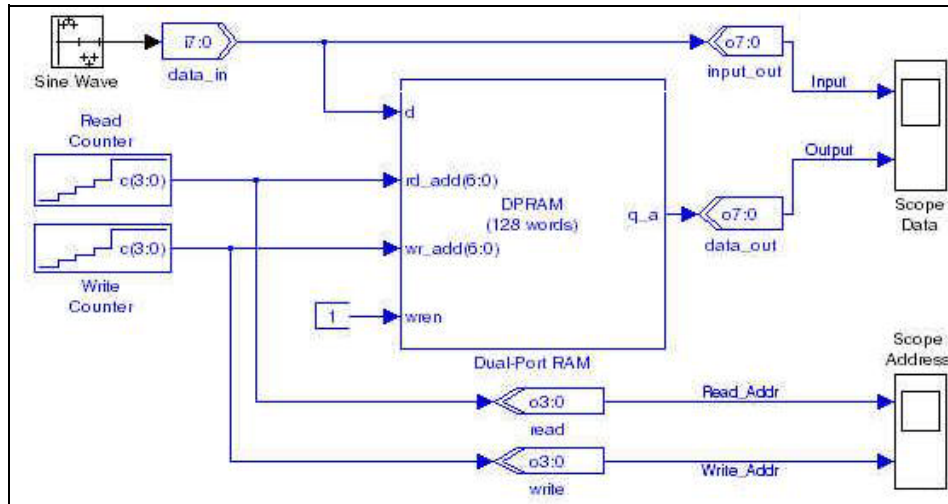
FIFO Buffer

The FIFO block implements a parameterized, single-clock FIFO buffer.

☞ Reading an empty FIFO buffer may give unknown (X) in hardware.

Table 25–13 shows the FIFO block inputs and outputs.

Table 25–13. FIFO Block Inputs and Outputs

Signal	Direction	Description
d	Input	Data input to the FIFO buffer.
wrreq	Input	Write request control. The d[] port is written to the FIFO buffer.
rreq	Input	Read request control. The oldest data in the FIFO buffer goes to the q[] port.
sclr	Input	Optional synchronous clear port that flushes the FIFO.
q	Output	Data output from the FIFO buffer.
full	Output	Indicates that the FIFO buffer is full and disables the wrreq port.
empty	Output	Indicates that the FIFO buffer is empty and disables the rreq port.
usdw	Output	Indicates the number of words that are in the FIFO buffer.

Table 25–14 shows the FIFO block parameters.

Table 25–14. FIFO Block Parameters

Name	Value	Description
Number of Words in the FIFO	User Defined (Parameterizable)	Specify how many words you want in the FIFO buffer. The default is 64.
Data Type	Inferred, Signed Integer, Signed Fractional, Unsigned Integer	The data input type format.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits stored on the left side of the binary point including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits stored on the right side of the binary point. This option applies only to signed fractional.
Memory Block Type	AUTO, M512, M4K, M9K, MLAB, M144K	The RAM block type. Some memory types are not available for all device types.
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear port (sclr).
Implement FIFO with logic Cells Only	On or Off	Turn on to implement the FIFO buffer with logic cells only.
Use Show-Ahead Mode of Read Request	On or Off	Turn on to use the show-ahead mode of read-request.

Table 25–15 shows the FIFO block I/O formats.

Table 25–15. FIFO Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$ $I2_{[1]}$ $I3_{[1]}$ $I4_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC I4: in STD_LOGIC	Explicit
O	$O1_{[L1].[R1]}$ $O2_{[1]}$ $O3_{[1]}$ $O4_{[L2].[0]}$	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) O2: out STD_LOGIC O3: out STD_LOGIC O4: out STD_LOGIC_VECTOR({L2 - 1} DOWNTO 0)	Explicit

Notes to Table 25–15:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 25–5 shows an example with the FIFO block.

Figure 25–5. FIFO Block Example
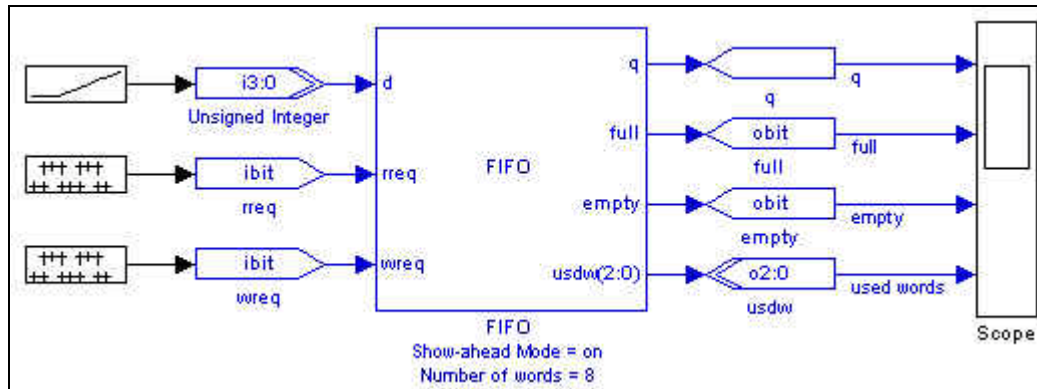
LUT (Look-Up Table)

The LUT (Look-Up Table) block stores data as $2^{(\text{address width})}$ words of data in a look-up table. The values of the words are specified in the data vector field as a MATLAB array.

Depending on the look-up table size, the synthesis tool may use logic cells or embedded array blocks (EABs), embedded system blocks (ESBs), or TriMatrix™ memory.

☞ If you want to use a **.hex** to store data, use the ROM block not the LUT block.

Table 25–16 shows the LUT block parameters.

Table 25–16. LUT Block Parameters

Name	Value	Description
Address Width	2–16	The address width as an unsigned integer.
Data Type	Signed Integer, Signed Fractional, Unsigned Integer, Single Bit	The data type format that you want to use.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of data bits stored on the left side of the binary point including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of data bits stored on the right side of the binary point.
MATLAB Array	User Defined (Parameterizable)	This field must be a one-dimensional MATLAB array with a length smaller than 2 to the power of the address width. A warning is given if the values in the MATLAB array cannot be exactly represented in the chosen data format.
Use Enable Port	On or Off	Turn on to use the optional clock enable input (ena).
Register Data	On or Off	Turn on to register the output result.

Table 25–16. LUT Block Parameters

Name	Value	Description
Use LPM	On or Off	When on, the look-up table implements as case conditions with the `lpm_rom` library of parameterized modules (LPM) function. You should turn on this option for large look-up tables, for example, greater than 8 bits. The input address always registers when this option is on.
Register Address	On of Off	When register address is on, the input address bus generates. If you use LPM, the input address is always registered.
Memory Block Type	AUTO, M512, M4K, M9K, MLAB, M144K	The RAM block type. Some memory types are not available for all device types.

Table 25–17 shows the LUT block I/O formats.

Table 25–17. LUT Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L1].[0]}$ I2$_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0) I2: in STD_LOGIC	Explicit
O	O1$_{[LPO].[RPO]}$	O1: out STD_LOGIC_VECTOR({LPO + LPO - 1} DOWNTO 0)	

Notes to Table 25–17:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 25–6 shows an example with the LUT block.

Figure 25–6. LUT Block Example
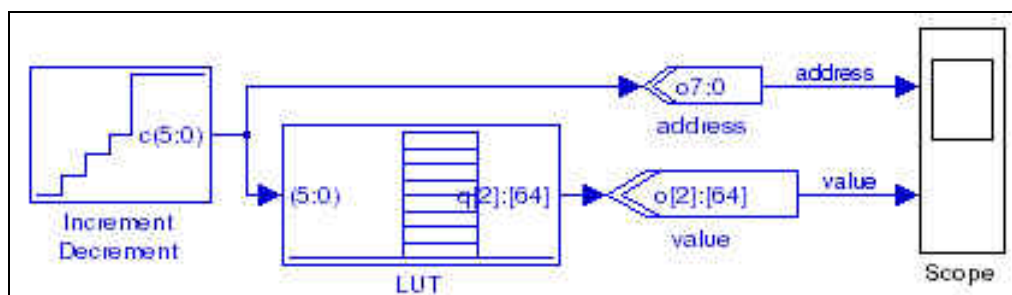
Memory Delay

The `Memory Delay` block implements a shift register that uses the Altera device's embedded memory blocks, when possible. You should typically use this block for delays greater than 3.

Table 25–21 shows the `Memory Delay` block inputs and outputs.

Table 25–18. Memory Delay Block Inputs and Outputs

Signal	Direction	Description
d	Input	Input data port.
ena	Input	Optional clock enable port.
aclr	Input	Optional asynchronous clear port.
q	Output	Output data port.

Table 25–19 shows the `Memory Delay` block parameters.

Table 25–19. Memory Delay Block Parameters

Name	Value	Description
Data Type	Inferred, Signed Integer, Signed Fractional, Unsigned Integer	The data type format that you want to use.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of data bits stored on the left side of the binary point including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of data bits stored on the right side of the binary point.
Number of Pipeline Stages	0 to number of bits (Parameterizable)	When non-zero, adds pipeline stages to increase the data throughput. The clock enable and synchronous clear ports are available only if the block is registered (that is, if the number of pipeline stages is greater than or equal to 1).
Memory Block Type	AUTO, M512, M4K, M9K, MLAB, M144K	The RAM block type. Some memory types are not available for all device types.
Use Enable Port	On or Off	Turn on to use the clock enable input.
Use Asynchronous Clear Port	On or Off	Turn on to use the asynchronous clear port (aclr).

Table 25–20 shows the `Memory Delay` block I/O formats.

Table 25–20. Memory Delay Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1$_{[L1].[R1]}$ I2$_{[1]}$ I3$_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC	Implicit
O	O1$_{[L1].[R1]}$	O1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit

Notes to Table 25–20:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 25–7 shows an example with the `Memory Delay` block.

Figure 25–7. Memory Delay Block Example
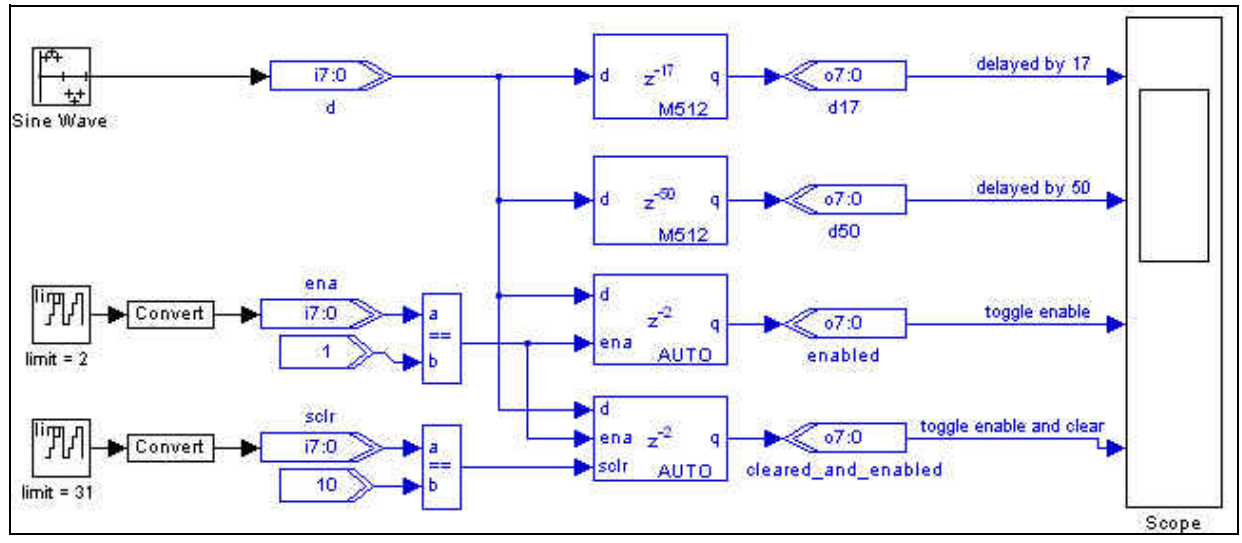
Parallel To Serial

The `Parallel To Serial` block takes a bus input on load and outputs the individual bits one cycle at a time with either the MSB or LSB first.

You can specify to continually output the last bit until the last load. For example, if input is an 8-bit unsigned integer value 1 the output is:

```
 0 ... 0 ... 0 ... 0 ... 0 ... 0 ... 0 ... 1 ... 1  ... 1 ... 1 ............
<-------------- data values --------------->|<- last bit repeated until next load ->
```

Alternatively, if this option is off, you can output 0 after the data has finished, that is, for the same example:

```
 0 ... 0 ... 0 ... 0 ... 0 ... 0 ... 0 ... 1 ... 0  ... 0 ... 0 ............
<-------------- data values --------------->|<----- zeros until next load ---->
```

Table 25–21 shows the `Parallel To Serial` block inputs and outputs.

Table 25–21. Parallel To Serial Block Inputs and Outputs

Signal	Direction	Description
d	Input	Parallel input port.
load	Input	Load port.
ena	Input	Optional clock enable port.
sclr	Input	Optional synchronous clear port.
sd	Output	Serial output port.

Table 25–22 shows the `Parallel To Serial` block parameters.

Table 25–22. Parallel To Serial Block Parameters

Name	Value	Description
Data Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The bus type format.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits stored on the left side of the binary point.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits stored on the right side of the binary point. This option applies only to signed fractional formats.
Serial Bit Order	MSB First, LSB First	Transmit the MSB or LSB first.
Repeat Last Bit Until Next Load	On or Off	Turn on to repeat the last bit until the next load.
Use Enable Port	On or Off	Turn on to use the clock enable input.
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear port (`sclr`).

Table 25–23 shows the `Parallel To Serial` block I/O formats.

Table 25–23. Parallel To Serial Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$ $I2_{[1]}$ $I3_{[1]}$ $I4_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC I4: in STD_LOGIC	Explicit
O	$O1_{[1]}$	O1: out STD_LOGIC	Explicit

Notes to Table 25–23:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 25–8 shows an example with the `Parallel To Serial` block.

Figure 25–8. Parallel To Serial Block Example
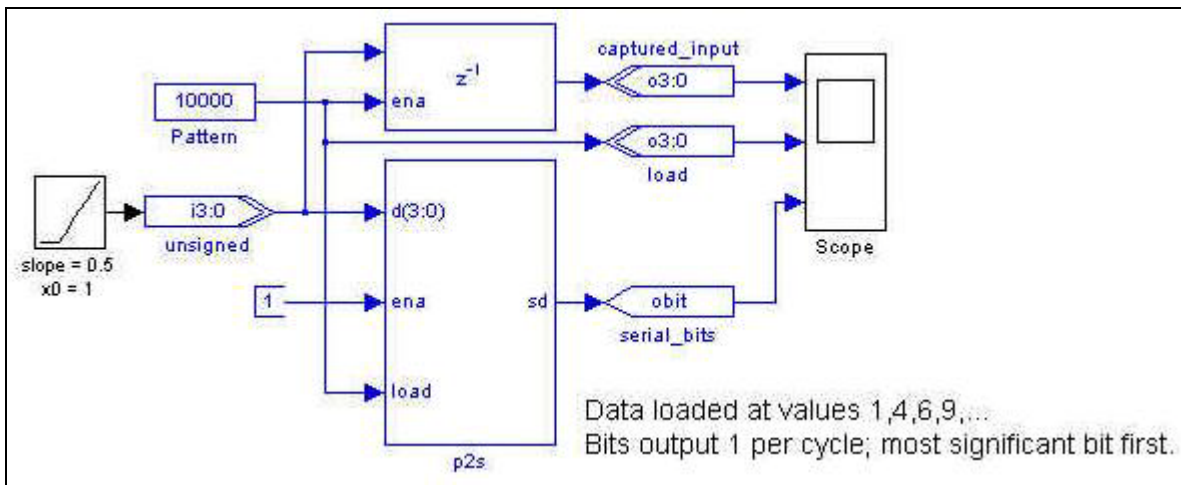
ROM

The `ROM` block maps data to an embedded RAM (embedded array block, EAB; or embedded system block, ESB) in Altera devices, with read-only access. The `ROM` block can store any data type. The address port is registered, and you can optionally register the data output port.

☞ The input address bus must be `Unsigned`. The clock enable signal (`ena`) bypasses any output register.

The contents of the ROM are pre-initialized from an Intel Hexadecimal (.**hex)** format file, or from a MATLAB array.

Use the Quartus II software to generate a .**hex** file that you must save in your DSP Builder working directory.

The data in a standard .**hex** file is formatted in multiples of eight and the output bit width should also be in multiples of eight. The Quartus Prime software does allow you to create non-standard .**hex** files but pads 1's to the front for negative numbers to make them multiples of eight. Thus, large numbers with less bits may be treated as negative numbers. A warning issues if you specify a non-standard .**hex** file. If you require a different bit width, you should set the output bit width to the same as that in the .**hex** file but use an AltBus block to convert to the required bit width. DSP Builder supports 32-bit addressing with extended linear address records in the .**hex** file.

👣 For instructions on creating a .**hex** file, refer to *Creating a Memory Initialization File or Hexadecimal (Intel-Format) File* in the Quartus II Help.

The MATLAB array parameter must be a one dimensional MATLAB array with a length less than or equal to the number of words. Specify the array from the MATLAB workspace or directly in the MATLAB Array box.

Table 25–24 shows the ROM block inputs and outputs.

Table 25–24. ROM Block Inputs and Outputs

Signal	Direction	Description
addr	Input	Input data port.
ena	Input	Optional clock enable port.
q	Output	Output data port.

Table 25–25 shows the ROM block parameters.

Table 25–25. ROM Block Parameters

Name	Value	Description
Number of Words	User Defined (Parameterizable)	Specify the depth of the ROM in words.
Data Type	Signed Integer, Signed Fractional, Unsigned Integer	The data type format.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits stored on the left side of the binary point including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits stored on the right side of the binary point. This option applies only to signed fractional formats.
Memory Block Type	AUTO, M512, M4K, M9K, MLAB, M144K	The RAM block type. Some memory types are not available for all device types.
Initialization	From HEX file, From MATLAB array	Specify whether the ROM is initialized from a **.hex** file or from a MATLAB array.
Input HEX File	User defined	Specify the name of a **.hex** file that must be in your DSP Builder working directory. For example: `input.hex`. DSP Builder supports 32-bit addressing with extended linear address records in the **.hex** file.
MATLAB Array	User defined (Parameterizable)	Specify a one-dimensional MATLAB array with a length less than or equal to the number of words. For example: `[0:1:15]`
Register output Port	On or Off	Turn on to register the output port.
Use Enable Port	On or Off	Turn on to use the optional clock enable input (`ena`).
Clock Phase Selection	User Defined	Specify the phase selection with a binary string, where a 1 indicates the phase in which the block is enabled. For example: `1`—The block is always enabled and captures all data passing through the block (sampled at the rate 1). `10`—The block is enabled every other phase and every other data (sampled at the rate 1) passes through. `0100`—The block is enabled on the second phase of and only the second data of (sampled at the rate 1) passes through. That is, the data on phases 1, 3, and 4 do not pass through the delay block.

Table 25–26 shows the ROM block I/O formats.

Table 25–26. ROM Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[L1].[0] I2[1]	I1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0) I2: in STD_LOGIC	Explicit
O	O1[LPO].[RPO]	O1: out STD_LOGIC_VECTOR({LPO + RPO - 1} DOWNTO 0)	Explicit

Notes to Table 25–26:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 25–9 shows an example with the ROM block that reads a 256×8 ramp waveform .**hex** file.

Figure 25–9. ROM Block Example
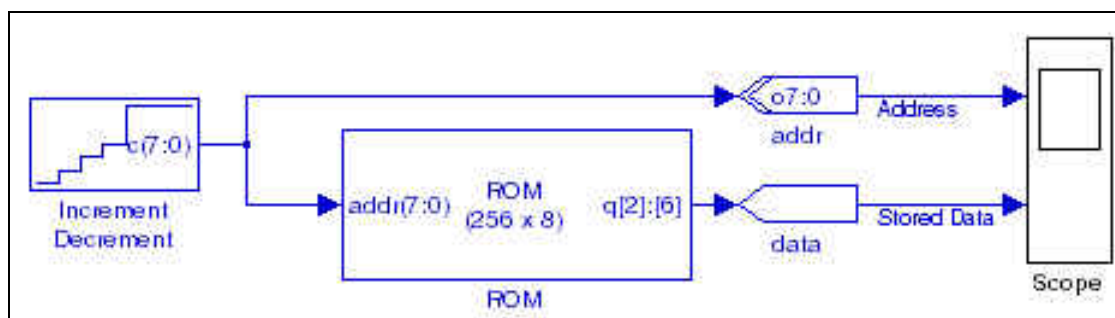
Serial To Parallel

The Serial To Parallel block implements a serial (input sd) to parallel bus conversion (output d). Treat the input bit stream as either MSB first, or LSB first.

Table 25–27 shows the Serial To Parallel block inputs and outputs.

Table 25–27. Serial To Parallel Block Inputs and Outputs

Signal	Direction	Description
sd	Input	Serial input port.
ena	Input	Optional clock enable port.
sclr	Input	Optional synchronous clear port.
d	Output	Parallel output port.

Table 25–28 shows the `Serial To Parallel` block parameters.

Table 25–28. Serial To Parallel Block Parameters

Name	Value	Description
Data Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	The bus type format.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits stored on the left side of the binary point including the sign bit.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits stored on the right side of the binary point. This option applies only to signed fractional formats.
Serial Bit Order	MSB First, LSB First	Transmit the MSB or LSB first.
Use Enable Port	On or Off	Turn on to use the clock enable input.
Use Synchronous Clear Port	On or Off	Turn on to use the synchronous clear port (`sclr`).

Table 25–29 shows the `Serial To Parallel` block I/O formats.

Table 25–29. Serial To Parallel Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[1] I2[1] I3[1]	I1: in STD_LOGIC I2: in STD_LOGIC I3: in STD_LOGIC	Explicit
O	O1[L1].[R1]	O1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit

Notes to Table 25–29:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 25–10 shows an example with the `Serial To Parallel` block.

Figure 25–10. Serial To Parallel Block Example
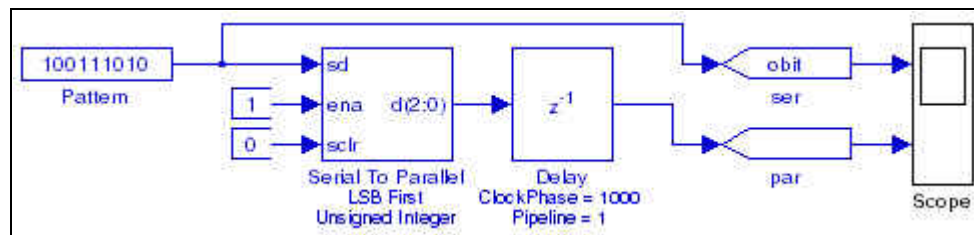
Shift Taps

The `Shift Taps` block implements a shift register that you can use for filters or convolution.

In Stratix IV, Stratix III, Stratix II, Stratix II GX, Stratix GX, Arria GX, Arria II GX, Cyclone III, Cyclone II, and Cyclone devices, the block implements a RAM-based shift register that is useful for creating very large shift registers efficiently. The block outputs occur at regularly spaced points along the shift register (that is, taps).

In Stratix devices, this block implements in the small memory.

Table 25–30 shows the `Shift Taps` block inputs and outputs.

Table 25–30. Shift Taps Block Inputs and Outputs

Signal	Direction	Description
d	Input	Data input port.
ena	Input	Optional clock enable port.
t0-tn	Output	Output ports for taps 0–n.
sout	Output	Optional shift out port.

Table 25–31 shows the `Shift Taps` block parameters.

Table 25–31. Shift Taps Block Parameters

Name	Value	Description
Number of Taps	User Defined (Parameterizable)	Specifies the number of regularly spaced taps along the shift register.
Distance Between Taps	User Defined (Parameterizable)	Specifies the distance between the regularly spaced taps in clock cycles, which translates to the number of RAM words that DSP Builder uses.
Use Shift Out Port	On or Off	Turn on to create an output from the end of the shift register for cascading.
Use Enable port	On or Off	Turn on to use an additional clock enable control input.
Use Dedicated Circuitry	On or Off	Turn on to enable selection of the memory block type. This option is only valid when the **Distance Between Taps** is greater than 2.
Memory Block Type	AUTO, M512, M4K, M9K, MLAB, M144K	The RAM block type. Some memory types are not available for all device types.

Table 25–32 shows the `Shift Taps` block I/O formats.

Table 25–32. Shift Taps Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[L1].[R1]	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit
	I2[1]	I2: in STD_LOGIC	Explicit

Table 25–32. Shift Taps Block I/O Formats [(1)]

I/O	Simulink [(2)], [(3)]	VHDL	Type [(4)]
O	$O1_{[L1].[R1]}$ …. $Oi_{[L1].[R1]}$ … $On_{[L1].[R1]}$	O1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) …. Oi: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) …. On: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit
	$O_{n+1[1]}$	O_{n+1}: out STD_LOGIC	Explicit

Notes to Table 25–32:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a Bus Conversion block to set the width.

Figure 25–11 shows an example with the Shift Taps block.

Figure 25–11. Shift Taps Block Example
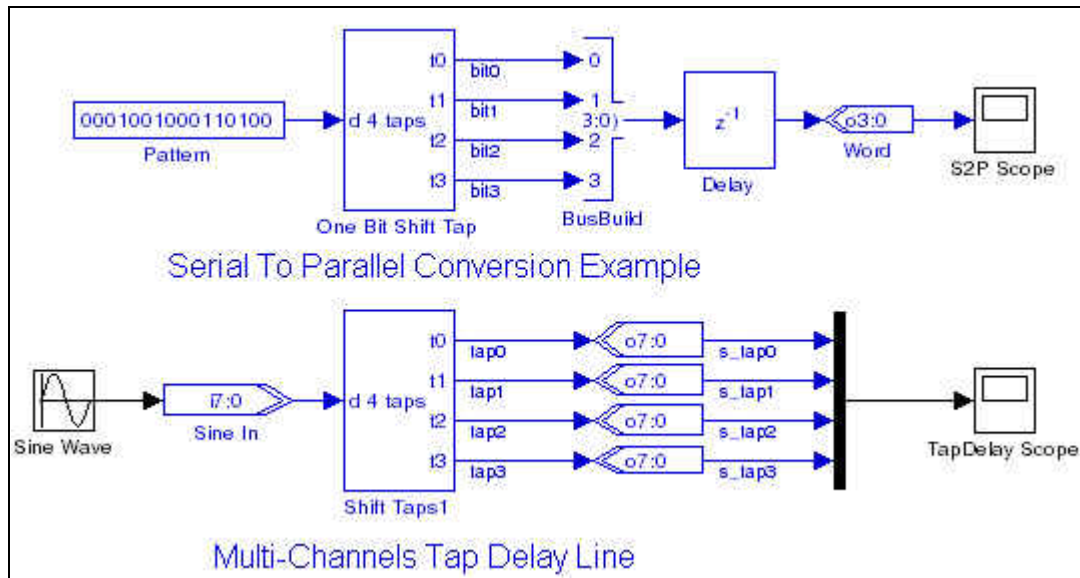
Single-Port RAM

The Single-Port RAM block maps data to an embedded RAM (embedded array block, EAB; or embedded system block, ESB) in Altera devices. A single read/write port allow simple access.

The Single-Port RAM block accepts any type as data input. The input port is registered, and the output port can optionally be registered. The input address bus must be Unsigned. The clock enable signal (ena) bypasses any output register.

The contents of the RAM are pre-initialized to zero by default. Use an Intel Hexadecimal (.**hex**) file or a MATLAB array to specify them.

Use the Quartus Prime software to generate a .**hex** file that must be in your DSP Builder working directory.

The data in a standard .**hex** file is formatted in multiples of eight and the output bit width should also be in multiples of eight. The Quartus Prime software does allow you to create non-standard .**hex** files but pads 1's to the front for negative numbers to make them multiples of eight. Thus, large numbers with less bits may be treated as negative numbers. A warning issues if you specify a non-standard .**hex** file. If you require a different bit width, you should set the output bit width to the same as that in the .**hex** file but use an AltBus block to convert to the required bit width. DSP Builder supports 32-bit addressing with extended linear address records in the .hex file.

For instructions on creating this file, refer to *Creating a Memory Initialization File or Hexadecimal (Intel-Format) File* in the Quartus II Help.

The MATLAB array parameter must be a one dimensional MATLAB array with a length less than or equal to the number of words. Specify the array from the MATLAB work-space or directly in the MATLAB Array box.

Table 25–33 shows the Single-Port RAM block inputs and outputs.

Table 25–33. Single-Port RAM Block Inputs and Outputs

Signal	Direction	Description
d	Input	Input data port.
addr	Input	Address bus.
wren	Input	Write enable.
ena	Input	Optional clock enable port
q_a	Output	Output data port.

Table 25–34 shows the Single-Port RAM block parameters.

Table 25–34. Single-Port RAM Block Parameters

Name	Value	Description
Number of words	>= 1 (Parameterizable)	Specify the address width in words.
Data Type	Inferred, Signed Integer, Unsigned Integer, Signed Fractional	The input data type format.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits stored on the left side of the binary point.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This option applies only to signed fractional formats.
Memory Block Type	AUTO, M512, M4K, M-RAM, M9K, MLAB, M144K	The FPGA RAM memory block type. Some memory types are not available for all device types. If you specify M-RAM, the RAM is always initialized to unknown in the hardware and simultaneous read/writes to the same address also give unknown in hardware. The unknowns are not modeled in Simulink, and comparisons with ModelSim shows differences.
Initialization	Blank, From HEX file, From MATLAB array	Specify the initialization. If Blank is selected, the contents of the RAM are pre-initialized to zero.

Table 25–34. Single-Port RAM Block Parameters

Name	Value	Description
Input HEX File	User defined	Specify the name of a **.hex** file that must be in your DSP Builder working directory. For example: `input.hex`. DSP Builder supports 32-bit addressing with extended linear address records in the .hex file.
MATLAB Array	User defined (Parameterizable)	Specify a one-dimensional MATLAB array with a length less than or equal to the number of words. For example: `[0:1:15]`
Register output Port	On or Off	Turn on to register the output port.
Use Enable Port	On or Off	Turn on to use the optional clock enable input (`ena`).
Clock Phase Selection	User Defined	Specify the phase selection with a binary string, where a 1 indicates the phase in which the block is enabled. For example: `1`—The block is always enabled and captures all data passing through the block (sampled at the rate 1). `10`—The block is enabled every other phase and every other data (sampled at the rate 1) passes through. `0100`—The block is enabled on the second phase of and only the second data of (sampled at the rate 1) passes through. That is, the data on phases 1, 3, and 4 do not pass through the delay block.

Table 25–35 shows the `Single-Port RAM` block I/O formats.

Table 25–35. Single-Port RAM Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	$I1_{[L1].[R1]}$ $I2_{[L2].[0]}$ $I3_{[1]}$ $I4_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L2 - 1} DOWNTO 0) I3: in STD_LOGIC I4: in STD_LOGIC	Explicit
O	$O1_{[L1].[R1]}$	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit

Notes to Table 25–12:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 25–12 shows an example with the `Single-Port RAM` block.

Figure 25–12. Single-Port RAM Block Example
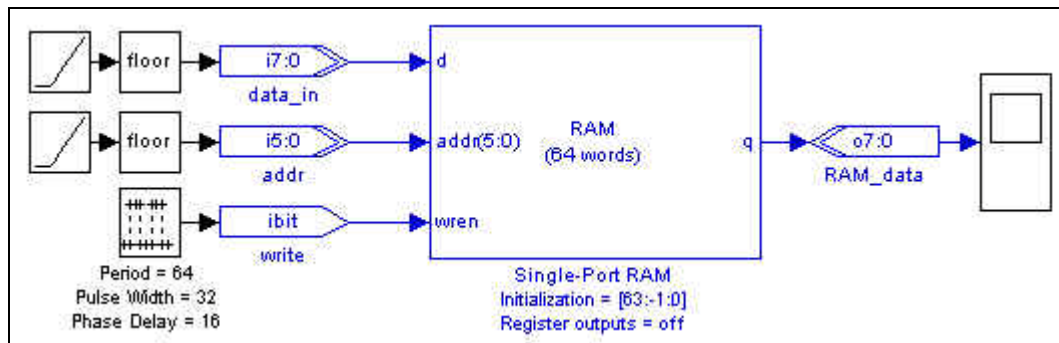
True Dual-Port RAM

The `True Dual-Port RAM` block maps data to an embedded RAM (embedded array block, EAB; or embedded system block, ESB) in Altera devices. Two read and two write ports allow true dual access.

The `True Dual-Port RAM` block accepts any data type as input. The input port is always registered and the output port can optionally be registered.

Turning on the **DONT_CARE** option may give a higher f_{MAX} for your design, especially if the memory implements as a MLAB. When this option is on, the output is not double-registered (and therefore, in the case of MLAB implementation, uses fewer external registers), and you gain an extra half-cycle on the output. The default is off, which outputs old data for read-during-write.

For more information about this option, refer to the *Read-During-Write Output Behavior* section in the *RAM Megafunction User Guide.*

The contents of the RAM are pre-initialized to zero by default. Use an Intel Hexadecimal (.**hex**) file or from a MATLAB array to specify them. Use the Quartus Prime software to generate a .**hex** file that must be in your DSP Builder working directory.

The data in a standard .**hex** file is formatted in multiples of eight and the output bit width should also be in multiples of eight. The Quartus Prime software does allow you to create non-standard .**hex** files but pads 1's to the front for negative numbers to make them multiples of eight. Thus, large numbers with less bits may be treated as negative numbers. A warning issues if you specify a non-standard .**hex** file. If you require a different bit width, you should set the output bit width to the same as that in the .**hex** file but use an AltBus block to convert to the required bit width. DSP Builder supports 32-bit addressing with extended linear address records in the .hex file.

For instructions on creating this file, refer to *Creating a Memory Initialization File or Hexadecimal (Intel-Format) File* in the Quartus II Help.

The MATLAB array parameter must be a one dimensional MATLAB array with a length less than or equal to the number of words. Specify the array from the MATLAB workspace or directly in the MATLAB Array box.

The input address bus must be `Unsigned`. The clock enable signal (`ena`) bypasses any output register.

⚠ CAUTION

If you write to the same address simultaneously with the `a` and `b` inputs, the data written to the RAM is indeterminate (corrupt). In ModelSim simulations, the data at this address is set to `Unknown` (all bits `X`). In DSP Builder simulation, the data at this address is set to zero, and a warning is given:

```
"Warning: True Dual-Port RAM: simultaneous a and b side writing to
address <addr>. Memory contents at this address will be Unknown (X)
in hardware."
```

If this data is read, DSP Builder warns that you are reading corrupt data:

```
"Warning: True Dual-Port RAM: <a|b>-side reading corrupt RAM data at
address <addr>. Memory contents at this address will be Unknown (X)
in hardware."
```

If you execute a testbench comparison to hardware, you may get simulation mismatches if you are making use of corrupt data in your design or outputting the read memory contents to a pin.

Table 25–36 shows the `True Dual-Port RAM` block inputs and outputs.

Table 25–36. True Dual-Port RAM Block Inputs and Outputs

Signal	Direction	Description
data_a	Input	Input data port a
addr_a	Input	Address bus a.
wren_a	Input	Write enable a
data_b	Input	Input data port b
addr_b	Input	Address bus b
wren_b	Input	Write enable b
ena	Input	Optional clock enable port
q_a	Output	Output data port a
q_b	Output	Output data port b

Table 25–37 shows the `True Dual-Port RAM` block parameters.

Table 25–37. True Dual-Port RAM Block Parameters

Name	Value	Description
Number of words	>= 1 (Parameterizable)	Specify the address width in words.
Data Type	Inferred, Signed Integer, Unsigned Integer, Signed Fractional	The input data type format.
[number of bits].[]	>= 0 (Parameterizable)	Specify the number of bits stored on the left side of the binary point.
[].[number of bits]	>= 0 (Parameterizable)	Specify the number of bits to the right of the binary point. This option applies only to signed fractional formats.

Table 25–37. True Dual-Port RAM Block Parameters

Name	Value	Description
Memory Block Type	AUTO, M512, M4K, M-RAM, M9K, MLAB, M144K	The FPGA RAM memory block type. Some memory types are not available for all device types. If you specify M-RAM, the RAM is always initialized to unknown in the hardware and simultaneous read/writes to the same address give unknown in hardware. The unknowns are not modeled in Simulink, and comparisons with ModelSim shows differences.
Use DONT_CARE when reading from and writing to the same address	On or Off	If the memory block type is set to **AUTO**, setting DONT_CARE gives more flexibility in RAM block placement. If the implementation is set to **MLAB**, the design uses fewer external registers, because the output is not double registered, and the resulting memory block can often be run at a higher f_{Max}. However, the output in hardware when reading from and writing to the same address is unpredictable. In ModelSim simulation, unknowns (X) are output when reading from and writing to the same address. The Simulink simulation is unchanged whether or not you use this option, but a warning message issues on every simultaneous read/write to the same address. If you compare the simulation results to ModelSim, you see mismatches associated with any read/write to the same address events. When this option is set, ensure that the same address is not read from and written to at the same time or that your design does not depend on the read output in these circumstances. By default this option is off, and data is always read before write.
Initialization	Blank, From HEX file, From MATLAB array	Specify the initialization. If Blank is selected, the contents of the RAM are pre-initialized to zero.
Input HEX File	User defined	Specify the name of an **.hex** file, which must be in your DSP Builder working directory. For example: input.hex. DSP Builder supports 32-bit addressing with extended linear address records in the .hex file.
MATLAB Array	User defined (Parameterizable)	Specify a one-dimensional MATLAB array with a length less than or equal to the number of words. For example: [0:1:15]
Register output Ports	On or Off	Turn on to register the output ports.
Use Enable Port	On or Off	Turn on to use the optional clock enable input (ena).
Clock Phase Selection	User Defined	Specify the phase selection with a binary string, where a 1 indicates the phase in which the block is enabled. For example: 1—The block is always enabled and captures all data passing through the block (sampled at the rate 1). 10—The block is enabled every other phase and every other data (sampled at the rate 1) passes through. 0100—The block is enabled on the second phase of and only the second data of (sampled at the rate 1) passes through. That is, the data on phases 1, 3, and 4 do not pass through the block.

Table 25–38 shows the `True Dual-Port RAM` block I/O formats.

Table 25–38. True Dual-Port RAM Block I/O Formats [1]

I/O	Simulink [2], [3]	VHDL	Type [4]
I	I1[L1].[R1]	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
	I2[L2].[0]	I2: in STD_LOGIC_VECTOR({L2 - 1} DOWNTO 0)	
	I3[L2].[0]	I3: in STD_LOGIC_VECTOR({L3 - 1} DOWNTO 0)	
	I4[L1].[R1]	I4: in STD_LOGIC_VECTOR({L4 + R4 - 1} DOWNTO 0)	
	I5[L2].[0]	I5: in STD_LOGIC_VECTOR({L5 - 1} DOWNTO 0)	
	I6[L2].[0]	I6: in STD_LOGIC_VECTOR({L6 - 1} DOWNTO 0)	
	I7[1]	I7: in STD_LOGIC	
	I8[1]	I8: in STD_LOGIC	
O	O1[L1].[R1]	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit

Notes to Table 25–12:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1[L].[R] is an input port. O1[L].[R] is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 25–13 shows an example with the `True Dual-Port RAM` block.

Figure 25–13. True Dual-Port RAM Block Example
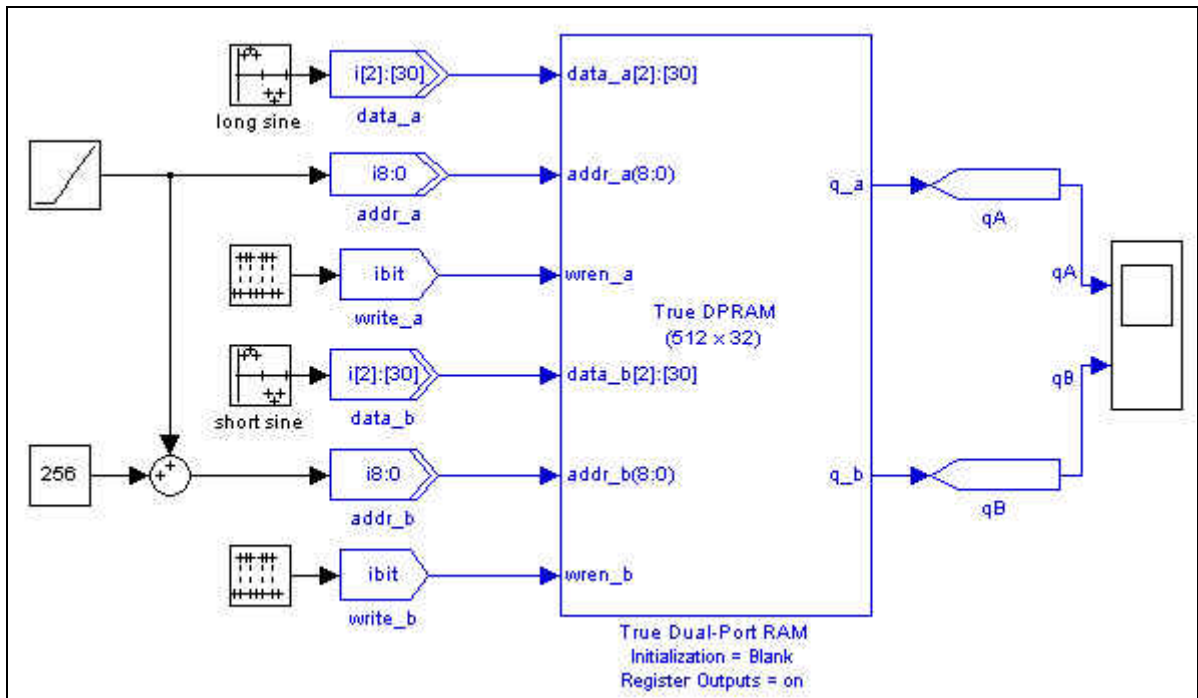
Up Sampling

The `Up Sampling` block increases the output sample rate from the input sample rate. The output data is sampled every N cycles where N is equal to the up sampling rate. The output holds this value for 1 cycle, then for the next *N*-1 cycles the output is zero.

Table 25–39 shows the `Up Sampling` block inputs and outputs.

Table 25–39. Up Sampling Block Inputs and Outputs

Signal	Direction	Description
d	Input	Input data port.
q	Output	Output data port.

Table 25–40 shows the Up Sampling block parameter.

Table 25–40. Up Sampling Block Parameter

Name	Value	Description
Up Sampling Rate	An integer greater than 1 (Parameterizable)	Specify the up sampling rate.

Table 25–41 shows the `Up Sampling` block I/O formats.

Table 25–41. Up Sampling Block I/O Formats *(1)*

I/O	Simulink *(2)*, *(3)*	VHDL	Type *(4)*
I	I1_{[L1].[R1]}	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit
O	O1_{[L1].[R1]}	O1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit

Notes to Table 25–41:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(3) I1$_{[L].[R]}$ is an input port. O1$_{[L].[R]}$ is an output port.

(4) Explicit means that the port bit width information is a block parameter. Implicit means that the port bit width information is set by the datapath bit width propagation mechanism. To specify the bus format of an implicit input port, use a `Bus Conversion` block to set the width.

Figure 25–14 shows an example with the `Up Sampling` block.

Figure 25–14. Up Sampling Block Example
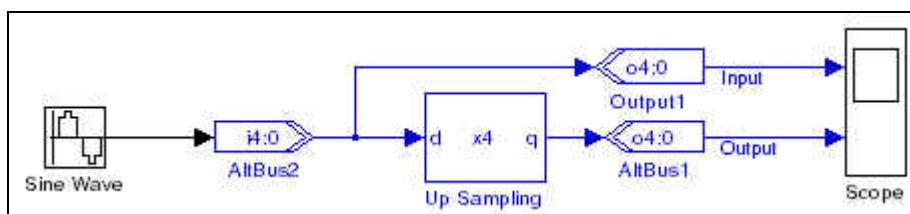
DSP Builder provides a variety of tutorials and design examples, which you can learn from or use as a starting point for your own design.

Altera supplies the following tutorials:

- Amplitude Modulation
- HIL Frequency Sweep
- Switch Control
- Avalon-MM Interface
- Avalon-MM FIFO
- HDL Import
- Subsystem Builder
- Custom Library
- State Machine Editor

Altera supplies the following design examples:

- CIC Interpolation (3 Stages x75)
- CIC Decimation (3 Stages x75)
- Convolution Interleaver Deinterleaver
- IIR Filter
- 32 Tap Serial FIR Filter
- MAC based 32 Tap FIR Filter
- Color Space Converter
- Farrow Based Resampler
- CORDIC, 20 bits Rotation Mode
- Imaging Edge Detection
- Quartus Prime Assignment Setting Example
- SignalTap II Filtering Lab
- SignalTap II Filtering Lab with DAC to ADC Loopback
- Arria II GX FPGA Development Board
- Cyclone IV GX FPGA Development Board (HSMC A)
- Cyclone IV GX FPGA Development Board (HSMC B)
- Stratix IV E FPGA Development Board
- Stratix IV GX/GT 40G/100G Interlaken Development Board

The following additional design examples demonstrate how you can combine blocks from the advanced and standard blocksets in a single design:

■ Combined Blockset Example

Viewing the Design Examples

To view the design examples:

1. To view the design examples:

    a. In MATLAB, click the **Help** icon and click **Supplemental software**.

    b. Alternatively, at the MATLAB command prompt type one of the following commands:

    ```
    demo blockset 'Altera DSP Builder Advanced Blockset'
    demo blockset 'Altera DSP Builder Standard Blockset'
    ```

2. Expand **Examples**.

3. Expand **DSP Builder Standard Blockset**. The design examples are in the following folders:

    ■ **Tutorial Designs**

    ■ **Demonstration Designs**

Displaying a Model

To display the model corresponding to each design example, click **Open this model.** For example, if you click **Open this model** for the HIL example, the model design window opens displaying the HIL frequency sweep model (Figure 26–1).

Figure 26–1. Hardware in the Loop Design Example
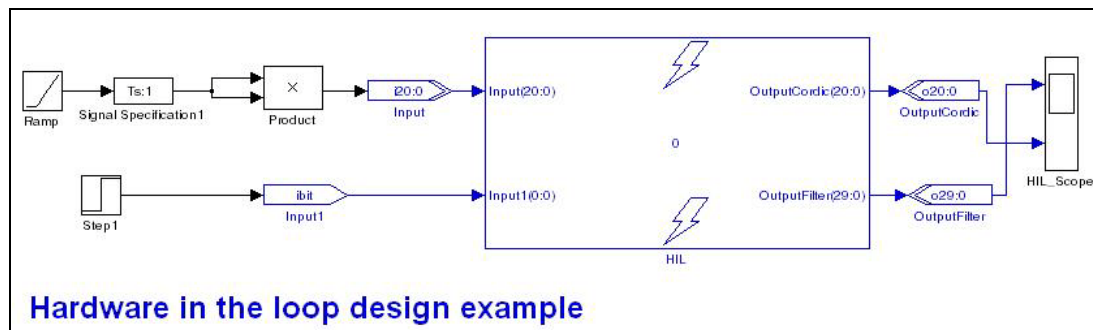
The *<DSP Builder install path>***DesignExamples****Tutorials**\ directory contains the getting started tutorials and design examples.

For more information, refer to the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

☞ You can also access simple design examples for most of the blocks in the DSP Builder blockset that correspond to the examples in the block descriptions. Many of these example blocks include Simulink Scope blocks that display the output waveforms when you simulate the models. Access these examples in the directory *<DSP Builder install path>***\DesignExamples\Tutorials\UnitBlocks**

Amplitude Modulation

Open this model

The *Getting Started Tutorial* uses the amplitude modulation design example to demonstrate the DSP Builder design flow. The design example is a modulator that has a sine wave generator, a quadrature multiplier, and a delay element.

The design example file is **singen.mdl**.

For more information about this design, refer to the *Getting Started Tutorial* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

HIL Frequency Sweep

Open this model

This HIL design shows a low-pass filter on the output of a modulated sine wave generation that the CORDIC algorithm creates.

The design example file is **FreqSweep_HIL.mdl**.

For more information about this design, refer to the *HIL Example Designs* in the *Using Hardware in the Loop (HIL)* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

Switch Control

Open this model

This design example shows how you can use blocks to control the switches on a DSP Development board and how to perform the SignalTap II analysis in DSP Builder.

The design example file is **switch_control.mdl**.

For more information about this design, refer to the *SignalTap II Design Example* in the *Performing SignalTap II Logic Analysis* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

Avalon-MM Interface

Open this model

This example consists of a 4-tap FIR filter with variable coefficients. The coefficients load with an Avalon-MM write slave while an off-chip source supplies the input data through an analog-to-digital converter. The design example sends filtered output data off-chip through a digital-to-analog converter. You can include the design as an Qsys peripheral to the Avalon-MM bus.

The design example file is **topavalon.mdl**.

For more information about this design, refer to the *Avalon-MM Interface Blocks* in the *Using the Interfaces Library* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

The design example uses a Stratix III EP3SL150 development board but you can configure the design for other boards (for example, the Cyclone III EP3C120 development board). Altera provide alternative design examples in the subdirectories under the *<DSP Builder install path>***DesignExamples\Tutorials\SOPCBuilder\SOPCBlock\Finished Examples** directory.

Avalon-MM FIFO

Open this model

This design example consists of a Prewitt edge detector with one Avalon-MM Write FIFO buffer and one Avalon-MM Read FIFO buffer. DSP Bu idler uses an additional slave port as a control port. You can include the design as an Qsys peripheral to the Avalon-MM bus.

The design example file is **sopc_edge_detector.mdl**.

For more information about this design, refer to *Avalon-MM FIFO Buffer* in the *Using the Interfaces Library* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

HDL Import

Open this model

This design example is a template design that you can use to create a simple, implicit, black-box model with the HDL Import block.

The design example file is **empty_MyFilter.mdl**.

For more information about this design, refer to *HDL Import* in the *Using Black-Box Designs for HDL Subsystems* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

Subsystem Builder

Open this model

This design example allows you to create a simple, explicit, black-box model with the Subsystem Builder block.

The design example file is **filter8tap.mdl**.

For more information about this design, refer to *Subsystem Builder* in the *Using Black-Box Designs for HDL Subsystems* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

Custom Library

Open this model

This design example shows how you can use a custom library block to implement a parameterizable Simulink block.

The design example file is **top.mdl**.

For more information and procedures to create your own library block, refer to the *Using Custom Library Blocks* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

State Machine Editor

Open this model

This example shows how you can use a `State Machine Editor` block to implement a FIFO controller in DSP Builder.

The design example file is **fifo_control_logic.mdl**.

For more information about this design, refer to the *State Machine Example Designs* in the *Using the State Machine Library* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

Demonstration Designs

The *<DSP Builder install path>***\DesignExamples\Demos** directory contains additional design examples.

CIC Interpolation (3 Stages x75)

Open this model

CIC (cascaded integrator and comb) structures are an economical way to implement high sample rate conversion filters. This example implements a 3-stage interpolating CIC filter with a rate change factor of 75, therefore, the output is 75 times faster than the input. The design uses Stratix or Cyclone device PLLs. The input frequency is 2 MHz and the output is 150 MHz.

The design example file is **CiCInterpolator75.mdl**.

CIC Decimation (3 Stages x75)

Open this model

CIC (cascaded integrator and comb) structures are an economical way to implement high sample rate conversion filters. This example implements a 3-stage decimating CIC filter with a rate change factor of 75, therefore, the output is 75 times slower than the input. Use this design in digital down-conversion applications. The design uses Stratix or Cyclone device PLLs. The input frequency is 150 MHz and the output is 2 MHz.

The design example file is **CicDecimator75.mdl**.

Convolution Interleaver Deinterleaver

Open this model

Use convolution interleaver deinterleavers on the transmission side for forward error correction. It provides an example of how the interleaver and deinterleaver work together. The example uses a Memory Delay block for the interleaver FIFO buffers.

The design example file is **top12x17.mdl**.

IIR Filter

Open this model

This design example illustrates how to implement an order 2 IIR filter with a direct form two structure. The coefficients compute with the MATLAB function butter, which implements a Butterworth filter, with an order of two and a cutoff frequency of 0.4. This function creates floating-point coefficients, which are scaled in the design with the Gain block.

The design example file is **topiir.mdl.**

32 Tap Serial FIR Filter

Open this model

This design example illustrates how to implement a low pass 32 tap FIR (finite impulse response) filter with a 4-8 look-up table (LUT) for partial product pre-computation. This design requires the Mathworks Signal Processing ToolBox to calculate the coefficient with the FIR1 function:

```
FilterOrder = 32
InputBitWidth = 8
LowPassFreqBand = [0 0.1 0.2 1];
LowPassMagnBand = [1 0.9 0.0001 0.0001];
FlCoef =   firls(FilterOrder,LowPassFreqBand,LowPassMagnBand);
CoefBitWidth = InputBitWidth +
ceil(log2((max(abs(FlCoef))/min(abs(FlCoef)))))
ScalingFactor = (2^(CoefBitWidth-1))-1;
FpCoef = fix(ScalingFactor * FlCoef);
plot(FpCoef,'o');
title('Fixed-point scaled coefficient value');
ImpulseData = zeros(1,1000);
ImpulseData(1) = 100;
h = conv(ImpulseData,FpCoef);
fftplot(h);
title('FIR Frequency response');
FirSamplingPeriod=1;
```

The design example file is **AltrFir32.mdl**.

MAC based 32 Tap FIR Filter

Open this model

This design example illustrates how to implement a MAC-based, fixed-coefficient, 32-tap, low pass FIR (finite impulse response) filter with a single Multiply Accumulate block and a single memory element for the tap delay line. This design requires the MathWorks Signal Processing ToolBox to calculate the coefficient with the fir1 function:

```
coef = fix(fir1(32,3/8)*2^16-1);
Impulse =  zeros(1,1000);
Impulse(1) = 1;
h = conv(coef,Impulse);
plot(coef,'o');
title('Fixed-point scaled coefficient value');
fftplot(h);
title('Impulse Frequency response');
```

The design example file is **FIR_MAC32.mdl**.

Color Space Converter

Open this model

This design example illustrates how to implement a color space converter, which converts R'G'B to Y'C'bCr.

The design example file is **TopCsc.mdl**.

Farrow Based Resampler

Open this model

This design example illustrates how to implement a Farrow based decimating sample rate converter.

Many integrated systems, such as software defined radios (SDR), require you to resample data so that a unit can comply with communication standards where the sample rates are different. In some cases, where one clock rate is a simple integer multiple of another clock rate, use interpolating and decimating FIR filters to accomplish resampling. However, in most cases the interpolation and decimation factors are so high that this approach is impractical.

Farrow resamplers provide an efficient way to resample a data stream at a different sample rate. The underlying principle is that the phase difference between the current input and wanted output is determined on a sample by sample basis. This phase difference then combines the phases of a polyphase filter in such a way that a sample for the output phase, which you want, generates.

This design example illustrates a Farrow resampler. You can simulate its performance in MATLAB, change it as required for your application, generate VHDL and synthesize the model to Altera devices. The design example has an input clock rate identical to the system clock. For applications where the input rate is much lower than the system clock, time sharing should be implemented to achieve a cost effective solution.

The design example file is **FarrowResamp.mdl**.

For more information about this design, click on the `Doc` symbol in the design model window.

CORDIC, 20 bits Rotation Mode

Open this model

This design example illustrates an iterative 20 bit rotation mode, which computes sine and cosine angles and implements with the coordinate rotation digital computer (CORDIC) algorithm.

The design example file is **DemoCordic.mdl**.

Imaging Edge Detection

Open this model

This design example illustrates an edge detection design.

The design example file is **Edge_detector.mdl**.

Quartus Prime Assignment Setting Example

Open this model

This design example illustrates Quartus Prime assignment setting from DSP Builder. You can launch the Signal Compiler block to compile the design and program the Stratix IV development board.

The design example file is **TopEP4SE530Board.mdl**.

SignalTap II Filtering Lab

Open this model

Two numerically-controlled oscillators generate a 833.33kHz sinusoidal signal and a 83.33kHz sinusoidal signal. The design example adds the signals together. The resulting signal loops back to a low-pass 34-tap filter with 14-bit fixed-point coefficients. The low-pass filter removes the 833.33-kHz sinusoidal signal and allows the 83.33-kHz sinusoidal signal through to the `fir_result` output.

The design example file is **FilteringLab.mdl**.

SignalTap II Filtering Lab with DAC to ADC Loopback

Open this model

Two numerically-controlled oscillators generate a 833.33kHz sinusoidal signal and a 83.33kHz sinusoidal signal. The design example adds the signals together on chip before they pass through a digital-to-analog converter on a Stratix IV GX development board. The resulting analog signal loops back to an analog-to-digital converter on the board and then passes to an on-chip, low-pass filter. The low-pass filter removes the 833.33kHz sinusoidal signal and allows the 83.33kHz sinusoidal signal through to the `fir_result` output.

The design example file is **StFilteringLab.mdl**.

Arria II GX FPGA Development Board

Open this model

The design example file is **TestEP2AGX125_Leds.mdl**.

Cyclone IV GX FPGA Development Board (HSMC A)

Open this model

The design example file is **TestEP4CGX150_HMCA.mdl**.

Cyclone IV GX FPGA Development Board (HSMC B)

Open this model

The design example file is **TestEP4CGX150_HMCB.mdl**.

Stratix IV E FPGA Development Board

Open this model

The design example file is **TestEP4SE530_QuadDisplay.mdl**.

Stratix IV GX/GT 40G/100G Interlaken Development Board

Open this model

The design example file is **TestEP4S100G5_Leds.mdl**.

Combined Blockset Example

Open this model

This design example illustrates how to embed a DSP Builder Advanced Blockset design inside a top-level standard blockset design. The resulting system comprises blocks from both blocksets, simulates cycle-accurately and you can us the standard blockset `TestBench` block to test it.

The design example file is **demo_adapted_ad9856.mdl**.

For more information about this design example, refer to the *DSP Builder* chapter in the *DSP Design Flow User Guide*.

U

Up Sampling block 25–29

V

VCC block 20–22
VCD Sink block 14–17
Virtual Pins block 14–17

W

Walkthrough
    Avalon-MM FIFO 7–14
    Avalon-MM interface blocks 7–6
    Black box
        HDL import 8–1
        Subsystem Builder 8–6
    Custom library 9–1
    Hardware in the loop 5–3
    SignalTap II 6–3
    State Machine Editor 11–2
Warning message
    I/O blocks conflict with clock or aclr ports 13–6

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this document.

Date	Version	Changes
2016.08.01	16.0	■ Added legacy note to About DSP Builder and Using MegaCore functions chapters.
2015.11.01	15.1	■ Deleted MegaCore function design example ■ Changed Quartus Prime to Quartus Prime software. ■ Deprecated Avalon-ST Packet Format Converter block ■ Removed `State Machine Table` block
June 2014	4.0	■ Removed references to Cyclone III and Stratix III devices and development boards. ■ Updated instructions for viewing design examples.
November 2013	3.1	■ Removed the following demo boards and updated any references to them: ■ Cyclone II DE2 Starter board ■ Cyclone II EP2C35 board ■ Cyclone II EP2C70 board ■ Stratix EP1S25 board ■ Stratix EP1S80 board ■ Stratix II EP2S60 board ■ Stratix II EP2S180 board ■ Stratix II EP2SGX90 PCI Express board ■ Updated all SOPC Builder references to Qsys.
May 2013	3.0	■ Added new parameter to `DSP` and `Pipelined Adder` blocks. ■ Added new `Variable Precision DSP` block
November 2012	2.1	Updated `Memory Delay` block parameters and signals for asynchronous entry.
June 2012	2.0	■ Added note on asynchronous clears ■ Added information on Simulink Bus Creator, Bus Selector, and Bus Assignment blocks
November 2011	1.2	Added information on Simulink simulation resets.
April 2011	1.1	■ Removed `HIL` block frame mode references ■ Added `Virtual Pins` block
June 2010	1.0	First published.

How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

Contact [1]	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Nontechnical support (general)	Email	nacomp@altera.com
(software licensing)	Email	authorization@altera.com

Note to Table:

(1)   You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, $n + 1$. Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>*.**pof** file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections in a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
Courier type	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
☞	The hand points to information that requires special attention.

Visual Cue	Meaning
	The question mark directs you to a software help system with related information.
	The feet direct you to another document or website with related information.
	The multimedia icon directs you to a related multimedia presentation.
CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
WARNING	A warning calls attention to a condition or possible situation that can cause you injury.
	The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.
	The feedback icon allows you to submit feedback to Altera about the document. Methods for collecting feedback vary as appropriate for each document.