DSP Builder

Reference Manual

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com

Software Version: 3.0.0
Document Version: 3.0.0
Document Date: January 2005

I.S. EN ISO 9001

MNL-DSPBLDR-2.0

Contents

Chapter 11. Example Designs

About This Manual

Revision History

The table below displays the revision history for the chapters in this Manual.

Chapter	Date	Version	Changes Made
all	January 2005	3.0.0	Added support for Hardware in the Loop
all	August 2004	2.2.0	Added support for MegaCore® functions
1	March 2004	2.2.0 Beta	Added support for Stratix™ II and Cyclone™ II devices
all	July 2003	1.0.0	First publication

How to Contact Altera

For technical support or other information about Altera products, go to the Altera world-wide web site at www.altera.com. You can also contact Altera through your local sales representative or any of the sources listed below.

Information Type	USA & Canada	All Other Locations
Technical support	www.altera.com/mysupport/	www.altera.com/mysupport/
	800-800-EPLD (3753) 7:00 a.m. to 5:00 p.m. Pacific Time	+1 408-544-8767 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
Product literature	www.altera.com	www.altera.com
Altera literature services	lit_req@altera.com	lit_req@altera.com
Non-technical customer service	800-767-3753	+ 1 408-544-7000 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
FTP site	ftp.altera.com	ftp.altera.com

Typographic Conventions

This document uses the typographic conventions shown below.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: "Typographic Conventions."
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input`. Active-low signals are denoted by suffix `n`, for example, `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), as well as logic function names (for example, `TRI`) are shown in Courier.
1., 2., 3., and a., b., c., and so on	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ● •	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
⚠ CAUTION	The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.
⚠	The warning indicates information that should be read prior to starting or continuing the procedure or processes
↵	The angled arrow indicates you should press the Enter key.
👣	The feet direct you to more information on a particular topic.

Chapter 1. AltLab Library

Introduction

The blocks in the AltLab library are used to manage design hierarchy and generate RTL VHDL for synthesis and simulation.

Bus Probe Block

The Bus Probe (BP) block is a sink block, which can be placed on any node of a model. After simulating the model, the BP block back-annotates the following information in the parameter dialog box of the **BP** block:

- Sampling period
- Maximum number of bits required during simulation
- Maximum or minimum value reached during simulation

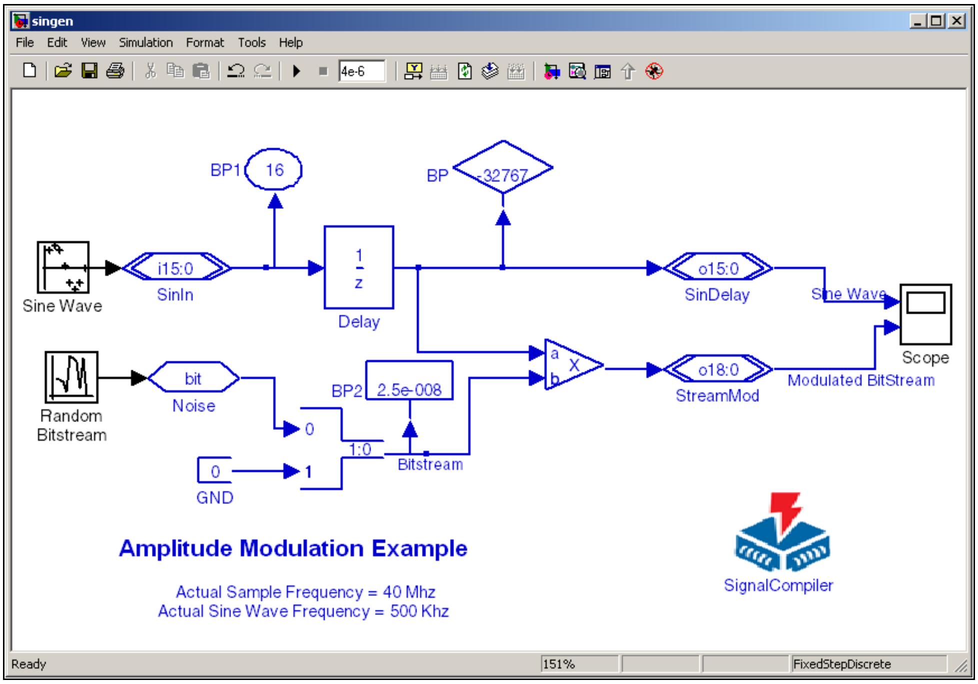
The **display in symbol** parameter selects the graphical shape of the symbol in the model and what information is reported there, as follows:

Table 1–1. Bus Probe Block "Display in Symbol" Parameter

Shape of Symbol	Data Reported in Symbol
Rectangle	Node sampling period.
Circle	Maximum number of bits required during simulation.
Diamond	Maximum or minimum value reached during simulation.

The BP block does not have any hardware representation and therefore will not appear in the VHDL RTL representation generated by the SignalCompiler block. Figure 1–1 shows example usage of the BP block.

Figure 1–1. BP Block Example Usage



Amplitude Modulation Example

Actual Sample Frequency = 40 Mhz
Actual Sine Wave Frequency = 500 Khz

Device Programmer Block

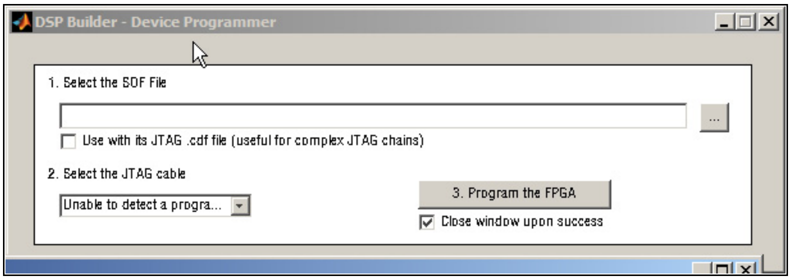
The Device programmer block provides easy access to the Quartus® II device programmer, and can work in conjunction with the Hardware in the Loop (HIL) block. It allows you to configure an FPGA using a **.sof** file (SRAM Object File), and optionally using a **.cdf** file (Chain Description File). See the Quartus II help for additional information on the **.cdf** file.

- In non-JTAG mode, the Device programmer requires only a **.sof** file.
- In JTAG mode, the Device programmer requires a **.sof** file, plus a **.cdf** file if the FPGA to configure is not the first in the JTAG chain (the **.cdf** file enables determination of the position of the FPGA in the JTAG chain).

The parameters for the Device programmer block are shown in Figure 1–2.

The Device programmer cannot create a **.cdf** file. If you require a **.cdf** file, it can be created by the Quartus II software programmer, or by the HIL block programmer.

Figure 1–2. Device Programmer Block Parameters



1. Select the SOF File

☐ Use with its JTAG .cdf file (useful for complex JTAG chains)

2. Select the JTAG cable

Unable to detect a progra...

☑ Close window upon success

For more information on FPGA configuration, see the Quartus II programmer online Help.

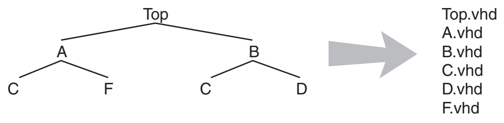
HDL SubSystem Block

You can use the HDL SubSystem block to add a level of hierarchy to your design. SignalCompiler generates a separate VHDL file for each DSP Builder HDL SubSystem block, each containing a single entity/architecture pair. For example, if you have a single model file with two subsystems, SignalCompiler creates three output files, one for the model file and one for each of the subsystems.

Note that the hierarchy of the DSP Builder model is not preserved in translated Verilog HDL files generated by SignalCompiler, just in the VHDL files.

The SignalCompiler block creates VHDL files in which the entity name space is global. The entity name is by default the HDL Subsystem name, and therefore all subsystem names are expected to be unique by default. If your model has the structure shown in Figure 1–3, both instances of subsystem C must be identical. If they are different, only one will be used.

Figure 1–3. Example Design Hierarchy



If the HDL subsystem C has a different IO port signature or a different parameter signature (such as a different sampling period), turn on "Hierarchical VHDL entity names are unique" on Page 1 of 2 of the

SignalCompiler block dialog box. When this option is on, SignalCompiler will generate the following VHDL files: a1.vhd, b2.vhd, c3.vhd, f4.vhd, d5.vhd, c6.vhd.

For a listing of the HDL SubSystem blocks used in the design, see the DSP Builder Report File, which is output when SignalCompiler finishes generating HDL.

The DSP Builder HDL SubSystem block is derived from the Simulink SubSystem block, in other words, it is a Simulink Subsystem block with the **MaskType** parameter set to `SubSystem AlteraBlockSet`. SignalCompiler uses this parameter setting to detect which subsystem to translate into VHDL. If you want to convert a Simulink subsystem to an HDL SubSystem, select the Simulink subsystem and type the following command at the MATLAB prompt:

```
set_param(gcb,'MaskType', 'SubSystem AlteraBlockSet') ↵
```

This command sets the **MaskType** parameter to `SubSystem AlteraBlockSet`.

HDL SubSystem input/output ports are defined with a combination of:

- Simulink in port blocks and DSP Builder Input blocks for the inputs
- Simulink out port blocks and DSP Builder Output blocks for the outputs

Figure 1–4 shows the input ports for this example.

Figure 1–4. HDL SubSystem Example Input Ports

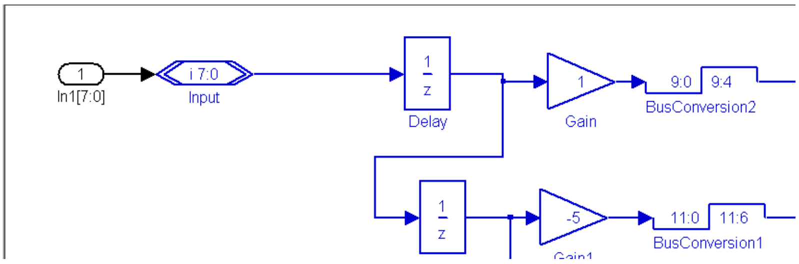
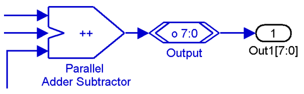


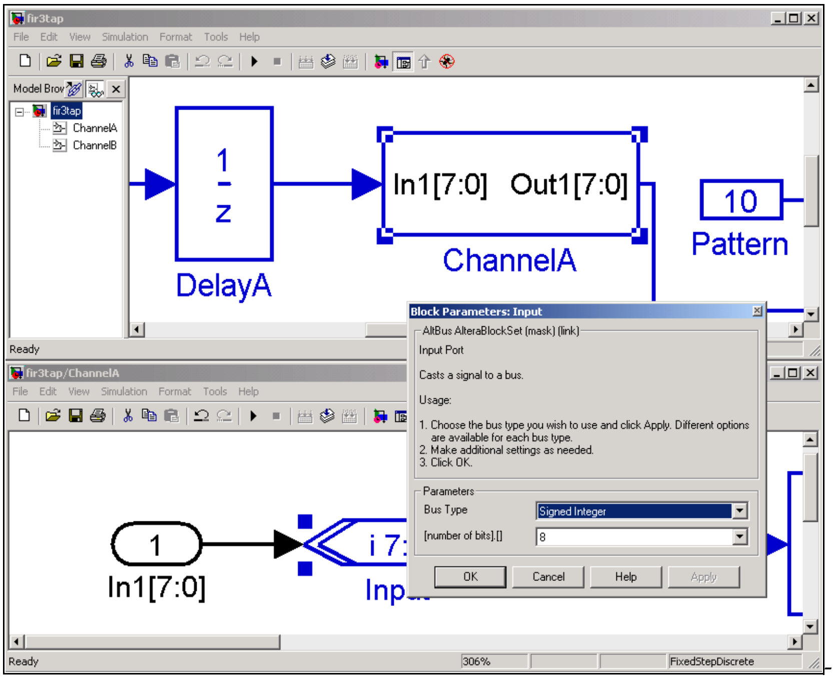
Figure 1–5 shows the output ports for this example.

Figure 1–5. HDL SubSystem Example Output Ports



☞ The DSP Builder Input block bit width information is written automatically to the Simulink in port block label name (in other words, `In1[7:0]`), and is displayed in the HDL SubSystem symbol port name as shown in Figure 1–6. The same mechanism applies for output signals.

Figure 1–6. HDL SubSystem Example Bit Width Information



Block Parameters: Input

AltBus AlteraBlockSet (mask) (link)

Input Port

Casts a signal to a bus.

Usage:

1. Choose the bus type you wish to use and click Apply. Different options are available for each bus type.
2. Make additional settings as needed.
3. Click OK.

Parameters

Bus Type: Signed Integer

[number of bits].[]: 8

OK | Cancel | Help | Apply

See "Fixed-Point Notation" on page 1–2 for additional information.

HIL Block

The HIL (Hardware In the Loop) block allows you to use an FPGA as a simulation device inside a Simulink design. This hardware accelerates the simulation time, and also allows access to real hardware within a simulation.

To use an HIL block, you need an FPGA board (Stratix® II, Stratix, Cyclone™ II, and Cyclone) with a JTAG interface. You can use any JTAG download cable, such as a BytBlasterMV™, ByteBlaster, or USB-Blaster™ cable.

The design flow is as follow:

1. Create a Quartus II project that defines the functions you want to co-simulate in hardware.

2. Compile the Quartus II project through the Quartus II Fitter step.

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

4. Specify the various parameters of the HIL block, including:
    - the Quartus II project you compiled to define its functionality,
    - the input and output pin characteristics, and
    - the use of single-step versus burst and frame mode.

5. Compile the HIL block to create a programming object file that will be used for hardware co-simulation.

6. Program the board that contains your target FPGA.

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

HIL supports advanced features, including:

- Exported ports (allows the use of hardware components connected to the FPGA)
- Burst and frame modes (improves HIL simulation speed)

Table 1–4 shows the block parameters.

Table 1–2. HIL Block Parameters

Name	Value	Description
Quartus Project	.qpf file	Hardware design used in the HIL block.
Clock Pin	Port name	Clock pin name of the hardware design in the Quartus II software.
Port Identity	Signed or Unsigned	Nature of the port.
Port Export	On or Off	When On, the 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.
Burst Mode	On or Off	Allows sending data to the FPGA in bursts. This improves the simulation speed, but delays the outputs by the burst length used.When Off, it defaults to single-step mode.
Burst Length	2 to 1000	Defines the length of a burst ("1" would be equivalent to disabling burst mode). Use higher values to produce faster simulations (although the extra gain becomes negligible as bigger burst sizes are used).
Frame Mode	On or Off	Used in burst mode when data is sent or received in frames. Allows synchronizing the output data frames to the input data frames.
Frame Input Sync	Port name	Input port used as the synchronization signal (only in Frame Mode).
Frame Output Sync	Port name	Output port used as the synchronization signal (only in Frame Mode).
Sclr	On or Off	Allows assertion of the `synchronous clear` signal before the simulation starts (only for designs created by SignalCompiler).

When using HIL in a Simulink model, remember to set the model simulation parameters as follows:

- **Solver option**: `Fixed-step`
- **Mode**: `Single-tasking`

Table 1–3 shows the HIL block parameters.

Table 1–3. HIL Block Compilation (Part 1 of 2)

Name	Needs (re)compilation of the HIL block when changed?
Quartus Project	yes
Clock Pin	yes
Port Identity	no

Table 1–3. HIL Block Compilation (Part 2 of 2)

Name	Needs (re)compilation of the HIL block when changed?
Port Export	yes
Burst Mode	no
Burst Length	no
Frame Mode	no
Frame Input Sync	no
Frame Output Sync	no
Sclr	no

Node Block

You use the Node block with the SignalTap® II Analysis block to capture signal activity from internal Altera® device nodes while the system under test runs at speed. The Node block indicates the signals (also called nodes) for which you want to capture activity. When you add a Node block, you specify a range of bits to analyze by choosing the bit position of the most significant bit (MSB) and least significant bit (LSB). For example, if you want to analyze the three most significant bits of an 8-bit bus, you would specify seven for the **MSB** parameter and five for the **LSB** parameter.

Table 1–4 shows the node block parameters.

Table 1–4. Node Block Parameters

Name	Value	Description
MSB	0 to 51	Indicate the bit position for the MSB of the range of bits you want to analyze.
LSB	0 to 51	Indicate the bit position for the LSB of the range of bits you want to analyze.

Figure 1–21 on page 1–29 shows an example using the Node block.

See the following sources for more information:

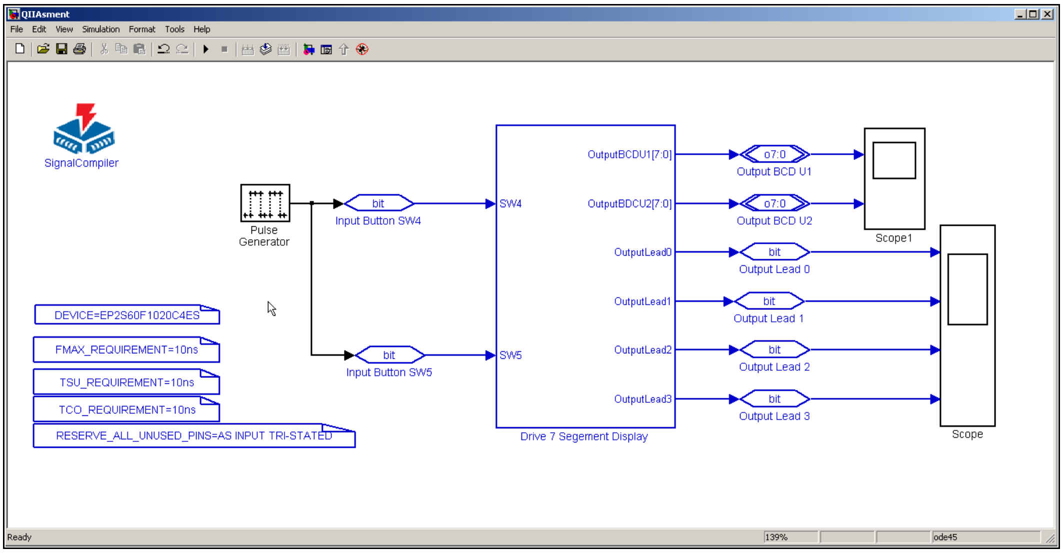
- "SignalTap II Analysis Block" on page 1–28
- The *Perform SignalTap II Analysis* section in the *Performing SignalTap II Logic Analysis* chapter in the *DSP Builder User Guide.*

Quartus II Global Project Assignment Block

This block passes Quartus II Project Global Assignments to the Quartus II project generated by the SignalCompiler blocks.

Although each block sets a single assignment, multiple blocks can be used for multiple assignments as shown in Figure 1–7.

Figure 1–7. Multiple Assignments with the Quartus II Global Project Assignment Block



These assignments could set Quartus II compilation directives such as target device or timing requirement.

☞ If identical, the Quartus II global project assignment set in this block will override the Quartus II global project assignment set in the SignalCompiler block.

Table 1–5 shows the block parameters.

Table 1–5. Quartus II Global Project Assignment Block Parameters

Name	Value	Description
Assignment Name	String	Set the assignment name of the Quartus II global assignment.
Assignment Value	String	Set the assignment value of the Quartus II global assignment.

For a description of the Quartus II Global Assignment name and value, consult the Quartus II Help.

Quartus II Pinout Assignment Block

This block passes Quartus II project pinout assignments to the Quartus II project generated by the SignalCompiler blocks.

The Quartus II Pinout Assignment block must be used only at the top level of the model. This block sets the pinout location of the Input or Output blocks from the IO & Bus DSP Builder Simulink library folder. You must use the Quartus II Global Project Assignment block when you use the Quartus II Pinout Assignment block in order to set a target device for the Quartus II project generated by SignalCompiler.

For buses, use a comma to separate the bit pin assignment location from LSB to MSB, that is:

```
    pin name=abc:pin value="Pin_AA, Pin_AB, Pin_AC"
```

will assign `abc[0]` to `Pin_AA`, `abc[1]` to `Pin_AB`, and `abc[2]` to `Pin_AC`.

To set the pin assignment of the implicit hardware system clock, use **clock** for the pin name, that is:

```
    pin name =clock:pin value = Pin_AM17
```

To set the pin assignment of the implicit hardware global reset, use **sclr** for the pin name, that is:

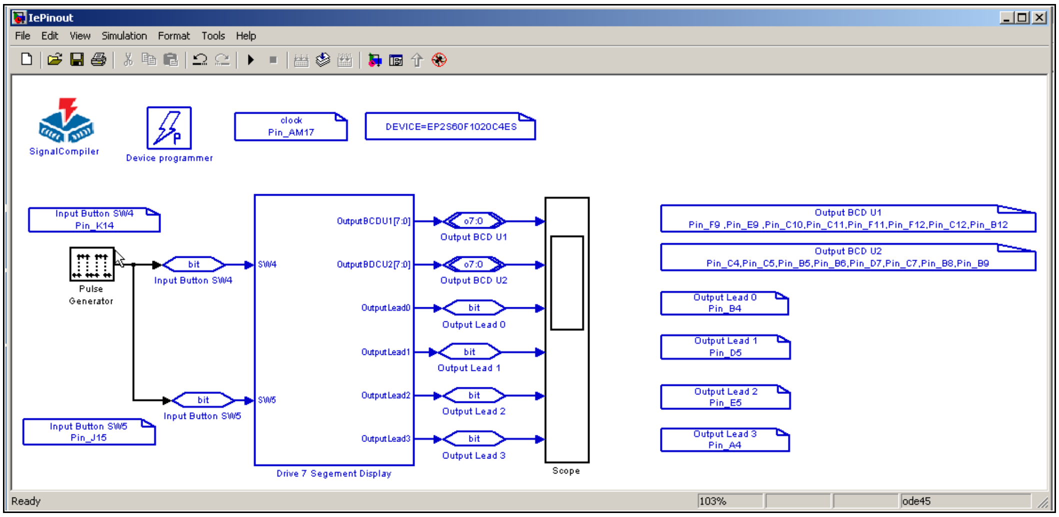
```
    pin name=sclr:pin value=Pin_B4
```

Table 1–6 shows the block parameters.

Table 1–6. Quartus II Pinout Assignment 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 DSP Builder Simulink library folder, as shown in Figure 1–8.
Pin Location	String	Pin location value of the FPGA IO. See Quartus II Help for the pinout values of a given device.

Figure 1–8. Using the Quartus II Pinout Assignment Block



SignalCompiler Block

The SignalCompiler block is the heart of DSP Builder, and performs the following functions:

- Converts your Simulink design into synthesizable RTL VHDL
- Generates VHDL testbenches
- Generates Verilog testbenches
- Generates Verilog HDL simulation model (via the Quartus® II software)
- Exports Simulink stimulus into a VHDL or Verilog HDL testbench and produces the expected response in an ASCII (**.txt**) file
- Generates Tcl scripts for Quartus II compilation
- Generates Tcl scripts for the LeonardoSpectrum™, Precision RTL, Synplify, and ModelSim® software
- Generates a vector file (**.vec**) for Quartus II simulation
- Generates a PTF configuration file, which you can use to import the design automatically into the SOPC Builder
- Enables generation of a SignalTap II (**.stp**) file
- Generates Quartus II Block Symbol Format (**.bsf**) file

For more information on the SOPC Builder, see the Products > Design Software section of the Altera wev site (www.altera.com). Under Sesign Software, select the SOPC Builder link.

You can run SignalCompiler at the MATLAB command prompt using the command `sc_altera` or by double clicking on the SignalCompiler block in your model. You can use SignalCompiler to control your design flow for synthesis, compilation, and simulation. DSP Builder supports the following flows:

- Synthesis and compilation:
  - *Automated Flow*—The automated flow allows you to control the entire design process from within the MATLAB/Simulink environment using the SignalCompiler block. With this flow, SignalCompiler creates RTL HDL design files and Tcl scripts in your working directory. Then, SignalCompiler executes the Tcl scripts to synthesize the design in the Quartus II, LeonardoSpectrum, or Synplify software and compile it in the Quartus II software. The results of the synthesis and compilation are displayed in the SignalCompiler **Messages** box. You can also use the automated flow to download your design into supported development boards.
  - *Manual Flow*—With the manual flow, you use SignalCompiler to output VHDL files and Tcl scripts; however, you do not use the scripts to run synthesis or compilation tools. After you choose to generate VHDL, SignalCompiler generates the RTL HDL design files and Tcl scripts in your working directory. You can then use

the VHDL files to synthesize the design in your tool of choice. You can compile the synthesized results in the Quartus II software. You should use the manual flow if you want to make your own synthesis or compilation settings.

- Simulation—The **Generate Stimuli for VHDL Testbench** option (**Testbench** tab) in the SignalCompiler block (**Testbench** tab), causes SignalCompiler to generate a VHDL testbench and ModelSim Tcl script for your model as well as a Vector File (**.vec**) for Quartus II simulation. You can use the testbench and Tcl script with the ModelSim® software, you can use the testbench in another simulation tool, or you can use the Vector File with the Quartus II software. For Verilog HDL simulation, turn on the **Generate Quartus II Verilog Output File (.vo)** option on the Verilog tab, and click **1 - Convert MDL to VHDL**, the **SignalCompiler** block generates a Verilog testbench `tb_`*`<design_name>`*`.v`, a ModelSim Tcl script for a Verilog simulation flow `tb_vo_`*`<design_name>`*`.tcl`, and a Verilog model output file *`<design_name>`*`.vo`. This option is located on the Verilog tab.

Table 1–7 shows the parameters for the SignalCompiler analyzer.

Table 1–7. SignalCompiler Page 1 of 2

Name	Value	Description
Re-run update diagram to solve workspace parameters	On or Off	When this option is turned on, SignalCompiler performs a simulation update command on the design and then extracts DSP Builder block information, such as sampling rate.
Analyze	–	When you click this button, SignalCompiler reads the current MDL file and detects hierarchy level and sample period of all DSP Builder blocks. You must analyze the design every time you modify it.
Skip Analysis	–	Click this button to bypass the analysis, that is, to change the design.
Hierarchical VHDL Entity names are unique	On or off	This option is available when the model contains more than one HDL Subsystem. This option avoids potential namespace conflicts in the VHDL model generated by SignalCompiler. SignalCompiler converts each HDL Subsystem into a VHDL file such that: ● When turned on, each VHDL entity name is unique: *<HDL Subsystem name> <#instance value>* that is, `subsystem1.vhd`, `subsystem2.vhd`. ● When turned off, each VHDL entity inherits the HDL Subsystem name

Table 1–8 shows the parameters for the SignalCompiler block.

Table 1–8. SignalCompiler Block Parameters Page 1 of 2 (Part 1 of 2)

Name	Value	Description
Device	Stratix II, Stratix, Stratix GX, APEX 20KC, APEX™ 20KE, APEX II, Mercury™, ACEX® 1K, FLEX 10K®, FLEX® 6000, Cyclone Cyclone II, Development Board	Indicate which Altera device family you want to target. You can also target the Stratix or Stratix II development boards. If you are using the automated design flow, the Quartus II software chooses the smallest device in which your design fits. The SignalCompiler block extracts device family information from the library block for the DSP development board you are using. For more information see Chapter 5, Boards Library.
Synthesis Tool	Quartus II, LeonardoSpectrum, Precision RTL, Synplify	Indicate which synthesis tool you want to use. SignalCompiler generate the following tcl script files based on your selection: <*design_name*>**_leo.tcl** for LeonardoSpectrum software, <*design_name*>**_precision.tcl** for Precision software, <*design_name*>**_spl.tcl** for Synplicity software, <*design_name*>**_quartus.tcl** for Quartus II software
Optimization	Area, Speed, Balanced, Fast Fit - No timing optimization or Use Current Quartus II Project	Indicate Quartus II optimization options during the mapping and fitting stages. When the Optimization is set to "Use Current Quartus II project," SignalCompiler does not create a new Quartus II project and compiles the model using the Quartus II option of an existing Quartus II project.
Select the Top-Level model file	User Defined	Click the icon to browse to the location of your top-level model file (**.mdl**). Specifying the file sets your working directory.
Generate Stimuli for VHDL Testbench	On or Off	To perform simulation in a third-party simulator, turn on this option and then run your simulation in Simulink. SignalCompiler outputs files for use with ModelSim or other simulators. This option is located on the **Testbench** tab. If this option is turned on, the Simulink simulation runs more slowly than if it is turned off. Therefore, you should only turn on this option when you wish to generate files for use with ModelSim or other simulators.
Period (ns)	User Defined	Specify the clock period. This box is located on the **Clock** tab.
Global Reset	Active High, Active Low	This option is located on the **Reset** tab. This option sets the polarity of the DSP Builder design's global reset circuitry.

Table 1–8. SignalCompiler Block Parameters Page 1 of 2 (Part 2 of 2)

Name	Value	Description
Connect to Ground	On or Off	Indicate whether you want to connect the reset signal to ground. This option is located on the **Reset** tab.
Insert SignalTap II Logic Analyzer	On or Off	Indicate whether you want to insert an instance of the SignalTap II logic analyzer into the design. You can use the logic analyzer to capture and probe signals on the DSP development boards. This option is located on the **SignalTap II** tab and only applies if your are targeting the DSP development boards.
Generate PTF SOPC File	On or Off	When this option is turned on, SignalCompiler generates a **class.ptf** configuration file. With this file, the DSP Builder design is a plug-and-play peripheral for use with SOPC Builder. This option is located on the **SOPC Info** tab.
Convert MDL to VHDL creates a Verilog HDL testbench and model	On or Off	When you turn on this option and click **Convert MDL to VHDL**, SignalCompiler generates a Verilog model output file <*design_name*>**.vo**, a Verilog testbench **tb_**<*design_name*>**.v**, and a ModelSim Tcl script for a Verilog simulation flow **tb_vo_**<*design_name*>**.tcl**. This option is located on the **Verilog** tab.
1 - Convert MDL to VHDL	–	Click this button to generate VHDL design files and Tcl scripts for your design.
2 - Synthesize	–	Click this button to synthesize the generated VHDL in the synthesis tool that you selected.
3 - Quartus II Fitter	–	Click this button to compile the design in the Quartus II software.
Execute steps 1, 2 and 3	–	Click this button to: 1 – Convert MDL to VHDL 2 – Synthesize VHDL 3 – Compile in the Quartus II software.
4 - Program Device	–	If you selected one of the DSP development boards from the **Device** list, this button is enabled. Click this button to download your design into the targeted board. See the *Stratix II DSP Development Kit Getting Started User Guide* or the *DSP Development Kit, Stratix & Stratix Professional Edition Getting Started User Guide* for information on setting up the board and connecting it to your PC.
Project Info	–	Click this button to view the paths to the Synplify, LeonardoSpectrum, Precision RTL, the Quartus II software, or your working directory.
Report File	–	Click this button to view the SignalCompiler report file.

See the following topics for additional information:

- "Fixed-Point Notation" on page 1–2
- "SignalCompiler Data Width Propagation" on page 1–17
- "SignalCompiler Clock Assignment" on page 1–22

SignalCompiler Data Width Propagation

During the Simulink-to-VHDL conversion, SignalCompiler assigns a bit width to all of the Altera blocks in the design. You can specify the bit width of an Altera block in the Simulink design. However, you do not need to specify the bit width for all blocks. SignalCompiler propagates the bit width from the source of a data path to its destination. Each DSP Builder block has specific bit-width growth rules, which are described in the documentation for each block. You can optionally specify the bit width of intermediate blocks.

The following design example (Figure 1–9) **fir3tapsub.mdl**, illustrates bit-width propagation. The **fir3tapsub.mdl** design is a 3-tap finite impulse response (FIR) filter. See Figure 1–9.

The design 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 are used to build the tapped delay line
- The coefficient values are {1.0000, -5.0000, 1.0000}. A Gain block performs the coefficient multiplication

Figure 1–9. 3-Tap FIR Filter

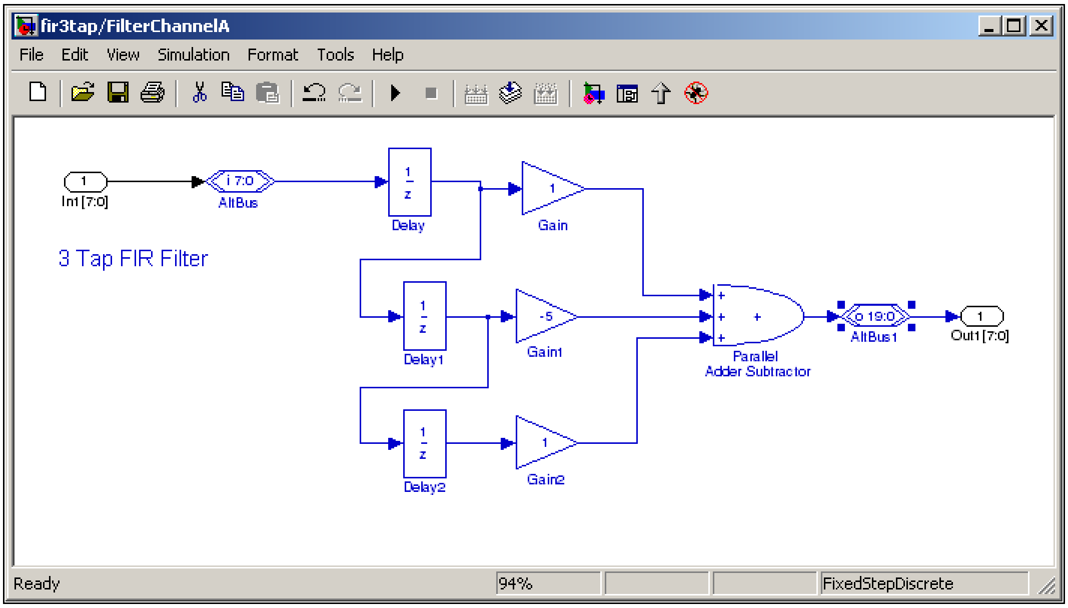
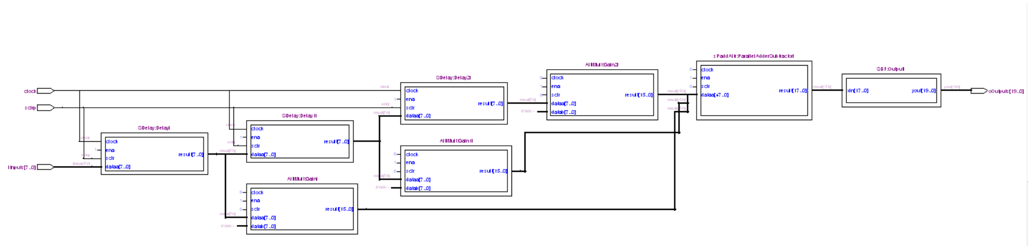


Figure 1–10 shows the RTL representation of **fir3tapsub.mdl** created by SignalCompiler.

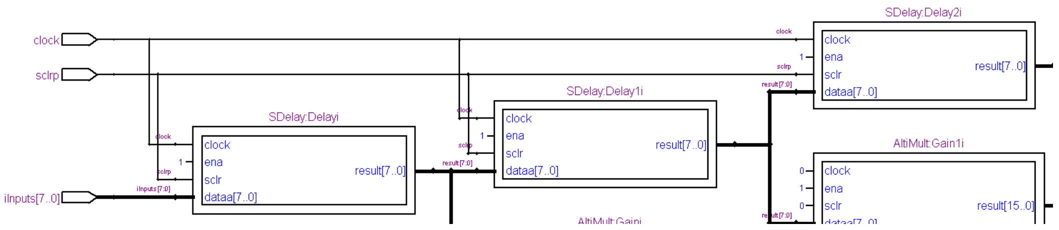
Figure 1–10. 3-Tap FIR Filter in Quartus II RTL View



Tapped Delay Line

The bit width propagation mechanism starts at the source of the data path, in this case at the AltBus block iInputDatas, which is an 8-bit input bus. This bus feeds the register `U0`, which feeds `U1`, which feeds `U2`. SignalCompiler propagates the 8-bit bus in this register chain where each register is eight bits wide. See Figure 1–11.

Figure 1–11. Tap Delay Line in Quartus II Version RTL Viewer



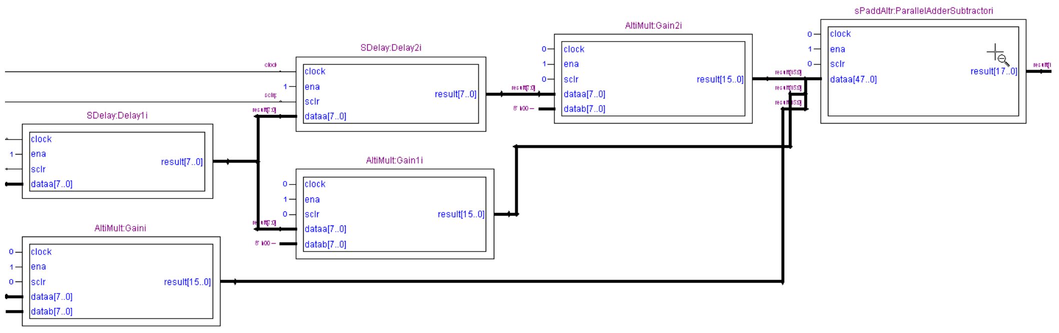
Arithmetic Operation

Figure 1–12 shows the arithmetic section of the filter, which 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 1–12. 3-Tap FIR Filter Arithmetic Operation in Quartus II Version RTL Viewer



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

- Multiplying $a \times b$ 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*]))

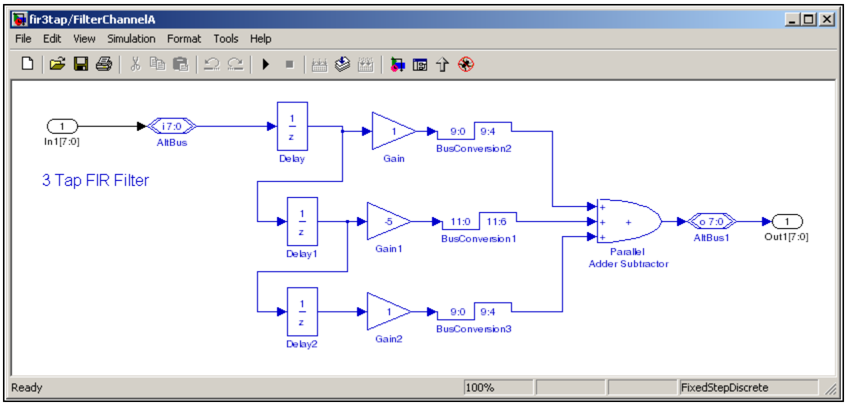
The parallel adder has three input buses of 14, 16, and 14 bits. To perform this addition in binary, SignalCompiler automatically sign extends the 14 bit buses to 16 bits. The output bit width of the parallel adder is 18 bits, which covers the full resolution.

There are several options that can change the internal bit width resolution and therefore change the size of the hardware required to perform the function described in Simulink:

- Change the bit width of the input data.
- Change the bit width of the output data. The VHDL synthesis tool will remove the unused logic.
- Insert BusConversion blocks to change the internal signal bit width.

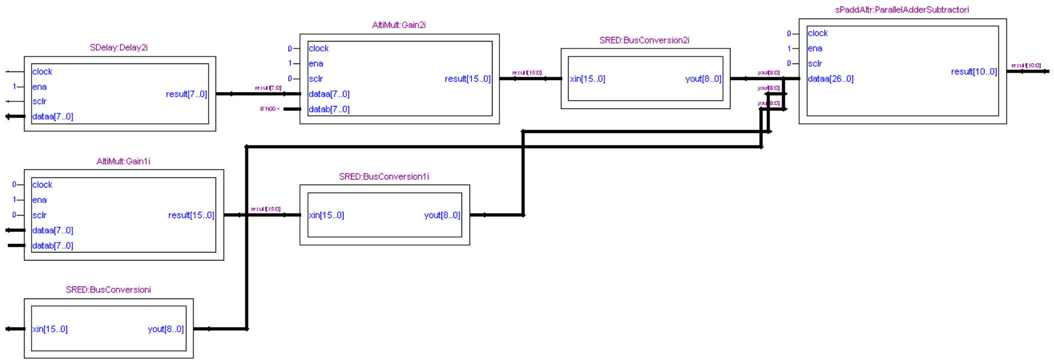
Figure 1–13 shows how AltBus blocks are used to control internal bit widths. In this example, the output of the Gain block has 4 bits removed (BusConversion format converts [11:0] to [11:3]). The scope displays the functional effect of this truncation on the coefficient values.

Figure 1–13. 3-Tap Filter with BusConversion to Control Bit Widths



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. See Figure 1–14.

Figure 1–14. 3-Tap Filter with BusConversion to Control Bit Widths in Quartus II RTL Viewer



See the following topics for additional information:

- "Fixed-Point Notation" on page 1–2
- "SignalCompiler Data Width Propagation" on page 1–17
- "SignalCompiler Clock Assignment" on page 1–22

SignalCompiler Clock Assignment

As described in "Frequency Design Rule" on page 1–5, SignalCompiler 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 the design does not contain a **PLL** block or **ClockAltr** block, the SignalCompiler implicitly connects all of the registered DSP Builder blocks' clock pins to a single clock domain (signal 'clock' in VHDL). When the design contains the PLL block and the detected Simulink sampling period of the registered DSP Builder blocks is equal to one of the output clock periods defined in the PLL block, SignalCompiler implicitly connects all of the registered blocks' clock pins to one of the PLL's output clocks, otherwise SignalCompiler returns an error. When the design contains one or more ClockAltr blocks and the detected Simulink sampling period of the registered DSP Builder blocks is equal to one of the output clock periods defined in one of the ClockAltr blocks, SignalCompiler implicitly connects all of the registered blocks' clock pins to one of the ClockAltr blocks or returns an error.

From a clocking standpoint, DSP Builder blocks fall into two categories:

- *Combinatorial 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 1–15 illustrates DSP Builder block combinatorial behavior. The Magnitude block translates as a combinatorial signal in VHDL. SignalCompiler does not add clock pins to this function.

Figure 1–15. Magnitude Block: Combinatorial Behavior

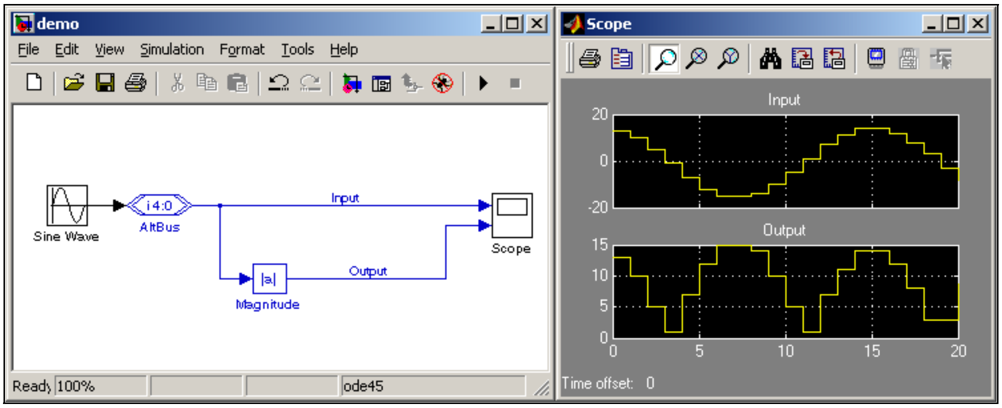
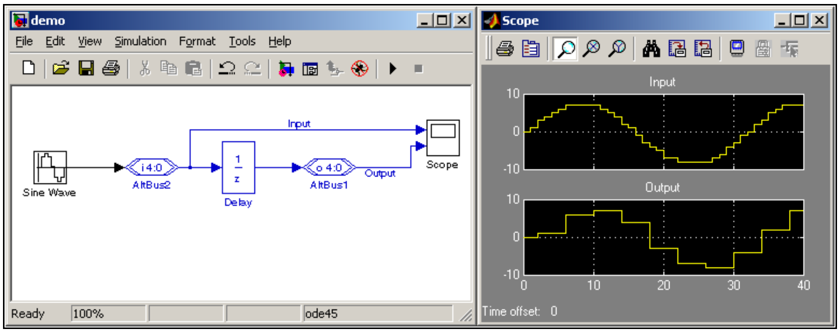


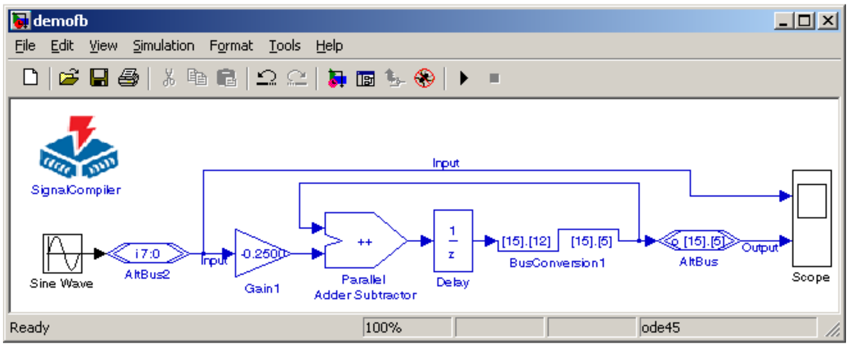
Figure 1–16 illustrates the behavior of a registered DSP block. In the VHDL netlist, SignalCompiler adds clock pin inputs to this function. The Delay block, with the `clock phase selection` parameter equal to 100, is converted into a VHDL shift register with a decimation of three and an initial value of zero.

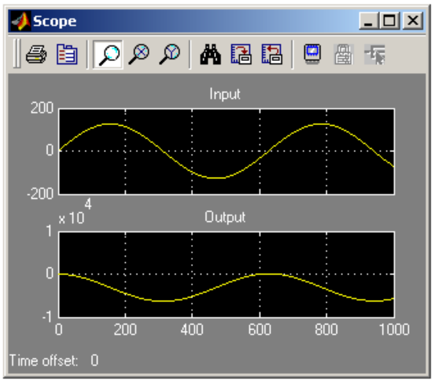
Figure 1–16. Delay Block: Registered Behavior



For feedback circuitry (that is, the output of a block fed back into the input of a block), a registered block must be in the feedback loop. Otherwise, an unresolved combinatorial loop is created. See Figure 1–17.

Figure 1–17. Feedback Loop





You can design multi-rate designs by using the PLL block and assigning different sampling periods on registered DSP Builder blocks. Alternatively, you can design multi-rate designs without the DSP Builder PLL block by using a single clock domain and the following design rules. For a multi-rate design using a single clock domain with clock enable:

- The fastest sample rate is an integer multiple of the slower sample rates. The values are specified in the **Clock Phase Selection** box in the **Delay** dialog box.

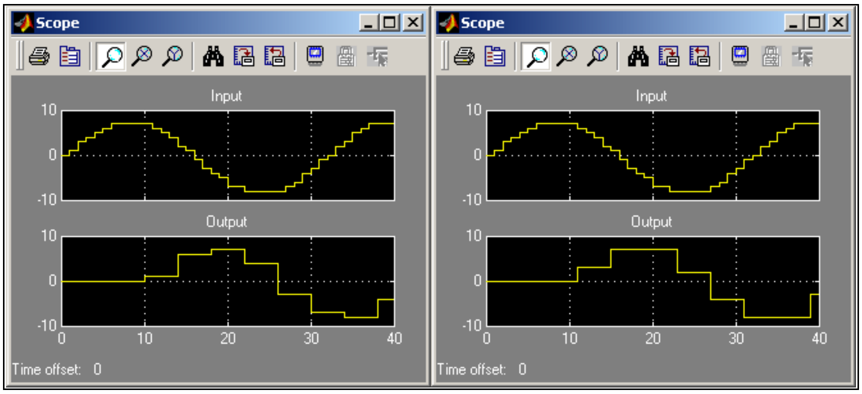
- The **Clock Phase Selection** box accepts a binary pattern string to describe the clock phase selection. Each digit or bit of this string is processed sequentially on every cycle of the fastest clock. When a bit is equal to one, the block is enabled; when a bit is equal to zero, the block is disabled. For example, see Table 1–9.

Table 1–9. 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 second phase out of 4 and only the second data out of 4 (sampled at the rate 1) passes through. In other words, the data on phases 1, 3, and 4 do not pass through the Delay block.

Figure 1–18 compares the scopes for the Delay block operating at a one quarter rate on the `1000` and `0100` phases, respectively.

Figure 1–18. 1000 as Opposed to 0100 Phase Delay



See the following topics for additional information:

- "Fixed-Point Notation" on page 1–2
- "SignalCompiler Data Width Propagation" on page 1–17
- "SignalCompiler Clock Assignment" on page 1–22

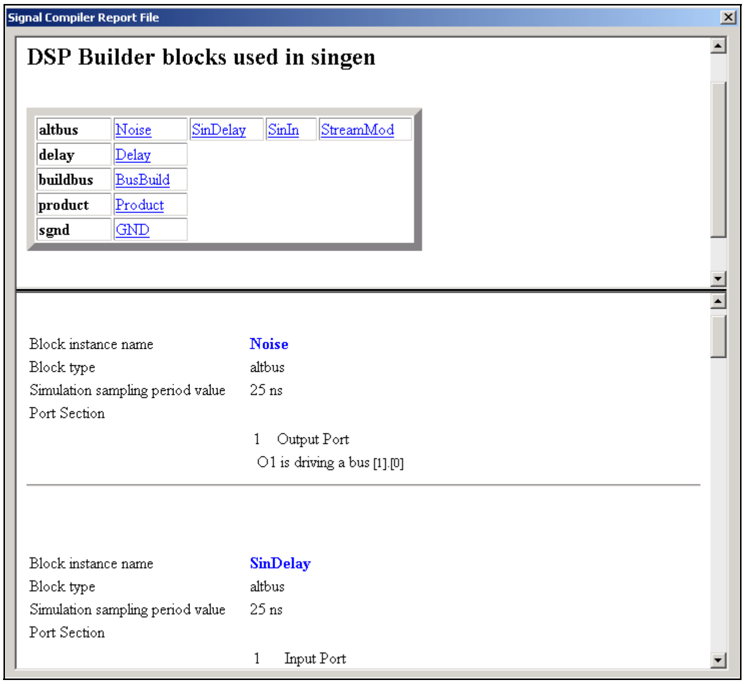
DSP Builder Report File

After generating VHDL, SignalCompiler outputs a report file that lists SignalCompiler block parameters, the files generated by SignalCompiler, links to the synthesis log file and the Quartus II Report File, and the following information for each VHDL entity:

- SignalCompiler settings
- Signal width mismatches
- Out-of-range signals
- Detected Simulink sampling period
- Block port bit width information for the entity
- Number of HDL SubSystem instances used (hierarchy information)

The report file shows how SignalCompiler propagates the bit widths to all of the design blocks at each level of hierarchy. Figures 1–19 and 1–20 show an example DSP Builder report file.

Figure 1–19. DSP Builder Block Report File



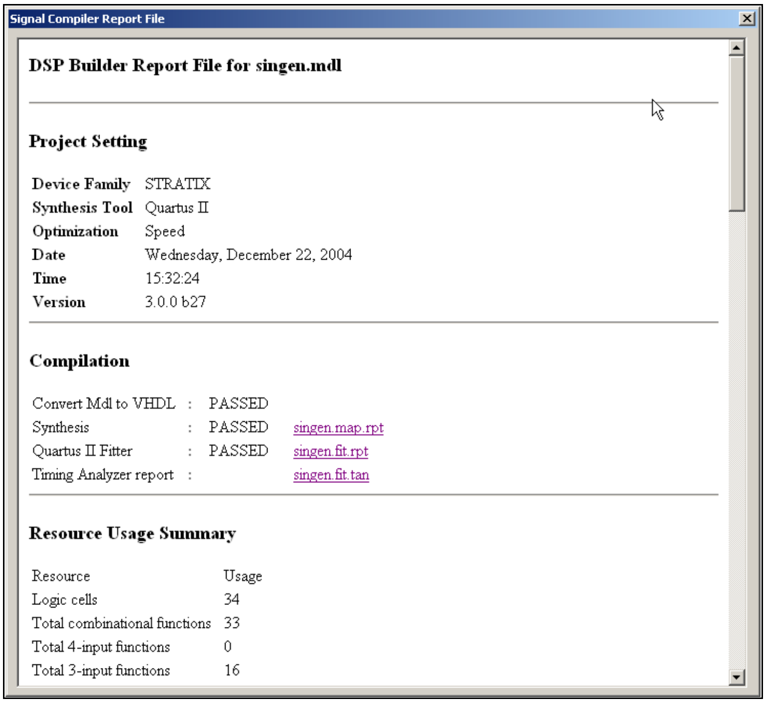
Signal Compiler Report File

DSP Builder blocks used in singen

altbus	Noise	SinDelay	SinIn	StreamMod
delay	Delay			
buildbus	BusBuild			
product	Product			
sgnd	GND			

```
Block instance name                 Noise
Block type                          altbus
Simulation sampling period value    25 ns
Port Section
                                     1    Output Port
                                     O1 is driving a bus [1].[0]

Block instance name                 SinDelay
Block type                          altbus
Simulation sampling period value    25 ns
Port Section
                                     1    Input Port
```

Figure 1–20. DSP Builder Block Report File



Signal Compiler Report File

DSP Builder Report File for singen.mdl

Project Setting

Device Family	STRATIX
Synthesis Tool	Quartus II
Optimization	Speed
Date	Wednesday, December 22, 2004
Time	15:32:24
Version	3.0.0 b27

Compilation

Convert Mdl to VHDL	:	PASSED	
Synthesis	:	PASSED	singen.map.rpt
Quartus II Fitter	:	PASSED	singen.fit.rpt
Timing Analyzer report	:		singen.fit.tan

Resource Usage Summary

Resource	Usage
Logic cells	34
Total combinational functions	33
Total 4-input functions	0
Total 3-input functions	16

SignalTap II Analysis Block

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 methods of system verification with efficient board-level verification. DSP Builder version 2.0 and higher 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.

DSP Builder includes the SignalTap II Analysis block with which you can set up event triggers, configure memory, and display captured waveforms (you use the 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 using the USB-Blaster™ or ByteBlasterMV™ 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 work space as a global variable.

DSP Builder and the SignalTap II Analysis block create a SignalTap II embedded logic analyzer that:

- 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

You can also use the Quartus II software to insert an instance of the SignalTap II embedded logic analyzer in your design. The Quartus II software supports additional features such as analyzing nodes in all levels of the design hierarchy, using multiple clock domains, and adjusting what percentage of data is captured around the trigger point.

For more information on using the SignalTap II embedded logic analyzer with the Quartus II software, see the Quartus II Help.

SignalTap II Design Flow

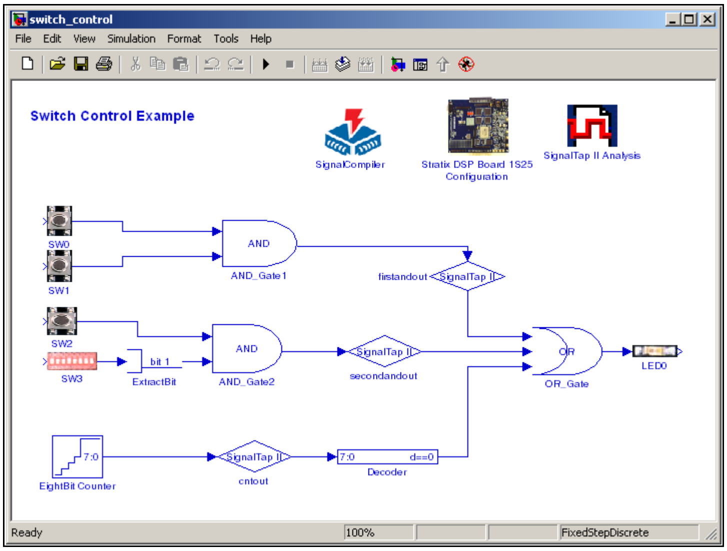
Working with the SignalTap II embedded logic analyzer and DSP Builder involves the following flow:

1. Add a SignalTap II Analysis block to your Simulink design.

☞ You cannot open the SignalTap Analyzer block unless you have generated VHDL by clicking **1 - Convert MDL to VHDL** in SignalCompiler.

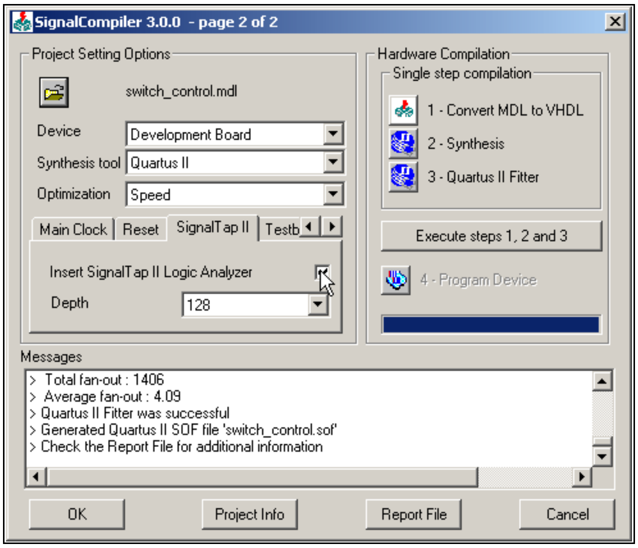
2. Specify the signals (nodes) that you want to analyze by inserting Node blocks. Figure 1–21 shows an example design.

Figure 1–21. Example SignalTap II Analysis Model



3. Turn on the **Insert SignalTap Logic Analyzer** option in the SignalTap II tab of SignalCompiler. See Figure 1–22.

Figure 1–22. SignalTap II Tab in SignalCompiler



SignalCompiler 3.0.0 - page 2 of 2

Project Setting Options

switch_control.mdl

Device: Development Board

Synthesis tool: Quartus II

Optimization: Speed

Main Clock | Reset | SignalTap II | Testb

Insert SignalTap II Logic Analyzer [✓]

Depth: 128

Hardware Compilation

Single step compilation

1 - Convert MDL to VHDL

2 - Synthesis

3 - Quartus II Fitter

Execute steps 1, 2 and 3

4 - Program Device

Messages

```
> Total fan-out : 1406
> Average fan-out : 4.09
> Quartus II Fitter was successful
> Generated Quartus II SOF file 'switch_control.sof'
> Check the Report File for additional information
```

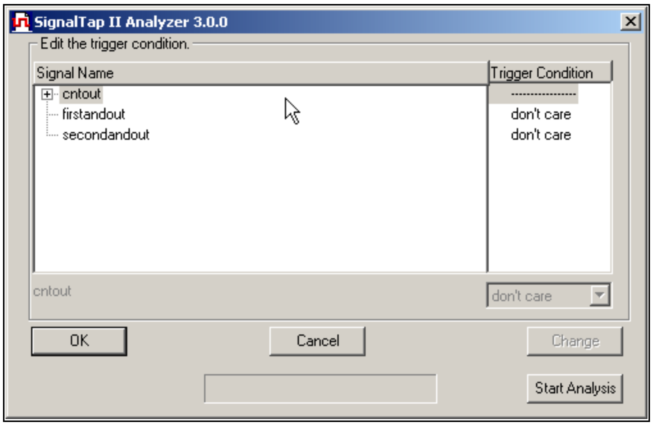
OK | Project Info | Report File | Cancel

4. Target one of the DSP development board devices in SignalCompiler.

5. Using SignalCompiler, generate VHDL, synthesize it, perform Quartus II compilation, and download the design into the DSP development board (starter or professional).

6. Specify trigger conditions in the SignalTap II Analysis block. See Figure 1–23.

Figure 1–23. Specifying Trigger Conditions



Signal Name	Trigger Condition
cntout	----------------
firstandout	don't care
secondandout	don't care

cntout — don't care

7. Specify the radix for the signal groups and run the analysis. You can view the captured data as a waveform in two MATLAB plots. The first plot displays the signals in binary format. The second plot displays the signals in the radix you specified. See Figures 1–24 and 1–25.

   DSP Builder only supports the SignalTap II embedded logic analyzer with the ByteBlasterMV download cable.

Figure 1–24. Example SignalTap II Analysis MATLAB Plot

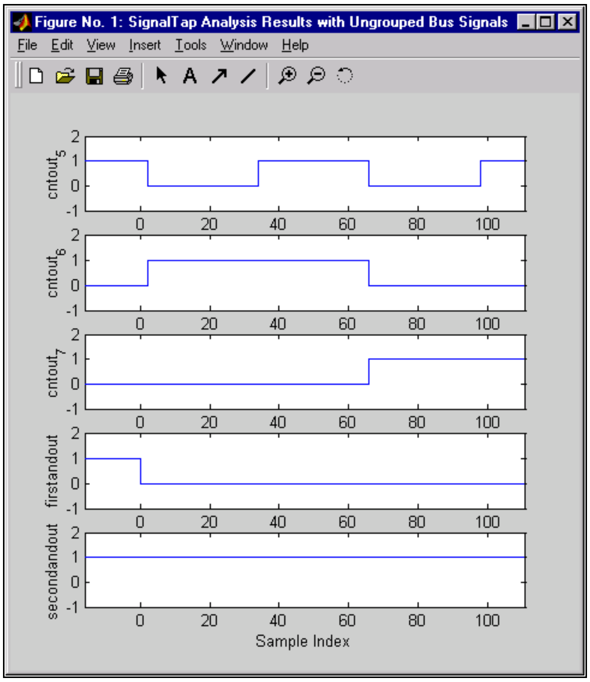
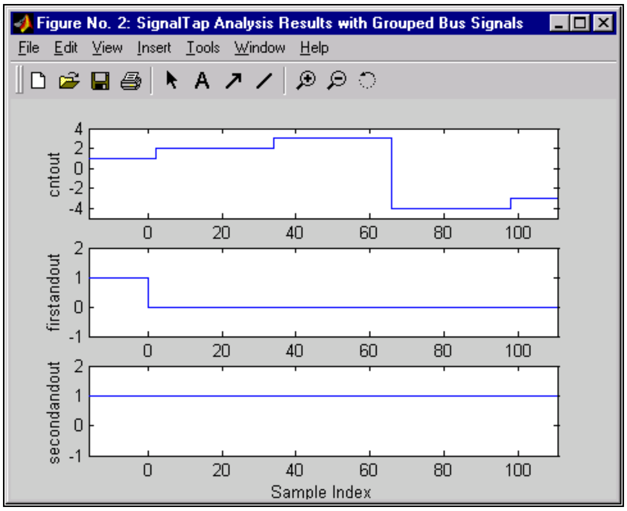


Figure 1–25. MATLAB Plot with User-Specified Radix



For detailed instructions on using the SignalTap II Analysis and SignalTap II blocks, see the *Perform SignalTap II Analysis* section in the *Performing SignalTap II Logic Analysis* chapter in the *DSP Builder User Guide*.

SignalTap II Nodes

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

When you implement the SignalTap II embedded logic analyzer using DSP Builder, you can only analyze signals in a top-level design file. You cannot probe signals within a subsystem.

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

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

SignalTap II Trigger Conditions

The trigger pattern describes a logic event in terms of logic levels and/or edges. It is a comparison register used by the SignalTap II embedded logic analyzer to recognize the moment when the input signals match the data specified in the trigger pattern.

The trigger pattern is composed of 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 embedded logic analyzer is triggered when it detects the trigger pattern on the input signals.

SubSystem Builder Block

The SubSystemBuilder block makes it easy for you to import a VHDL or Verilog HDL design's input and output signals into a Simulink subsystem. You can then add the rest of the design functionality to the subsystem (using DSP Builder blocks or MATLAB functions) and simulate it in Simulink. You may also treat the design as a black box.

The SubSystemBuilder 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 the value 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 the value 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 block, drag and drop it into your model, click **Select HDL File**, specify the file to import, and click the **Add** *<entity name>* **in** *<model name>* button.

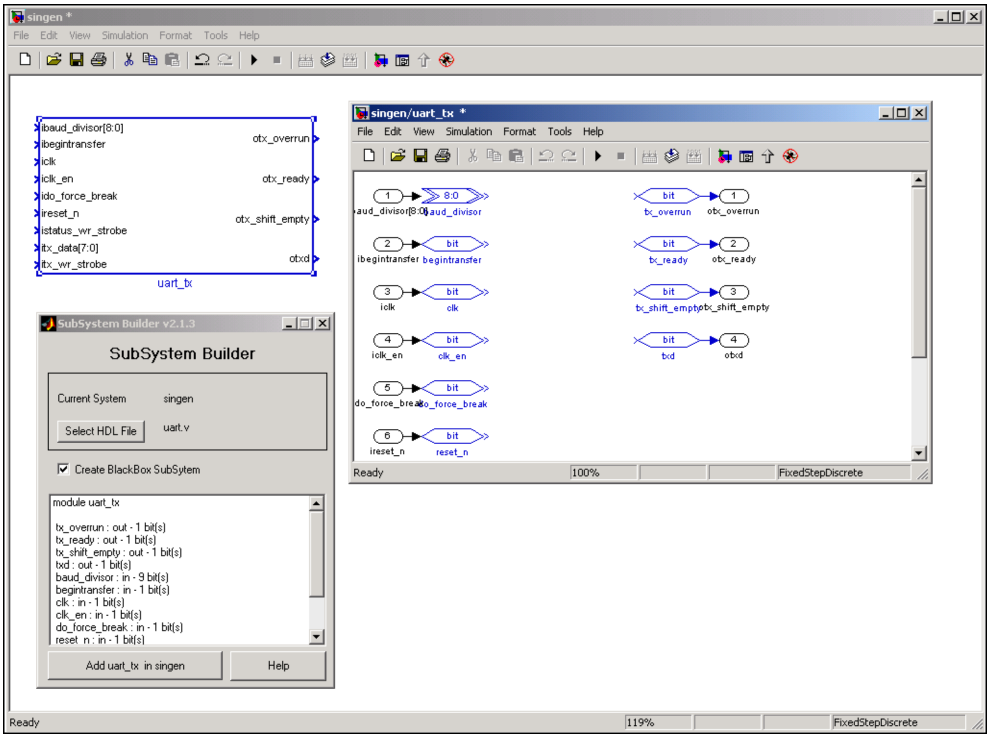
Table 1–10 shows the block parameters.

Table 1–10. SubSystem Builder Block Parameters

Name	Value	Description
Select HDL File	User defined	Use this button to specify the Verilog HDL file to import.
Create Black Box SubSystem	On or Off	Turn on this option if you want to treat the imported VHDL or Verilog HDL design as a black box. You must turn on this option if you are importing a Verilog HDL file.

Figure 1–26 shows an example using the SubSystemBuilder block.

Figure 1–26. SubSystemBuilder Block & Example



```
module uart_tx

tx_overrun : out - 1 bit(s)
tx_ready : out - 1 bit(s)
tx_shift_empty : out - 1 bit(s)
txd : out - 1 bit(s)
baud_divisor : in - 9 bit(s)
begintransfer : in - 1 bit(s)
clk : in - 1 bit(s)
clk_en : in - 1 bit(s)
do_force_break : in - 1 bit(s)
reset_n : in - 1 bit(s)
```

VCDSink Block

The VCDSink block is a sink block used to export Simulink signals to a third-party waveform viewer. When you run the simulation of your model, the VCDSink block generates a value change dump (**.vcd**) file named <*VCDSink block name*>**.vcd** which can be read by a third-party waveform viewer. The generated files are named after the VCDSink block that generated them.

To use the VCDSink block in your Simulink model, perform the following steps:

1. Add a VCDSink block to your Simulink model.
2. Connect the simulink signals you want to display in a third-party waveform viewer to the VCDSink block.
3. Run the Simulink simulation.
4. Read the VCD file in the third-party waveform viewer.

If you are using the ModelSim software, run the script <*VCDSink block name*>**_vcd.tcl**. This Tcl script converts VCD files into ModelSim waveform format (**.wlf**), starts the waveform viewer, and displays the signals.

Table 1–11 shows the parameters for the VCDSink block.

Table 1–11. VCDSink Block Parameters

Name	Value	Description
Number of Inputs	An integer greater than 0	Specify the number of input ports of the VCDSink block.
Input data type in VCD file	Signed, Unsigned, Single Bit	Specify the Input data type of all the input ports of the VCDSink block.

Figure 1–27 shows the VCDSink block used in a model with the Magnitude block. (For more information, see "Magnitude Block" on page 2–17.)

Figure 1–27. Simulink Model Using the VCDSink Block

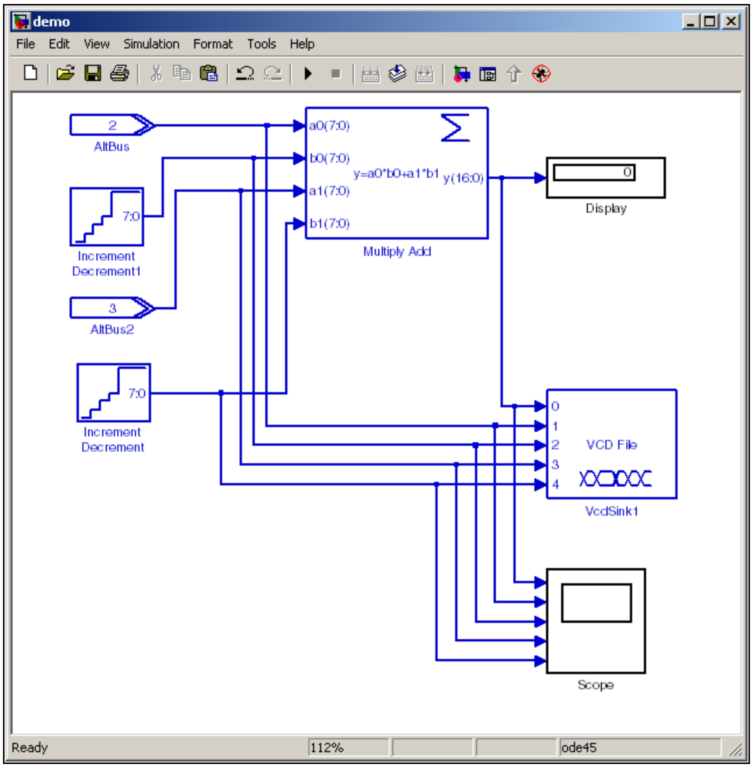
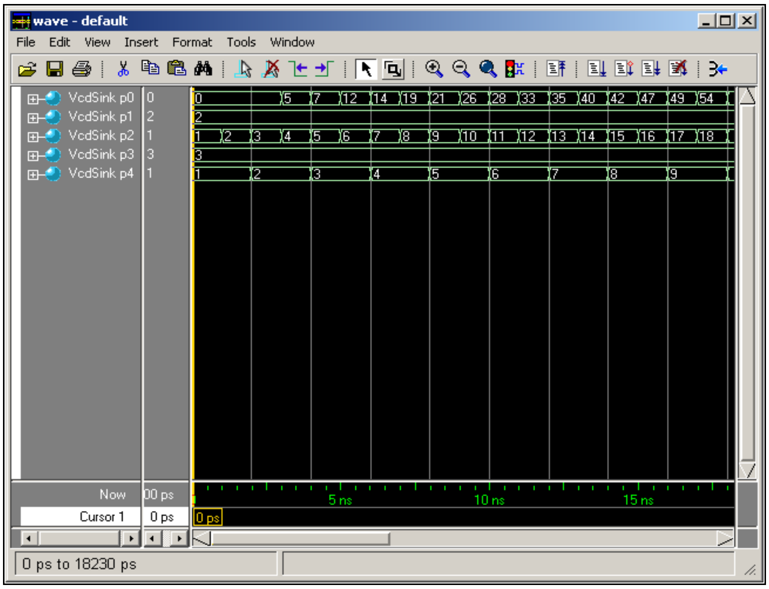


Figure 1–28 shows ModelSim displaying the waveforms generated by the design in Figure 1–27.

Figure 1–28. Output From VCDSink Block Displayed in ModelSim Simulator



Chapter 2. Arithmetic Library

Introduction

The Arithmetic library contains two's complement signed arithmetic blocks such as multipliers and adders. Some blocks have the **Use Dedicated Circuitry** option, which implements functionality into dedicated hardware in Altera® Stratix® II, Stratix, Stratix GX, Cyclone™ II (that is, in the dedicated DSP blocks of these devices.)

For more information on these device families, see the:

- *Stratix II Device Handbook*
- *Stratix Device Handbook*
- *Cyclone II Device Handbook*

Barrel Shifter Block

The Barrel Shifter block shifts the input data by the amount set by the input bus. The Barrel Shifter can shift data to the left (toward the MSB) or to the right (toward the LSB).

The Barrel Shifter can shift data to the left only, or to the right only, or both directions. For both directions, an additional input pin, `direction`, dynamically sets the direction of the shift. The shifting operation is an arithmetic shift and not a logical shift, i.e, the shifting operation preserves the input data sign. Table 2–1 shows the Barrel Shifter block parameters.

Table 2–1. Barrel Shifter Parameters (Part 1 of 2)

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Indicate the bus number format that you want to use for the counter.
[number of bits].[]	1 to 51	Select the number of bits to the left of the binary point.
[].[number of bits]	0 to 51	Select the number of bits to the right of the binary point for the gain. This option is zero (0) unless Signed Fractional is selected.
Shift Direction	Shift Left, Shift Right, or Use Direction Input Pin	Indicate which direction you would like to shift the bits.

Table 2–1. Barrel Shifter Parameters (Part 2 of 2)

Name	Value	Description
Pipeline	On or Off	Turn on this option to pipeline the barrel shifter.
Use Dedicated Circuitry	On or Off	If you are targeting Stratix II, Stratix, Stratix GX devices, turn on this option to implement the functionality in Stratix DSP blocks. If you are not targeting Stratix II, Stratix, or Stratix GX devices, the functionality is implemented in logic elements.

Table 2–2 shows the block I/O formats.

Table 2–2. Barrel Shifter Block I/O Formats Note (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 2–2:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Bit Level Sum of Products Block

The Bit Level Sum of Products block performs a sum of the multiplication of one-bit inputs by signed integer fixed coefficients. The block uses the equation:

$$q = a(0)C_0 + \ldots + a(i)C_i + \ldots + a(n)C_n$$

where:

- $a(i)$ is the one-bit input data
- C_i are the signed integer fixed coefficient
- n is the number of coefficients in the range two to eight

Table 2–3 shows the block parameters.

Table 2–3. Bit Level Sum of Products Block Parameters

Name	Value	Description
Number of Coefficients	2 to 8	Indicate the number of coefficients.
Coefficient Bit Width	2 to 51	Indicate the bit width as a signed integer.
Coefficient Value	User Defined	Indicate the coefficient values as signed integers.
Register Inputs	On or Off	When set to “On” add a register on the input signal.

Figure 2–1 shows an example using the Bit Level Sum of Products block. This example is a 32-tap fixed-coefficient filter and it uses a mixed 4- to 8-input lookup table for partial product pre-computation.

Table 2–4 shows the block I/O formats.

Table 2–4. Bit Level Sum of Products Block I/O Formats *Note (1)*

I/O	Simulink *(2), (3)*	VHDL	Type *(4)*
I	$I1_{[1].[0]}$ $Ii_{[1].[0]}$ ... $In_{[1].[0]}$	I1: in STD_LOGIC Ii: in STD_LOGIC ... In: in STD_LOGIC	Explicit
O	$O1_{[L0].[0]}$	O1: out STD_LOGIC_VECTOR({L0 - 1} DOWNTO 0	Explicit

Notes to Table 2–4:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

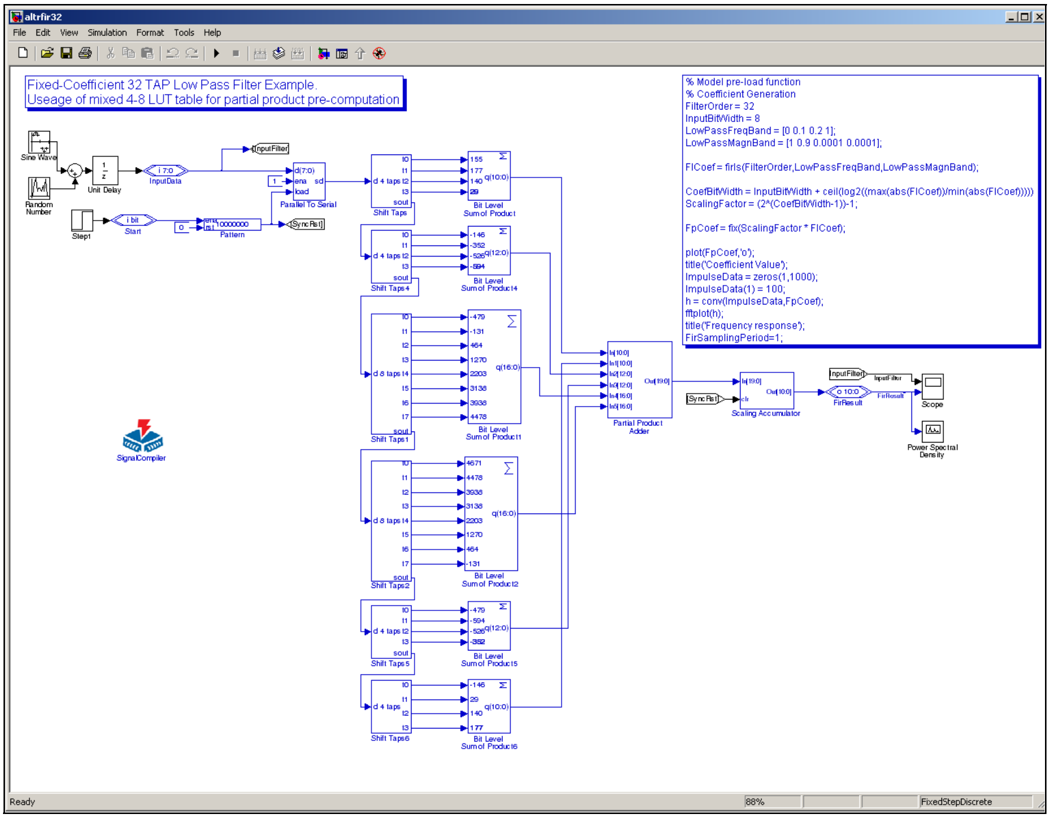
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–1 shows an the Bit Level Sum of Products block in an example design.

Figure 2–1. Bit Level Sum of Products Block Example



```
% Model pre-load function
% Coefficient Generation
FilterOrder = 32
InputBitWidth = 8
LowPassFreqBand = [0 0.1 0.2 1];
LowPassMagnBand = [1 0.9 0.0001 0.0001];

FICoef = firls(FilterOrder,LowPassFreqBand,LowPassMagnBand);

CoefBitWidth = InputBitWidth + ceil(log2((max(abs(FICoef))/min(abs(FICoef)))))
ScalingFactor = (2^(CoefBitWidth-1))-1;

FpCoef = fix(ScalingFactor * FICoef);

plot(FpCoef,'o');
title('Coefficient Value');
ImpulseData = zeros(1,1000);
ImpulseData(1) = 100;
h = conv(ImpulseData,FpCoef);
fftplot(h);
title('Frequency response');
FirSamplingPeriod=1;
```

Comparator Block

The Comparator block compares two Simulink signals and returns a single bit. The block implicitly understands the input data type (for example, signed binary or unsigned integer) and produces a single-bit output. Table 2–5 shows the block parameters.

Table 2–5. Comparator Block Parameters

Name	Value	Description
Operator	a == b, a ~= b, a < b, a <= b, a >= b, a > b	Indicate which operation you wish to perform on the two buses.

Table 2–6 shows the block I/O formats.

Table 2–6. Comparator Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L1].[R1]}$ $I2_{[L2].[R2]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L2 + R2 - 1} DOWNTO 0)	Implicit Implicit
O	$O1_{[1]}$	O1: out STD_LOGIC	Implicit

Notes to Table 2–6:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

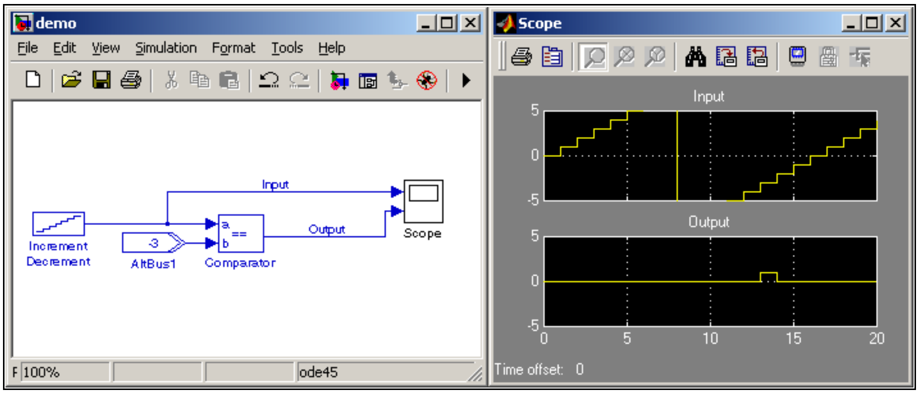
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–2 shows an example using the Comparator block.

Figure 2–2. Comparator Block Example



Counter Block

The Counter block is an up/down counter. Table 2–7 shows the block parameters.

Table 2–7. Counter Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Indicate the bus number format that you want to use for the counter.
[number of bits].[]	1 to 51	Select the number of bits to the left of the binary point.
[].[number of bits]	0 to 51	Select the number of bits to the right of the binary point for the gain. This option is zero (0) unless Signed Fractional is selected.
Count Modulo	User defined	Indicates the maximum count plus 1. Number of unique states in the counter's cycle.

Table 2–8 shows the block I/O formats.

Table 2–8. Divider Block I/O Formats *Note (1)*

I/O	Simulink *(2)*, *(3)*	VHDL	Type *(4)*
I	I1$_{[L].[R]]}$	I1: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Explicit
O	O1$_{[L].[R]}$	O1: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Explicit

Notes to Table 2–8:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Differentiator Block

The Differentiator block is a signed integer differentiator with the equation q = d - q. The block can be used for DSP functions such as CIC filters. Table 2–9 shows the block parameters.

Table 2–9. Differentiator Block Parameters

Name	Value	Description
Number of Bits	2 - 51	Indicate the number of bits.
Use Control Input	On or Off	Indicate whether you would like to use additional control inputs (clock enable and reset).

Table 2–10 shows the block I/O formats.

Table 2–10. Differentiator Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L1].[0]}$	I1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0)	Explicit
O	$O1_{[L1].[0]}$	O1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0)	Explicit

Notes to Table 2–10:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Divider Block

The Divider block takes a numerator and a denominator and computes a quotient and a remainder. The bit-width format of the numerator, denominator, quotient, and reminder are identical. Table 2–11 shows the block parameters.

Table 2–11. Divider Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Indicate the bus number format that you want to use for the divider.
[number of bits].[]	1 to 51	Select the number of bits to the left of the binary point.
[].[number of bits]	0 to 51	Select the number of bits to the right of the binary point for the gain. This option is only available when Signed Fractional is selected.
Pipeline	0 - 25	When non-nil, adds pipeline levels to increase the data throughput.

Table 2–12 shows the block I/O formats.

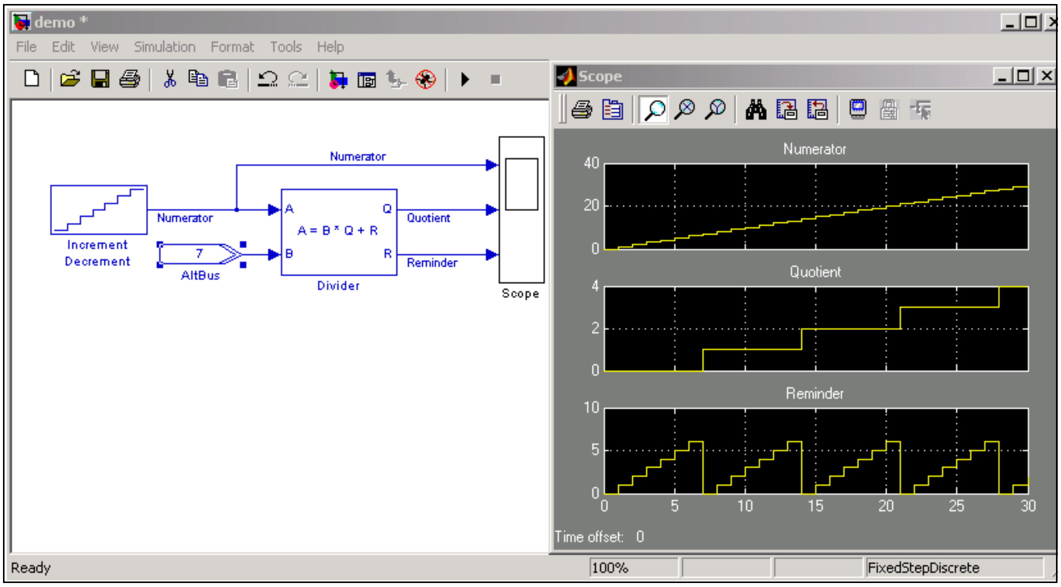
Table 2–12. Divider Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L].[R]}$ $I2_{[L].[R]}$	I1: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Explicit Explicit
O	$O1_{[L].[R]}$ $O2_{[L].[R]}$	O1: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0) O2: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Explicit Explicit

Notes to Table 2–12:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–3 shows an example using the Divider block.

Figure 2–3. Divider Block Example



Gain Block

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 signal input and gain must be scalars.

☞ The Simulink software also provides a Gain block. If you use the Simulink Gain block in your model, you can only use it for simulation; SignalCompiler cannot convert it to HDL.

Table 2–13 shows the block parameters.

Table 2–13. Gain Block Parameters

Name	Value	Description
Gain Value	User Defined	Indicate the gain value you want to use as 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	Indicate the bus number format you want to use for the gain value.
[Gain value number of bits].[]	1 - 51	Select the number of bits to the left of the binary point, including the sign bit for the gain.
[].[Gain value number of bits]	0 - 51	Select the number of bits to the right of the binary point for the gain. This option is only available when **Signed Fractional** is selected.
Number of Pipeline Levels	0 - 4	Specify the pipeline delay.
Use LPM	On or Off	This parameter is used for synthesis. When the **Use LPM** option is turned 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.
Use Control Inputs	On or Off	Indicate whether you would like to use additional control inputs (clock enable and reset). This option is only available if the **Number of Pipeline Levels** setting is greater than 1.
Clock Phase Selection	User Defined	Phase selection. This option is only available if the **Number of Pipeline Levels** setting is greater than 1. 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 out of 4 and only the second data out of 4 (sampled at the rate 1) passes through. In other words, the data on phases 1, 3, and 4 do not pass through the delay block.

Table 2–14 shows the block I/O formats.

Table 2–14. Gain Block I/O Formats Note (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	$O1_{[L1 + LK].2*max(R1,RK)]}$ where K is the gain constant with the format $K_{[LK].[RK]}$	O1: out STD_LOGIC_VECTOR({L1 + LK + 2*max(R1,RK) - 1} DOWNTO 0)	Implicit

Notes to Table 2–14:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

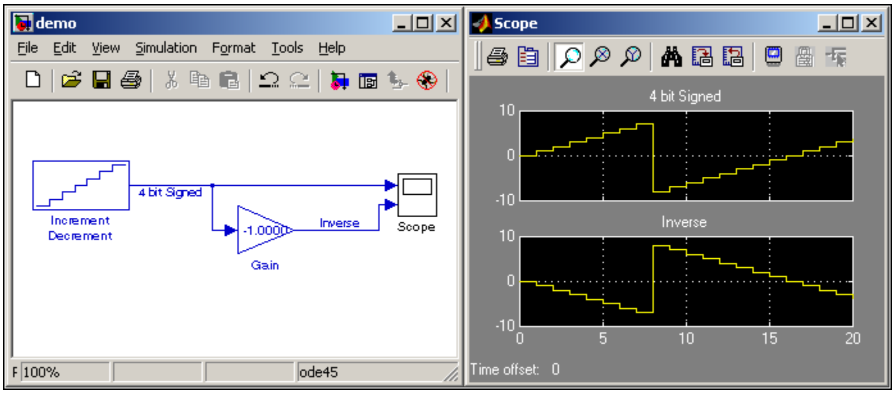
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–4 shows an example using the Gain block.

Figure 2–4. Gain Block Example



Increment Decrement Block

The Increment Decrement block generates a counting sequence in time. The output can be a signed integer, unsigned integer, or signed binary fractional number. For all number formats, the counting sequence increments/decrements the LSB bit by one. Table 2–15 shows the block parameters. The block has a clock phase selection control that operates as described in Table 2–15.

Table 2–15. Increment Decrement Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Choose the number format you wish to use for the bus.
<number of bits>.[]	1 - 51	Select the number of bits to the left of the binary point, including the sign bit.
[].<number of bits>	0 - 51	Select the number of bits to the right of the binary point. This option is only available when **Signed Fractional** is selected.
Direction	Increment or Decrement	Indicate whether you wish to count up or down.
Starting Value	User Defined	Enter the value with which to begin counting.
Use Control Inputs	On or Off	Indicate whether you would like to use additional control inputs (clock enable and reset).
Clock Phase Selection	User Defined	Phase selection. 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 out of 4 and only the second data out of 4 (sampled at the rate 1) passes through. In other words, the data on phases 1, 3, and 4 do not pass through the delay block.

Table 2–16 shows the block I/O formats.

Table 2–16. Increment Decrement Block I/O Formats Note (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$_{[LP].[RP]}$	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 2–16:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

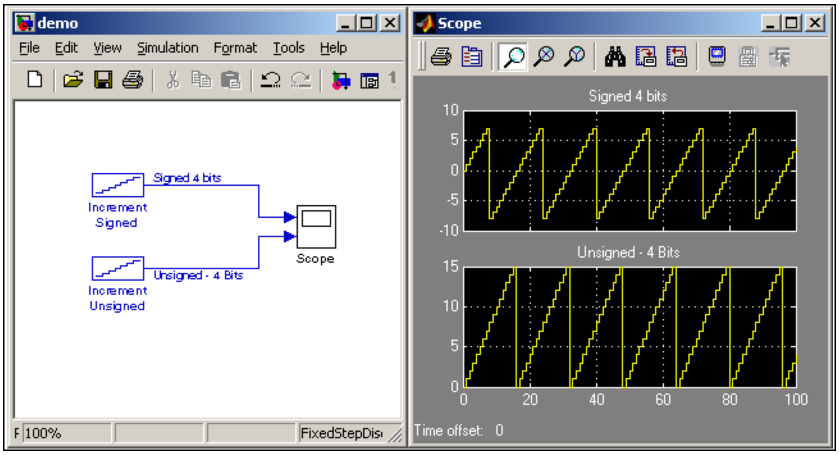
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–5 shows an example using the Increment Decrement block.

Figure 2–5. Increment Decrement Block Example



Integrator Block

The Integrator block is a signed integer integrator with the equation q = q + d. The block can be used for DSP functions such as CIC filters. Table 2–17 shows the block parameters.

Table 2–17. Integrator Block Parameters

Name	Value	Description
Number of Bits	2 - 51	Indicate the number of bits.
Use Control Input	On or Off	Indicate whether you would like to use additional control inputs (clock enable and reset).

Table 2–18 shows the block I/O formats.

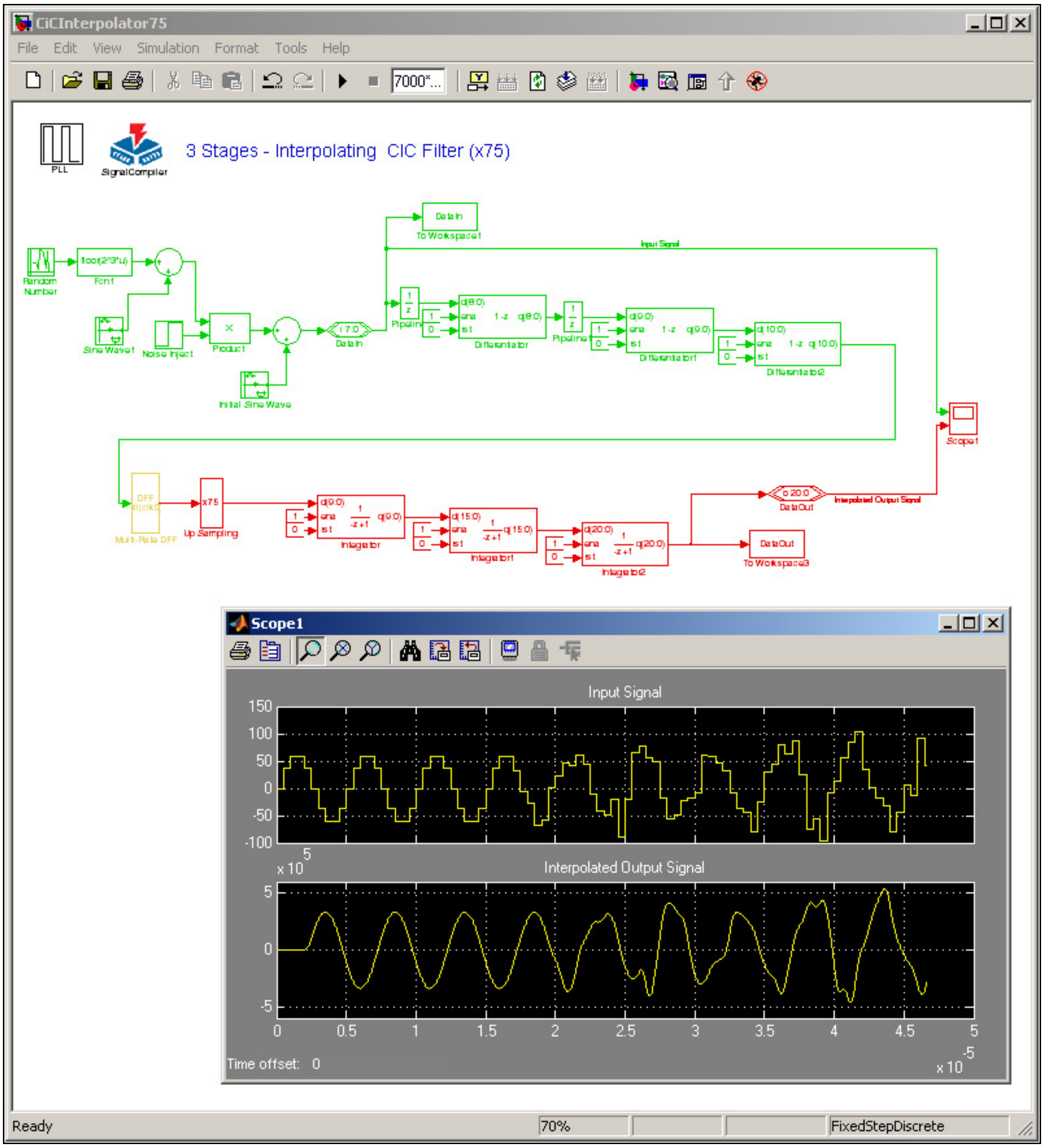
Table 2–18. Integrator Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L1].[0]}$	I1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0)	Explicit
O	$O1_{[L1].[0]}$	O1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0)	Explicit

Notes to Table 2–18:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–6 shows an example of the Integrator Block in a CIC Interpolation design.

Figure 2–6. Integrator Block Example Design



Magnitude Block

The scalar Magnitude block generates the output as the absolute value of the input.

Table 2–19 shows the block I/O formats.

Table 2–19. Magnitude Block I/O Formats Note (1)

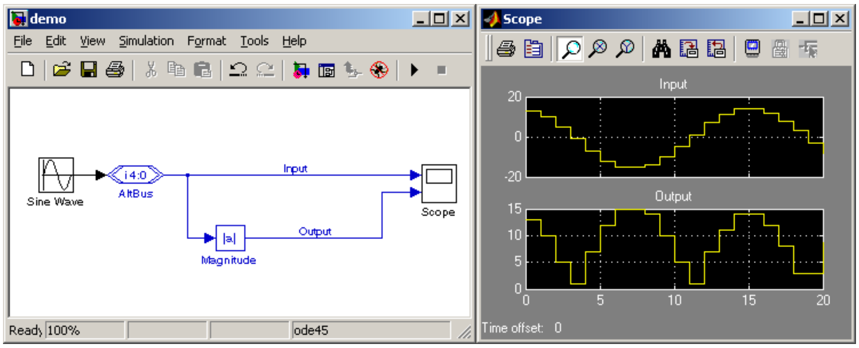
I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit
O	$O1_{[L1 + 1].[R1]}$	O1: in STD_LOGIC_VECTOR({L1 + R1} DOWNTO 0)	Implicit

Notes to Table 2–19:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–7 shows an example using the Magnitude block.

Figure 2–7. Magnitude Block Example



Multiplier Block

The Multiplier block supports two scalar inputs (no multi-dimensional Simulink signals).

Table 2–20 lists the parameters for the Multiplier block.

Table 2–20. Multiplier Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Indicate the bus number format to use for the Multiplier block.
Input [number of bits].[]	1 to 51	Select the number of bits to the left of the binary point of both input signals.
Input [].[number of bits]	0 to 51	Select the number of bits to the right of the binary point of both input signals. This parameter is set to zero (0) unless Signed Fractional is selected as Bus Type.
Full Resolution for Output Result	On or Off	When this parameter is set to On, the multiplier output bit width is full resolution.
Output [number of bits].[]	0 to 51	Select the number of bits to the left of the binary point.
Output [].[number of bits]	0 to 51	Select the number of bits to the left of the binary point of the output signal.
Pipeline Level	1 to 8	Specify the number of levels of pipeline.
Dedicated Multiplier Circuitry	AUTO, YES, NO	Specifies whether to use dedicated multiplier circuitry. Values are “AUTO,” “YES,” and “NO”. For HardCopy™ Stratix, Stratix II, and Stratix GX devices, the value of “AUTO” specifies that the Quartus® II software chooses whether to use the dedicated multiplier circuitry based on the width of the multiplier. For Mercury devices, a value of “AUTO” defaults to no dedicated multiplier circuitry.
Use Control Inputs	On or off	Indicate whether you want to use additional control inputs (clock enable and reset).
The bit width of input port A is equal to the bit width of input port B	On or off	Indicate whether you would like input port A and input port B to have the same bit width.

Table 2–21 describes the I/O ports available on the Multiplier block.

Table 2–21. Multiplier Block Input/Output Ports Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	I1[L].[R] I2[L].[R]	I1: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Explicit
O	O[Lo].[Ro]	O: out STD_LOGIC_VECTOR({Lo + Ro - 1} DOWNTO 0)	Explicit

Notes to Table 2–21:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Multiply Accumulate Block

The Multiply Accumulate block consists of a single multiplier feeding an accumulator. The input can be in signed integer, unsigned integer, or signed binary fractional formats.

Table 2–22 shows the block parameters.

Table 2–22. Multiply Accumulate Block Parameters (Part 1 of 2)

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Choose the number format you wish to use for the bus.
Input A [number of bits].[]	1 - 51	Select the number of data input bits to the left of the binary point, including the sign bit.
Input A [].[number of bits]	0 - 51	Select the number of data input bits to the right of the binary point. This option is only available when **Signed Fractional** is selected.
Input B [number of bits].[]	1 - 51	Select the number of data input bits to the left of the binary point, including the sign bit.
Input B [].[number of bits]	0 - 51	Select the number of data input bits to the right of the binary point. This option is only available when **Signed Fractional** is selected.
Output Result Bits	1 - 51	Indicate the number of output bits.
Pipeline Register	None, Data Inputs, Multiplier Output, Data Inputs and Multiplier	Indicate whether you want to add pipelining to the data inputs, multiplier output, both, or neither.

Table 2–22. Multiply Accumulate Block Parameters (Part 2 of 2)

Name	Value	Description
Accumulator Direction	Add, Subtract	Indicate whether to add or subtract the result of the multiplier.
Use Control Inputs	On or Off	Indicate whether you would like to use additional control inputs (clock enable and reset).
Create Overflow Output Port	On or Off	Indicate whether you want to use an overflow port for the accumulator adder.
Use Dedicated Circuitry	On or Off	If you are targeting Stratix II, Stratix, or Stratix GX devices, turn on this option to implement the functionality in Stratix DSP blocks. If you are not targeting Stratix II, Stratix, or Stratix GX devices, the functionality is implemented in logic elements.

Table 2–23 shows the block I/O formats.

Table 2–23. Multiply Accumulate Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L1].[R1]}$ $I2_{[L2].[R2]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L2 + R2 - 1} DOWNTO 0)	Implicit
O	$O1_{[LO].[RO]}$	O1: out STD_LOGIC_VECTOR({L0 + R0 - 1} DOWNTO 0)	Explicit

Notes to Table 2–23:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

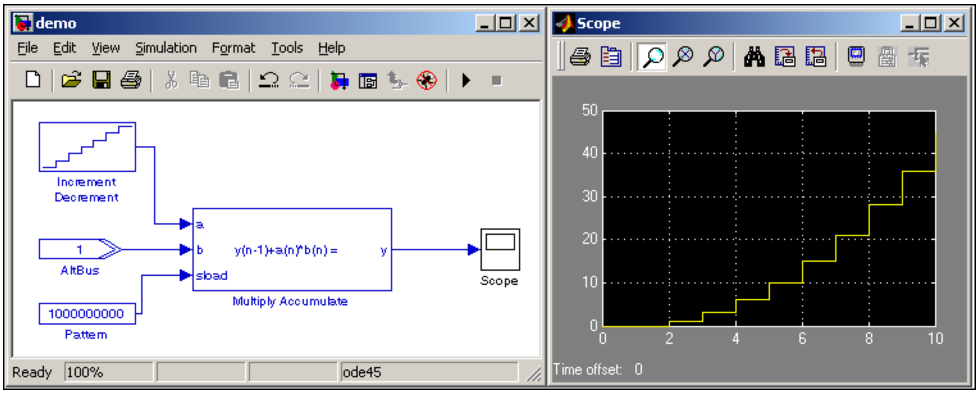
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–8 shows an example using the Multiply Accumulate block.

Figure 2–8. Multiply Accumulate Block Example



Multiply Add Block

The Multiply Add block consists of one or more multipliers feeding a parallel adder. The input can be in signed integer, unsigned integer, or signed binary fractional formats. Table 2–24 shows the block parameters.

Table 2–24. Multiply Add Block Parameters (Part 1 of 2)

Name	Value	Description
Number of Multipliers	2, 3, 4	Choose how many multipliers you want to feed the adder.
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Choose the number format you wish to use for the bus.
Inputs [number of bits].[]	1 - 51	Select the number of data input bits to the left of the binary point, including the sign bit.
Inputs [].[number of bits]	0 - 51	Select the number of data input bits to the right of the binary point. This option is only available when **Signed Fractional** is selected.
Adder Mode	Add Add, Add Sub, Sub Add, Sub Sub	Choose the operation of the adder.
Pipeline Register	No Register, Inputs Only, Multiplier Only, Adder Only, Inputs and Multiplier, Inputs and Adder, Multiplier and Adder, Inputs Multiplier and Adder	Choose the elements to which you want to add pipelining.

Table 2–24. Multiply Add Block Parameters (Part 2 of 2)

Name	Value	Description
Use Clock Enable	On or Off	Indicate whether you would like to use an additional control input (clock enable).
Use Dedicated Circuitry	On or Off	If you are targeting Stratix II, Stratix, or Stratix GX devices, turn on this option to implement the functionality in Stratix DSP blocks. If you are not targeting Stratix II, Stratix, or Stratix GX devices, the functionality is implemented in logic elements.
One Input is Constant	On or Off	Turn on this option if one of the inputs is a constant. This option is used together with the **Constant Values** parameter.
Constant Values	User Defined	Type the constant value in this box as a MATLAB array. This option is only available if you have turned on the **One Input is Constant** option.

Table 2–25 shows the block I/O formats.

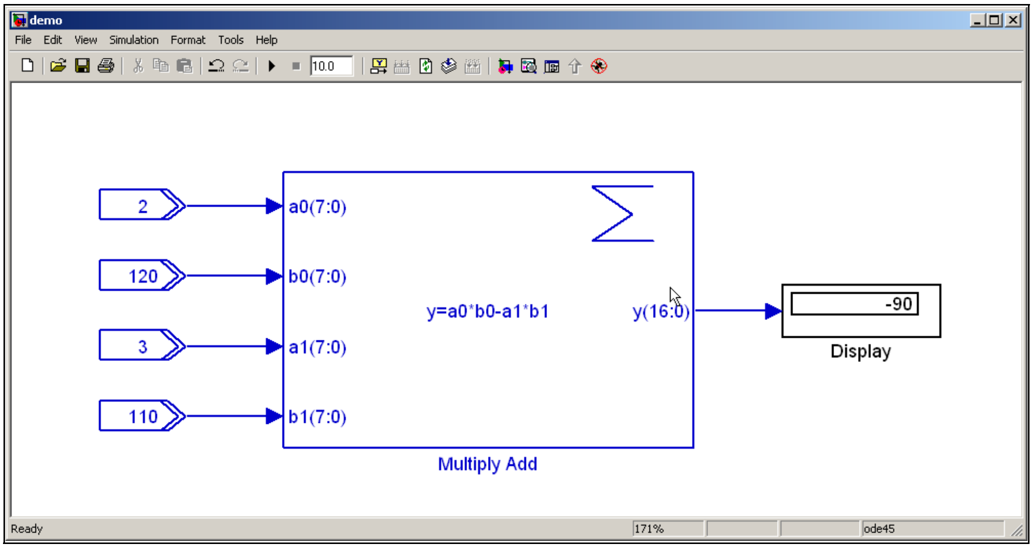
Table 2–25. Multiply Add Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L1].[R1]}$ .... $Ii_{[L1].[R1]}$ ... $In_{[L1].[R1]}$ 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) where 3 < n < 9	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 2–25:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–9 shows an example using the Multiply Add block.

Figure 2–9. Multiply Add Block Example



Parallel Adder Subtractor Block

The Parallel Adder Subtractor block takes any input data type. If the input widths are not the same, the SignalCompiler sign extends the buses so that they match the largest input width. The VHDL generated has an optimized, balanced adder tree. Table 2–26 shows the block parameters.

Table 2–26. Parallel Adder Subtractor Block Parameters (Part 1 of 2)

Name	Value	Description
Number of Inputs	2 - 16	Indicate the number of inputs you wish to use.
Add (+) Sub (-)	User Defined	Specify addition or subtraction operation of each port with the characters (+) / (-). in other words, for 3 ports +-+ yields a - b + c. SignalCompiler will not accept two consecutive subtractions (for example, --); however, -+- is acceptable.
Pipeline	On or Off	When this option is turned on, the pipeline delay is equal to ceil(log$_2$(number of inputs)), such that the pipeline register locations are balanced in the adder tree to ensure optimal timing.

Table 2–26. Parallel Adder Subtractor Block Parameters (Part 2 of 2)

Name	Value	Description
Use Control Inputs	On or Off	Indicate whether you would like to use additional control inputs (clock enable and reset).
Clock Phase Selection	User Defined	Phase selection. 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 out of 4 and only the second data out of 4 (sampled at the rate 1) passes through. In other words, the data on phases 1, 3, and 4 do not pass through the delay block.

Table 2–27 shows the block I/O formats.

Table 2–27. Parallel Adder Subtractor Block I/O Formats Note (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	$O1_{[max(LI) + ceil(log2(N))].[max(RI)]}$	O1: out STD_LOGIC_VECTOR({max(LI) + ceil(log2(N)) + max(RI) - 1} DOWNTO 0)	Implicit

Notes to Table 2–27:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

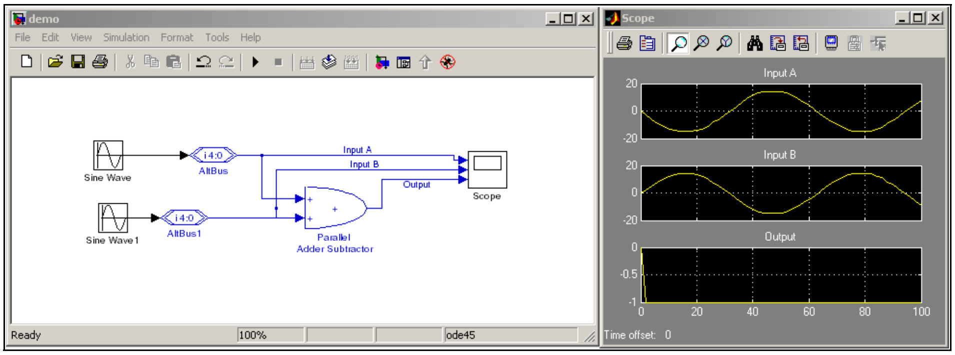
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–10 shows an example using the Parallel Adder Subtractor block.

Figure 2–10. Parallel Adder Subtractor Block Example



Pipelined Adder Block

The Pipelined Adder block is a pipelined adder/subtractor. Table 2–28 shows the block parameters.

Table 2–28. Pipelined Adder Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Indicate the bus number format that you want to use for the counter.
[number of bits].[]	1 to 51	Select the number of bits to the left of the binary point.
[].[number of bits]	0 to 51	Select the number of bits to the right of the binary point for the gain. This option is zero (0) unless the Bus Type parameter value is “Signed Fractional”
Pipeline	0 - 4	Specify the number of pipeline levels.
Use Control Signal	On or Off	When set to “On”, the rst (reset), ena (clock enable), cin (carry in), and add_su (addition when logic value 1, subtraction when logic value 0) signals are added to the block.

Table 2–29 shows the block I/O formats.

Table 2–29. Pipelined Adder Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L].[R]}$ $I2_{[L].[R]}$	I1: in STD_LOGIC_VECTOR({L + R} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L + R} DOWNTO 0)	Explicit
O	$O1_{[L].[R]}$	O1: out STD_LOGIC_VECTOR({L + R} DOWNTO 0)	Explicit

Notes to Table 2–29:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Product Block

The Product block supports two scalar inputs (no multi-dimensional Simulink signals).

There are two differences between the product block and the multiplier block: the product block uses a clock phase selection while the multiplier block does not, and 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 need to be specified as parameters.

☞ The Simulink software also provides a Product block. If you use the Simulink Product block in your model, you can only use it for simulation; SignalCompiler cannot convert it to HDL. If you try to use the Simulink Product block with SignalCompiler, SignalCompiler either treats it as a block box or generates an error.

Table 2–30 shows the block parameters.

Table 2–30. Product Block Parameters (Part 1 of 2)

Name	Value	Description
Pipeline	0 - 4	The Pipeline value represents the delay.
Use LPM	On or Off	This parameter is used for synthesis. When the **Use LPM** option is turned 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. If the option is turned 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 combinatorial logic or register balancing optimization—the `LPM_MULT` implementation generally yields a better result for both speed and area.
Use Dedicated Multiplier Circuitry	On or Off	Turn on the Dedicated Multiplier Circuitry option if you want to use the dedicated multiplier circuitry in Mercury, Stratix II, Stratix, or Stratix GX devices. This option is ignored if you do not target one of these devices.

Table 2–30. Product Block Parameters (Part 2 of 2)

Name	Value	Description
Use Control Inputs	On or Off	Indicate whether you would like to use additional control inputs (clock enable and reset).
Clock Phase Selection	User Defined	Phase selection. 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 out of 4 and only the second data out of 4 (sampled at the rate 1) passes through. In other words, the data on phases 1, 3, and 4 do not pass through the delay block.

Table 2–31 shows the block I/O formats.

Table 2–31. Product Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L1].[R1]}$ $I2_{[L2].[R2]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L2 + R2 - 1} DOWNTO 0)	Implicit
O	$O1_{[L1 + L2].[2 \times max(R1,R2)]}$	O1: out STD_LOGIC_VECTOR({L1 + L2 + 2 x max(R1,R2) - 1} DOWNTO 0)	Implicit

Notes to Table 2–31:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

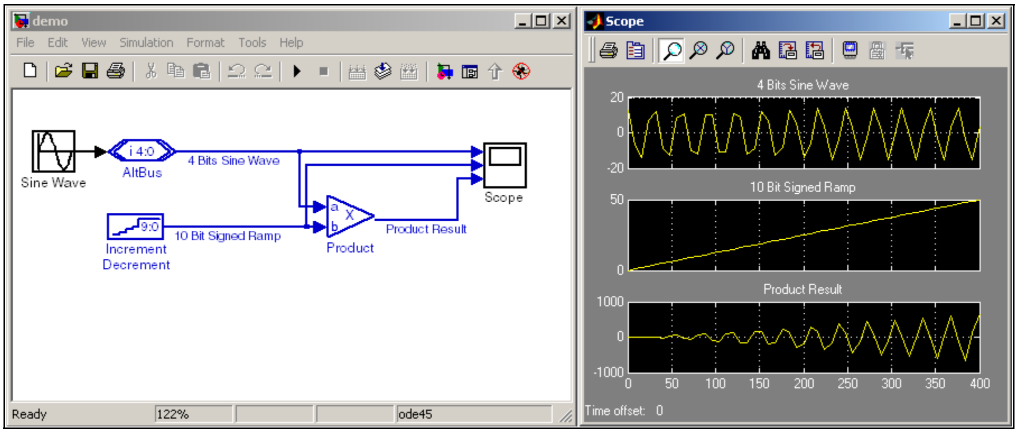
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–11 shows an example using the Product block.

Figure 2–11. Product Block Example



SOP TAP Block

The SOP TAP block performs a sum of products for two to four taps. The block has the equations:

$$q = c0 \times din(n) + c1 \times din\ (n - 1)$$

when the number of taps is 2.

$$q = c0 \times din(n) + c1 \times din(n - 1) + c2 \times din(n - 2) + c3 \times din(n - 3)$$

when the number of taps is 4.

You can use this block to build two to four tap FIR filters. You can cascade SOP TAP blocks to create filters with more taps. Table 2–32 shows the SOP TAP block parameters.

Table 2–32. SOP TAP Block Parameters

Name	Value	Description
Input Number of Bits	2 - 51	Indicate the number of bits.
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Indicate the bus number format that you want to use for the counter.
Number of Taps	2 or 4	Indicate the number of taps.

Table 2–33 shows the block I/O formats.

Table 2–33. SOP TAP Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L].[R]}$ $I2_{[L].[R]}$ $I3_{[L].[R]}$ $I4_{[L].[R]}$	I1: in STD_LOGIC_VECTOR({L + R -1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L + R -1} DOWNTO 0) I3: in STD_LOGIC_VECTOR({L + R -1} DOWNTO 0) I4: in STD_LOGIC_VECTOR({L + R -1} DOWNTO 0)	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 2–33:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Sum of Products Block

The Sum of Products block effects the following mathematical expression:

$$q = a(0) + \ldots + a(i)Ci + \ldots + a(n)Cn$$

Table 2–34 lists the parameters for the Sum of Products block.

Table 2–34. Sum of Products Block Parameters

Name	Value	Description
Input Data Number of Bits	1 - 51	Select the number of bits to the left of the binary point of all input signals.
Number of Coefficients	2 - 8	Select the number of coefficients.
Coefficient Number of Bits	1 - 51	Select 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	Indicate the coefficient values as signed integers.
Output [].[number of bits]		Select the number of bits to the left of the binary point of the output signal.
Pipeline Level		Specify the number of levels of pipeline.
FPGA Implementation	Distributed, Arithmetic, Dedicated Multiplier Circuitry, Auto	Specify whether to use a distributed arithmetic implementation.
Use Control Inputs	On or Off	Indicate whether you would like to use additional control inputs (clock enable and reset).

Table 2–35 shows the block I/O formats.

Table 2–35. Sum of Products Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L].[0]}$ $I2_{[L].[0]}$ $Ii_{[L].[0]}$ ... $In_{[L].[0]}$	I1: in STD_LOGIC_VECTOR({L - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L - 1} DOWNTO 0) Ii: in STD_LOGIC_VECTOR({L - 1} DOWNTO 0) ... In: in STD_LOGIC_VECTOR({L - 1} DOWNTO 0)	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 2–35:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Square Root Block

The Square Root block returns the square root of unsigned input data.

Table 2–36 lists the parameters for the Square Root block.

Table 2–36. Square Root Block Parameters

Name	Value	Description
Input [number of bits].[]	0 - 51	Select the number of bits of the unsigned input signal.
Pipeline Level	0 - 12	Specify the number of levels of pipeline.

Table 2–37 shows the block I/O formats.

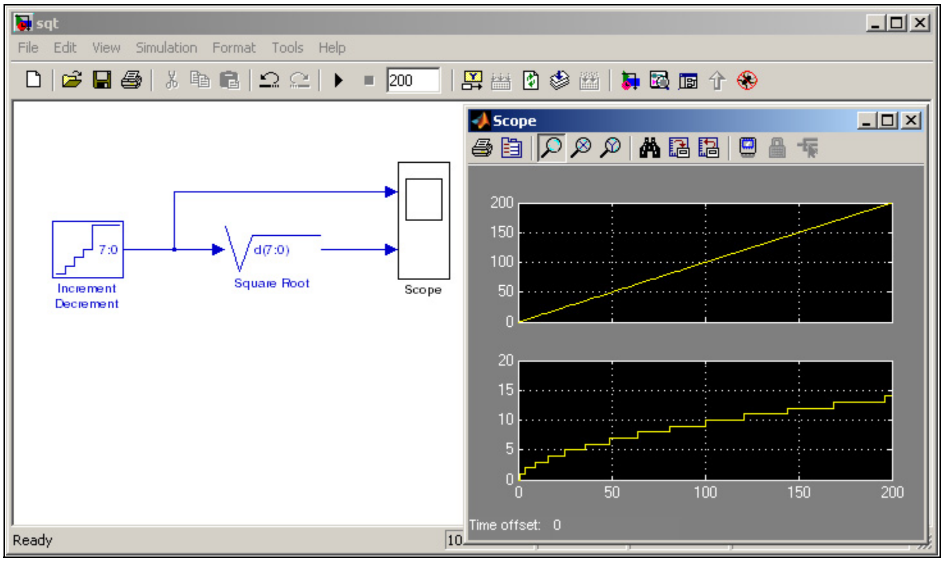
Table 2–37. Square Root Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L].[R]}$ $I2_{[L].[R}$	I1: in STD_LOGIC_VECTOR({L + R} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L + R} DOWNTO 0)	Explicit
O	$O1_{[L].[R]}$	O1: out STD_LOGIC_VECTOR({L + R} DOWNTO 0)	Explicit

Notes to Table 2–37:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 2–12 shows an example of the Square Root block in a CIC Interpolation design.

Figure 2–12. Square Root Block Example Design



Chapter 3. IO & Bus Library

Introduction

The blocks in the IO & Bus library are used to manipulate signals and buses to perform operations such as truncation, saturation, bit extraction, or bus format conversion.

AltBus Block

The AltBus block casts a floating-point Simulink bus to a fixed-point bus. You can insert AltBus into a data or I/O primary path for inputs and outputs.

When casting a signal to fixed point, you must specify the bit width. You have the option of truncating, saturating, or rounding the result. If you choose rounding or saturation, the appropriate logic is inserted.

Table 3–1 shows the **AltBus** block parameters.

Table 3–1. AltBus Block Parameters (Part 1 of 2)

Name	Value	Description
Node Type	Internal Node, Input Port, Output Port, Constant, Black Box Input, Black Box Output	Indicate the type of node you wish to create.
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, or Single Bit	Choose the number format of the bus.
[number of bits].[]	1 - 51	Indicate 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 - 51	Indicate the number of bits to the right of the binary point. This parameter only applies to signed fractional buses.
Saturate	On or Off	When this option is turned 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 this option is turned off, the MSB is truncated. This option is not valid for the input port or constant node types.
Round	On or Off	When this option is turned on, the output is rounded away from zero. When this option is turned off, the LSB is truncated: <*int*>(input +0.5). This option is not valid for the input port or constant node types.

Table 3–1. AltBus Block Parameters (Part 2 of 2)

Name	Value	Description
Bypass Bus Format	On or Off	Turn on this option if you wish to perform simulation in Simulink using floating-point numbers.
Constant Value	Double	Indicate the constant value that will be formatted with the bus parameter specified.

Table 3–2 shows the block I/O formats.

Table 3–2. AltBus Block I/O Formats *Note (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 3–2:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

You can use the AltBus block in a Simulink design in any of the following modes:

- AltBus Block Input Port & Output Port Modes
- AltBus Block Internal Node Mode
- AltBus Black Box Input Output Mode
- AltBus Block Constant Mode

AltBus Block Input Port & Output Port Modes

The Input Port and Output Port modes are used to define the boundaries of the hardware implementation as well as to cast floating-point Simulink signals (coming from generic Simulink blocks) to signed binary fractional format (feeding DSP Builder blocks). Table 3–3 and Figure 3–1 illustrate how a floating-point number (4/3 = 1.3333) is cast into SBF format with three different binary point locations.

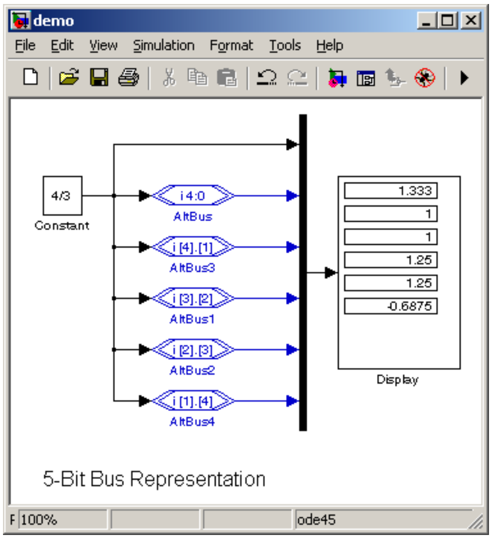
Table 3–3. Floating-Point Numbers Cast to SBF

Bus Notation	Input	Simulink	VHDL
[4].[1]	4/3	1.00	2.00
[2].[3]	4/3	1.25	10.00
[1].[4]	4/3	-0.6875 *(1)*	-11.00

Note to Table 3–3:

(1) In this case, more bits are needed to represent the integer part of the number.

Figure 3–1. Floating-Point Conversion

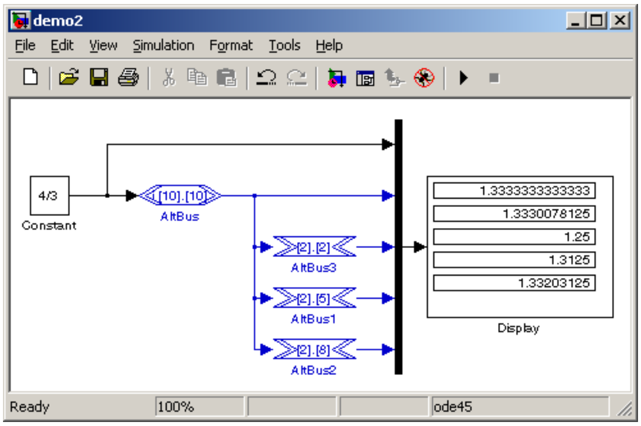


AltBus Block Internal Node Mode

This mode is used to convert a Simulink signal from one SBF format to another. This mode is used to assign the bus width of an internal node that will be implemented in hardware. Figure 3–2 illustrates the usage of

AltBus in Internal Node mode (blocks Altbus1, AltBus2, and AltBus3) and Input Port mode (block AltBus). In this example a 20-bit bus with a ([10].[10]) SBF format is converted to a 4-bit bus with a [2].[2] SBF format.

Figure 3–2. Block Internal Node



In VHDL, this operation 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))
```

You can also perform additional internal bus manipulation with the Altera® BusConversion, ExtractBit, or BuildBus blocks.

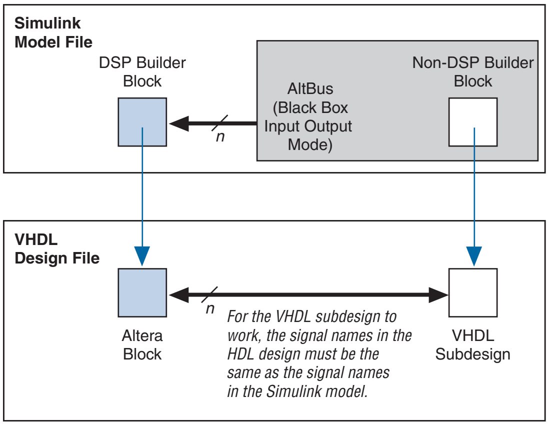
AltBus Black Box Input Output Mode

This AltBus mode is used for hierarchical designs. You should use this node type if you do not want SignalCompiler to translate the sub-level design to HDL (that is, only the top-level symbol appears in the HDL). This mode is useful when your model has a Simulink block that is associated with a separate HDL block.

☞ The pin names of the HDL block must match the pin names of the Simulink block.

Figure 3–3 illustrates the Black Box Input Output mode.

Figure 3–3. Black Box Input Output Mode



AltBus Block Constant Mode

Use this mode when a bus or bit must be set to a static value. SignalCompiler translates the static value to a constant STD_LOGIC or STD_LOGIC_VECTOR in VHDL. During synthesis, the synthesis tool typically reduces the gate count of any logic fed by this constant signal.

☞ If you use the Simulink Constant block in your Altera design, you can only use it for simulation. If you try to use the Simulink Constant block with SignalCompiler, SignalCompiler will either treat it as a block box or will generate an error.

Binary Point Casting Block

The Binary Point Casting block moves the input bus binary point position. The output bit width remains equal to the input bit width.

Table 3–4 shows the Binary Point Casting block parameters.

Table 3–4. Binary Point Casting Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, or Single Bit	Choose the number format of the bus.
[number of bits].[]	1 - 51	Indicate 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 - 51	Indicate the number of bits to the right of the binary point. This parameter only applies to signed fractional buses.
Output Binary Point Position	0 - 51	Specify the binary point location of the output.

Table 3–5 shows the block I/O formats.

Table 3–5. Binary Point Casting Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[Li].[Ri]}$	I1: in STD_LOGIC_VECTOR({Li + Ri - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L2 - 1} DOWNTO 0)	Explicit
O	$O1_{[LO].[RO]}$ where LO + RO - Li + Ri	O1: out STD_LOGIC_VECTOR({LO + RO - 1} DOWNTO 0)	Explicit

Notes to Table 3–5:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

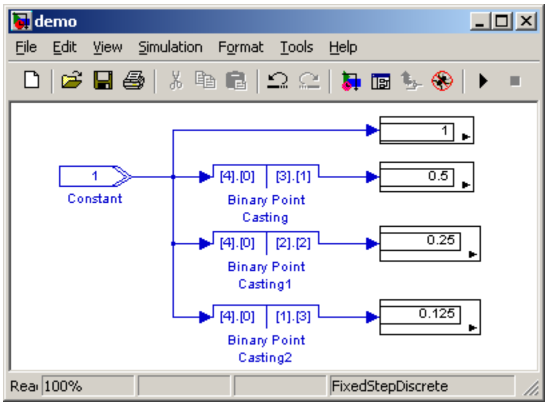
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 3–4 shows a design example using the Binary Point Casting block.

Figure 3–4. Binary Point Casting Block Example



BusBuild Block

The BusBuild block is used to construct buses from single-bit inputs. The output bus is defined in signed binary fractional representation. You can choose the bus type that you wish to use, and specify the number of bits on either side of the binary point. The BusBuild and ExtractBit blocks are typically used when mixing bit-level Boolean operation with arithmetic operations. The HDL mapping of BusBuild is a simple wire.

The input MSB (which is the sign bit of the signed binary fractional bus) is shown at the bottom left of the symbol and the input LSB is displayed at the top left of the symbol. Table 3–6 shows the block parameters.

Table 3–6. BusBuild Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, or Unsigned Integer	Choose the number format of the bus.
Output [number of bits].[]	1 - 51	Indicate the number of bits to the left of the binary point, including the sign bit.
Output [].[number of bits]	0 - 51	Indicate the number of bits to the right of the binary point. This parameter only applies to signed binary fractional buses.

Table 3–7 shows the block I/O formats.

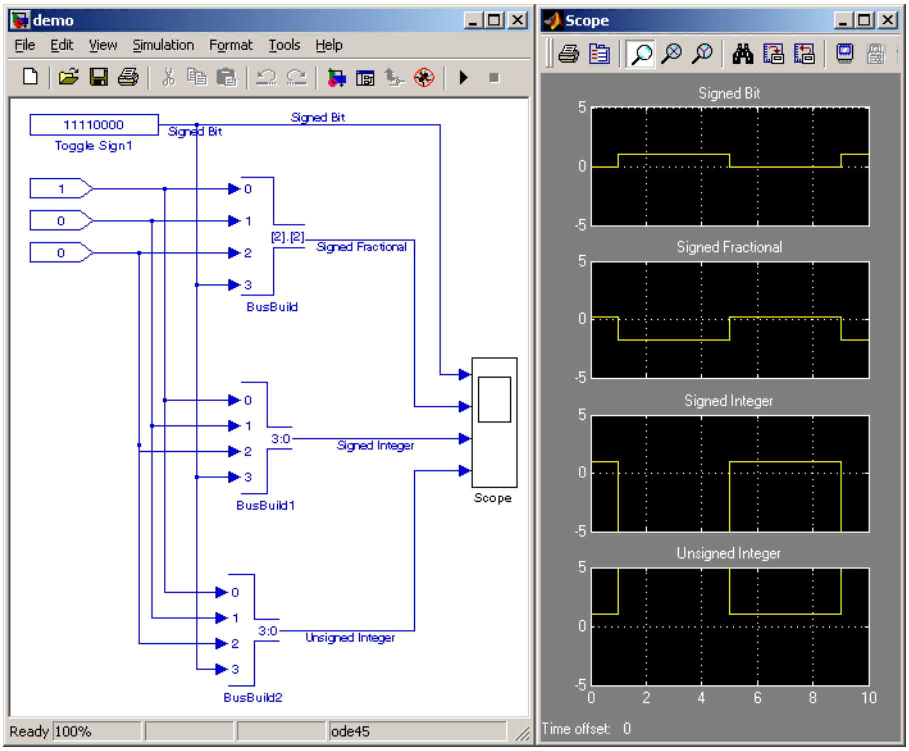
Table 3–7. BusBuild Block I/O Formats *Note (1)*

I/O	Simulink *(2)*, *(3)*	VHDL	Type *(4)*
I	$I1_{[1]}$ … $INPi_{[1]}$ …. $INPn_{[1]}$	I1: in STD_LOGIC … INPi: in STD_LOGIC …. INPn: in STD_LOGIC	Explicit
O	$O1_{[LP].[RP] \text{ with } LP + RP = N}$ where N is the number of inputs	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 3–7:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 3–5 shows a design example using the BusBuild block.

Figure 3–5. BusBuild Block Example



Bus Concatenation Block

This block concatenates two buses. The result is A + B bits wide, where A is the most significant bit (MSB) slice of the output bus and B is the least significant bit (LSB) slice of the output bus. Table 3–8 shows the block parameters.

Table 3–8. Bus Concatenation Block Parameters

Name	Value	Description
Bus A Width	1 - 51	Enter the width of the first bus to concatenate.
Bus B Width	1 - 51	Enter the width of the first bus to concatenate.
Output Is Signed	On or Off	Indicate whether the output bus is signed.

Table 3–9 shows the block I/O formats.

Table 3–9. Bus Concatenation Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L1].[0]}$ $I2_{[L2].[0]}$	I1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L2 - 1} DOWNTO 0)	Explicit
O	$O1_{[L1 + L2].[0]}$	O1: out STD_LOGIC_VECTOR({L1 + L2 - 1} DOWNTO 0)	Explicit

Notes to Table 3–9:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

BusConversion Block

This block converts a bus from one node type to another. Table 3–10 shows the block parameters.

Table 3–10. BusConversion Block Parameters

Name	Value	Description
Input Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Choose the input bus type for the simulator, VHDL or both.
Input [number of bits].[]	1 - 51	Indicate the number of bits to the left of the binary point including the sign bit.
Input [].[number of bits]	1 - 51	Indicate the number of bits to the right of the binary point. This parameter only applies to signed binary fractional buses.
Output Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Choose the output bus type for the simulator, VHDL or both.
Output [number of bits].[]	1 - 51	Indicate the number of bits to the left of the binary point
Output [].[number of bits]	1 - 51	Indicate the number of bit on the right side of the binary point. This parameter only applies to signed binary fractional buses.
Input Bit Connected to Output MSB	0 - 51	Indicate which slice of the input bus to use. This parameter designates the starting point (LSB) of the slice.
Input Bit Connected to Output LSB	0 - 51	Indicate which slice of the input bus to use. This parameter designates the ending point (MSB) of the slice.
Round	On or Off	Indicate whether rounding should be turned on or off.
Saturate	On or Off	Indicate whether saturation should be turned on or off.

Table 3–11 shows the block I/O formats.

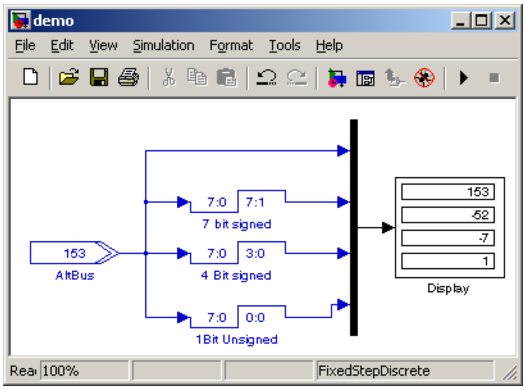
Table 3–11. Bus Conversion Block I/O Formats *Note (1)*

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 3–11:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 3–6 shows a design example using the BusConversion block.

Figure 3–6. BusConversion Block Example



Constant Block

The Constant block is derived from the AtlBus block. This block performs a subset of the functionality of the AltBus block.

Table 3–12 shows the Constant block parameters.

Table 3–12. Constant Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, or Single Bit	Choose the number format of the bus.
[number of bits].[]	1 - 51	Indicate 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 - 51	Indicate the number of bits to the right of the binary point. This parameter only applies to signed fractional buses.
Constant Value	Double	Indicate the constant value that will be formatted with the bus parameter specified.

Table 3–13 shows the block I/O formats.

Table 3–13. Constant Block I/O Formats Note (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 3–13:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

ExtractBit Block

The ExtractBit block reads a Simulink bus in signed binary fractional format and outputs the bit specified with the **Extracted Bit** parameter. Table 3–14 shows the block parameters.

Table 3–14. ExtractBit Block Parameters

Name	Value	Description
Input [number of bits].[]	1 - 51	Indicate the number of bits to the left of the binary point, including the sign bit.
Input [].[number of bits]	0 - 51	Indicate the number of bits to the right of the binary point.
Extracted Bit	0 - 32	Indicate which bit to extract.

Table 3–15 shows the block I/O formats.

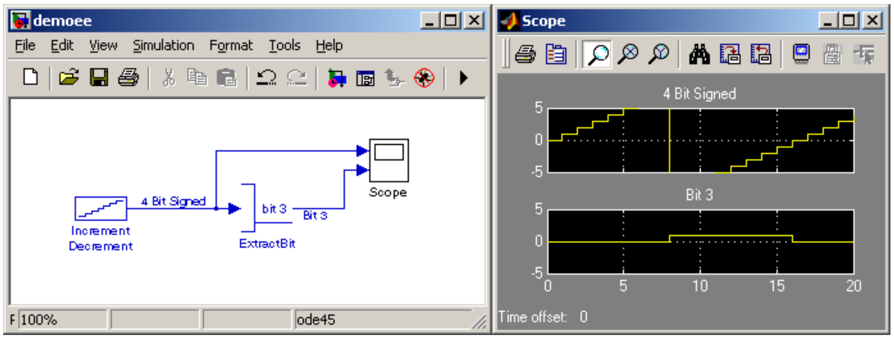
Table 3–15. ExtractBit Block I/O Formats Note (1)

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 3–15:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 3–7 shows a design example using the ExtractBit block.

Figure 3–7. ExtractBit Block Example



GND Block

The GND block is a single bit that outputs a constant 0.

Table 3–16 shows the block I/O formats.

Table 3–16. GND Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
O	$O1_{[1].[0]}$	O1: out STD_LOGIC	Explicit

Notes to Table 3–16:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

GlobalRst Block

The GlobalRst block is a single bit. This block provides RTL implementation and synthesis directives to the SignalCompiler block. All input pins driven by the SCLR block get connected to the global reset of the VHDL or Verilog HDL models created by the SignalCompiler block.

Table 3–17 shows the block I/O formats.

Table 3–17. GlobalRst Block I/O Formats *Note (1)*

I/O	Simulink (2), (3)	VHDL	Type (4)
O	$O1_{[1].[0]}$	O1: out STD_LOGIC	Explicit

Notes to Table 3–17:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Input Block

The Input block is derived from the AtlBus block. This block performs a subset of the functionality of the AltBus block.

Table 3–18 shows the Input block parameters.

Table 3–18. Input Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, or Single Bit	Choose the number format of the bus.
[number of bits].[]	1 - 51	Indicate 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 - 51	Indicate the number of bits to the right of the binary point. This parameter only applies to signed fractional buses.

Table 3–19 shows the block I/O formats.

Table 3–19. Input Block I/O Formats Note (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 3–19:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Output Block

The Output block is derived from the AtlBus block. This block performs a subset of the functionality of the AltBus block.

Table 3–20 shows the Output block parameters.

Table 3–20. Output Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer, or Single Bit	Choose the number format of the bus.
[number of bits].[]	1 - 51	Indicate 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 - 51	Indicate the number of bits to the right of the binary point. This parameter only applies to signed fractional buses.

Table 3–21 shows the block I/O formats.

Table 3–21. Output Block I/O Formats Note (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 3–21:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Round Block

The Round block rounds the output away from zero. You can indicate how many LSB bits to remove and whether to round or truncate the output, such *output* = fix(*input*/2^(LSB bits)) + 0.5).

Table 3–22 shows the Round block parameters.

Table 3–22. Round Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, or Unsigned Integer	Choose the number format of the bus.
Input [number of bits].[]	2 - 51	Indicate the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
Input [].[number of bits]	0 - 51	Indicate the number of bits to the right of the binary point. This parameter only applies to signed fractional buses.
Number of LSB Bits to Remove	0 - 51	Indicate how many bits to remove.
Rounding Type	Round or Truncate	Choose whether to round or truncate the output.
Pipeline	On or Off	Indicate whether you would like to pipeline the function.

Table 3–23 shows the block I/O formats.

Table 3–23. Round Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L1].[R1]}$	I1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
O	$O1_{[LP].[RP]}$	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 3–23:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Saturate Block

When you use the Saturate block, 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 choose to truncate the MSB.

Table 3–24 shows the Saturate block parameters.

Table 3–24. Saturate Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, or Unsigned Integer	Choose the number format of the bus.
Input [number of bits].[]	2 - 51	Indicate the number of bits to the left of the binary point, including the sign bit. This parameter does not apply to single-bit buses.
Input [].[number of bits]	0 - 51	Indicate the number of bits to the right of the binary point. This parameter only applies to signed fractional buses.
Number of MSB Bits to Remove	0 - 51	Indicate how many bits to remove.
Saturation Type	Saturate or Truncate MSB	Choose whether to saturate or truncate the output.
Pipeline	On or Off	Indicate whether you would like to pipeline the function.

Table 3–25 shows the block I/O formats.

Table 3–25. Saturate Block I/O Formats *Note (1)*

I/O	Simulink *(2), (3)*	VHDL	Type *(4)*
I	I1$_{[L1].[R1]}$	I1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit
O	O1$_{[LP].[RP]}$	O1: out STD_LOGIC_VECTOR({LP + RP - 1} DOWNTO 0)	Explicit

Notes to Table 3–25:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

VCC Block

The VCC block is a single bit that outputs a constant 1.

Table 3–26 shows the block I/O formats.

Table 3–26. VCC Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
O	$O1_{[1]}$	O1: out STD_LOGIC	Explicit

Notes to Table 3–26:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Chapter 4. Complex Signals Library

Like Simulink, DSP Builder supports native complex signal types. Using complex number notation simplifies the design of applications such as FFT, I-Q modulation, and complex filters.

☞ When connecting DSP Builder blocks to blocks from the Complex Signals library (for example, connecting AltBus to Complex AddSub), you must use Real-Imag to Complex or Complex to Real-Imag blocks between the blocks. See Figure 4–2 for an example.

Butterfly Operator Block

The Butterfly Operator block performs the following arithmetic operation on complex signed integer numbers:

A = a + bW
B = a - bW

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 x [input bit width] + 2. The output bit width and output LSB bit parameters are used to specify the bit slice used 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, the full precision is 34 bits and the output ports are A[19:4] and B[19:4]

Table 4–1 shows the Butterfly Operator block parameters.

Table 4–1. Butterfly Operator Block Parameters

Name	Value	Description
Input Bit Width (a, b, W)	4 - 51	Specify the bit width of the complex signed integer inputs a, b, and W.
Output Bit Width (A and B)	4 - 51	Specify the bit width of the complex signed integer outputs A and B.
Output LSB Bit	4 - 51	Specify the LSB bit of the output bus slice of the full resolution computation.

Table 4–2 shows the block I/O formats.

Table 4–2. Butterfly Operator Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{Real([Li].[0])Imag([Li].[0])}$ $I2_{Real([Li].[0])Imag([Li].[0])}$ $I3_{Real([Li].[0])Imag([Li].[0])}$	I1Real: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0) I1Imag: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0) I2Real: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0) I2Imag: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0) I3Real: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0) I3Imag: in STD_LOGIC_VECTOR({Li - 1} DOWNTO 0)	Explicit
O	$O1_{Real([Lo].[0])Imag([Li].[0])}$ $O2_{Real([Lo].[0])Imag([Li].[0])}$	O1Real: in STD_LOGIC_VECTOR({Lo - 1} DOWNTO 0) O1Imag: in STD_LOGIC_VECTOR({Lo - 1} DOWNTO 0) O2Real: in STD_LOGIC_VECTOR({Lo - 1} DOWNTO 0) O2Imag: in STD_LOGIC_VECTOR({Lo - 1} DOWNTO 0)	Explicit

Notes to Table 4–2:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

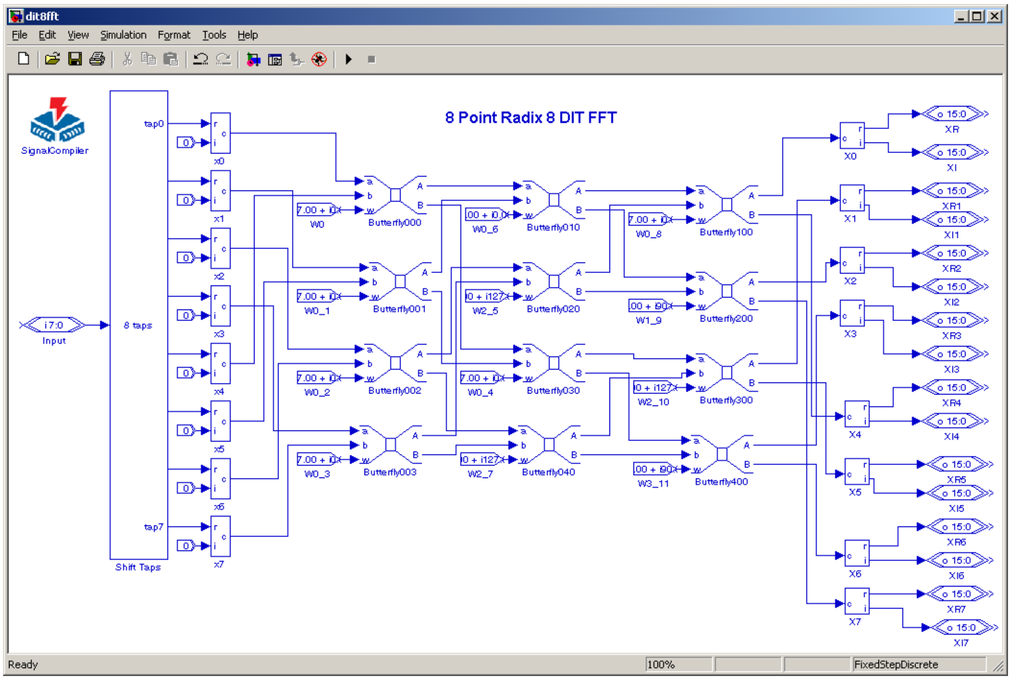
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 4–1 shows a design example using the Butterfly Operator block.

Figure 4–1. Butterfly Operator Block Example



Complex AddSub Block

The Complex AddSub block performs output addition or subtraction of two scalar complex inputs.

Table 4–3 shows the Complex AddSub block parameters.

Table 4–3. Complex AddSub Block Parameters

Name	Value	Description
Arithmetic Operation	Add or Subtract	Choose which operation to perform.

Table 4–4 shows the block I/O formats.

Table 4–4. Complex AddSub Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{Real([L1].[R1])Imag([L1].[R1])}$ $I2_{Real([L2].[R2])Imag([L2].[R2])}$	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)	Implicit
O	$O1_{Real(max(L1,L2) + 1),(max(RI,R2) + 1)Imag(max(L1,L2) + 1),(max(RI,R2) + 1)}$	O1Real: in STD_LOGIC_VECTOR({max(LI,L2) + max(RI,R2)} DOWNTO 0) O1Imag: in STD_LOGIC_VECTOR({max(LI,L2) + max(RI,R2)} DOWNTO 0)	Implicit

Notes to Table 4–4:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

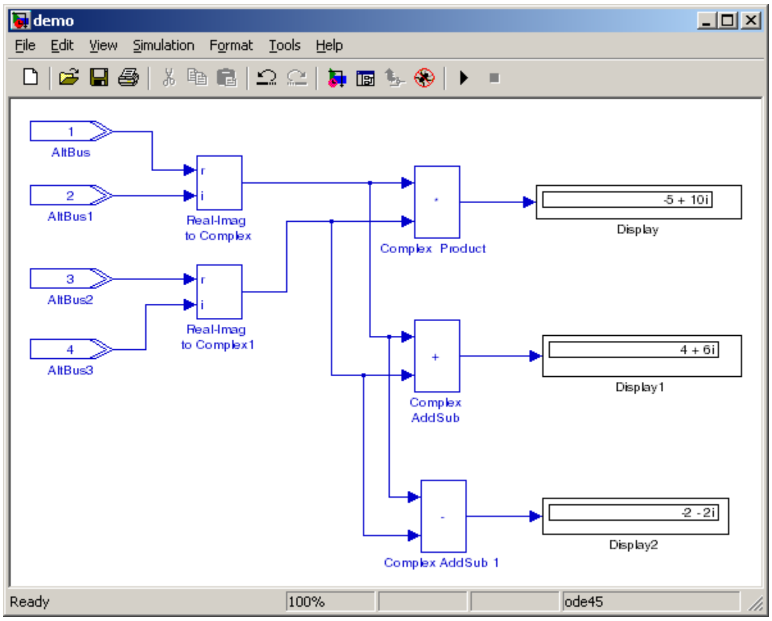
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 4–2 shows a design example using the Complex Add Sub block.

Figure 4–2. Complex Add Sub Block Example



Complex Product Block

The Complex Product block performs output multiplication of two scalar complex inputs.

Table 4–5 shows the Complex Product block parameters.

Table 4–5. Complex Product Block Parameters

Name	Value	Description
Arithmetic Operation	Multiply	Choose which operation to perform.

Table 4–6 shows the block I/O formats.

Table 4–6. Complex Product Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	I1$_{Real([L1].[R1])Imag([L1].[R1])}$ I2$_{Real([L2].[R2])Imag([L2].[R2])}$	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)	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 4–6:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 4–2 on page 4–5 shows a design example using the Complex Product block.

Complex Multiplexer Block

The Complex Multiplexer block is a 2 to 1 multiplexer for complex numbers. The select line (port number 3) is a non-complex scalar.

Table 4–7 shows the block I/O formats.

Table 4–7. Complex Multiplexer Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{Real([L1].[R1])Imag([L1].[R1])}$ $I2_{Real([L2].[R2])Imag([L2].[R2])}$ $I3_{[1]}$	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	Implicit
O	$O1_{Real(max(L1,L2)),(max(RI,R2))Imag(max(L1,L2)),(max(RI,R2))}$	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 4–7:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Complex Conjugate Block

The Complex Conjugate block outputs the conjugate or the inverse of the scalar complex inputs.

Table 4–8 shows the Complex Conjugate block parameters.

Table 4–8. Complex Conjugate Block Parameters

Name	Value	Description
Operation	Conjugate, Inverse	Choose which operation to perform.

Table 4–9 shows the block I/O formats.

Table 4–9. Complex Conjugate Block I/O Formats Note (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	$O1_{Real([L1] + 1.[R1])Imag([L1] + 1.[R1])}$	O1Real: in STD_LOGIC_VECTOR({LP1 + RP1} DOWNTO 0) O1Imag: in STD_LOGIC_VECTOR({LP1 + RP1} DOWNTO 0)	Implicit

Notes to Table 4–9:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

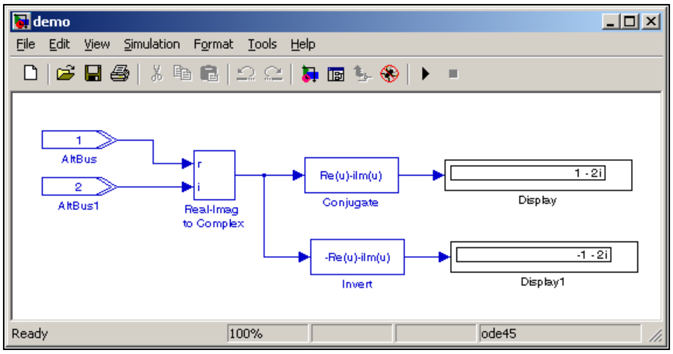
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 4–3 shows a design example using the Complex Conjugate block.

Figure 4–3. Complex Conjugate Block Example



Complex Delay Block

The Complex Delay block delays the incoming data by the amount specified by the `Depth` parameter. The input must be a complex number.

Table 4–10 shows the Complex Delay block parameters.

Table 4–10. Complex Delay Block Parameters

Name	Value	Description
Depth	User Defined	Indicate the delay length of the block.

Table 4–11 shows the block I/O formats.

Table 4–11. Complex Delay Block I/O Formats Note (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	$O1_{Real([L1].[R1])Imag([L1].[R1])}$	O1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) O1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Implicit

Notes to Table 4–11:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Complex Constant Block

The Complex Constant block outputs a fixed-point complex constant value.

Table 4–12 shows the Complex Constant block parameters.

Table 4–12. Complex Constant Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, or Unsigned Integer	Choose the number format of the bus.
[number of bits].[]	1 - 51	Indicate 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 - 51	Indicate the number of bits to the right of the binary point. This parameter only applies to signed fractional buses.
Real Value	User Defined	Indicate the value of the real part of the constant.
Imaginary Value	User Defined	Indicate the value of the real part of the constant.

Table 4–13 shows the block I/O formats.

Table 4–13. Complex Constant Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
O	$O1_{Real([L1].[R1])Imag([L1].[R1])}$	O1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) O1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Explicit

Notes to Table 4–13:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Complex to Real-Imag Block

The Complex to Real-Imag block constructs a fixed-point real and fixed-point imaginary output from a complex input.

Table 4–14 shows the Complex to Real-Imag block parameters.

Table 4–14. Complex to Real-Imag Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Choose the number format you wish to use for the bus.
Inputs [number of bits].[]	1 - 51	Select the number of data input bits to the left of the binary point, including the sign bit.
Inputs [].[number of bits]	0 - 51	Select the number of data input bits to the right of the binary point. This option is only available when **Signed Fractional** is selected.

Table 4–15 shows the block I/O formats.

Table 4–15. Complex to Real-Imag Block I/O Formats Note (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 4–15:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

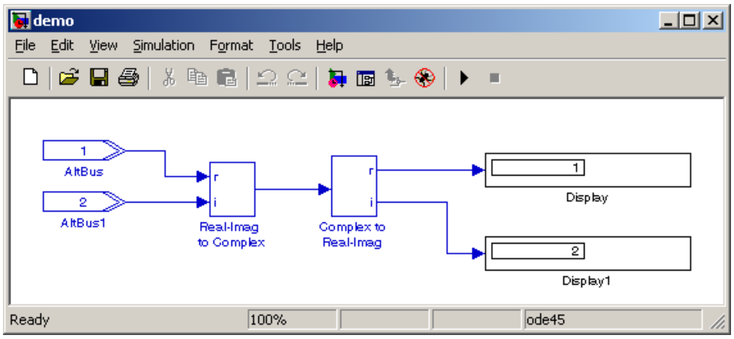
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 4–4 shows a design example using the Complex to Real-Imag block.

Figure 4–4. Complex to Real-Imag Block Example



Real-Imag to Complex Block

The Real-Imag to Complex block constructs a fixed-point complex output from real and imaginary inputs.

Table 4–16 shows the Real-Imag to Complex block parameters.

Table 4–16. Real-Imag to Complex Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Choose the number format you wish to use for the bus.
Inputs [number of bits].[]	1 - 51	Select the number of data input bits to the left of the binary point, including the sign bit.
Inputs [].[number of bits]	0 - 51	Select the number of data input bits to the right of the binary point. This option is only available when **Signed Fractional** is selected.

Table 4–17 shows the block I/O formats.

Table 4–17. Real-Imag to Complex Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{Real([L1].[R1])}$ $I2_{Imag([L1].[R1])}$	I1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) I1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Implicit
O	$O1_{Real([L1].[R1])Imag([L1].[R1])}$	O1Real: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0) O1Imag: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Explicit

Notes to Table 4–17:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Chapter 5. Boards Library

Stratix EP1S80 DSP Development Boards Library

The Stratix® EP1S80 DSP development Boards library is a prototyping platform that provides system designers with an economical solution for hardware and software verification. This rapid prototyping board enables the user to debug and verify both functionality and design timing. With two analog input and output channels per board and the ability to combine boards easily with right angle connectors, the board can be used to construct an extremely powerful processing system. Combined with DSP IP from Altera and the Altera® Megafunction Partners Program (AMPP™) partners, you can solve design problems that formerly required custom hardware and software solutions.

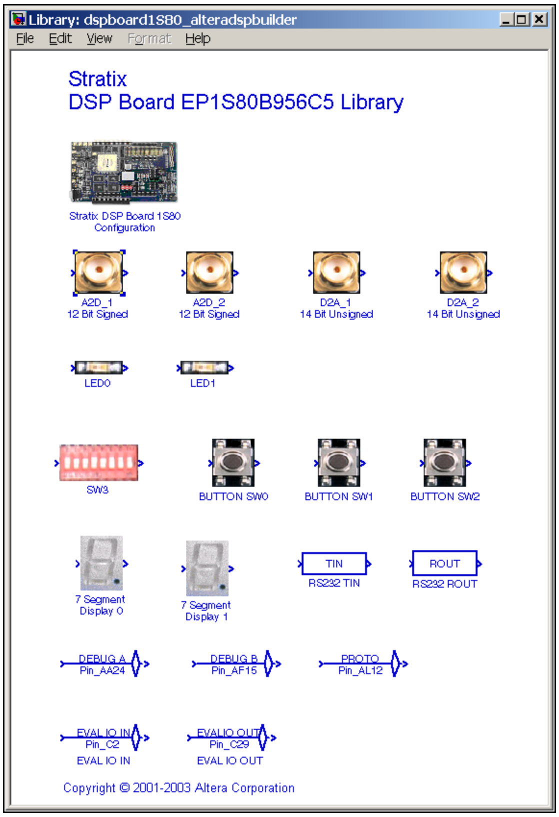
☞ For information on setting up the board and connecting it to your PC, see the *DSP Development Kit, Stratix & Stratix Professional Edition Getting Started User Guide*.

The blocks in this library provide a connection to analog-to-digital (A/D) converters, digital-to-analog (D/A) converters, LEDs, and switches on the Stratix EP1S80 DSP development board. With these blocks, you do not have to make pin assignments to connect to these board components. When targeting the Stratix EP1S80B956C5 development board, the design must contain the Stratix EP1S25 DSP development board configuration block at the top hierarchical level, to specify global clock and reset board connections. This library contains the following blocks (see Figure 5–1):

- Stratix EP1S80 DSP development board configuration
- A2D_1 12 Bit Signed—Controls A/D converter 1 (U10)
- A2D_2 12 Bit Signed—Controls A/D converter 2 (U30)
- D2A_1 14 Bit Unsigned—Controls D/A converter 1 (U21)
- D2A_2 14 Bit Unsigned—Controls D/A converter 2 (U23)
- LED0—Controls user LED 0 (D6)
- LED1—Controls user LED 1 (D7)
- BUTTON SW0—Controls push-button switch 0 (SW0)
- BUTTON SW1—Controls push-button switch 1 (SW1)
- BUTTON SW2—Controls push-button switch 2 (SW2)
- SW3—8 dipswitches (SW3)
- DEBUG_A—Controls debugging port A
- DEBUG_B—Controls debugging port B
- PROTO—Controls the prototyping area I/O
- EVALIO_IN
- EVALIO_OUT
- RS2323 TIN—RS-232 serial port transmit input

- RS2323 ROUT—RS-232 serial port receive output
- 7 Segment Display 0
- 7 Segment Display 1

Figure 5–1. Stratix DSP Board EP1S80 Blocks



Stratix EP1S25 DSP Development Boards Library

The Stratix EP1S25 DSP development Boards library is a prototyping platform that provides system designers with an economical solution for hardware and software verification. This rapid prototyping board enables the user to debug and verify both functionality and design timing. With two analog input and output channels per board and the ability to combine boards easily with right angle connectors, the board can be used to construct an extremely powerful processing system. Combined with DSP IP from Altera and AMPP™ partners, you can solve design problems that formerly required custom hardware and software solutions.

For information on setting up the board and connecting it to your PC, see the *DSP Development Kit, Stratix & Stratix Professional Edition Getting Started User Guide*.

The blocks in this library provide a connection to analog-to-digital (A/D) converters, digital-to-analog (D/A) converters, LEDs, and switches on the Stratix EP1S25 DSP development board. With these blocks, you do not have to make pin assignments to connect to these board components. When targeting the DSP development board EP1S25F780C5, the design must contain the Stratix EP1S25 DSP development board configuration block at the top hierarchical level, to specify global clock and reset board connections. This library contains the following blocks (see Figure 5–2):

- DSP Board EP1S25F780C5 Configuration—Board configuration
- A2D_0—Controls A/D converter 0 (J2)
- A2D_1—Controls A/D converter 1 (J3)
- D2A_0—Controls D/A converter 0 (J5)
- D2A_1—Controls D/A converter 1 (J6)
- LED0—Controls user LED 0 (D6)
- LED1—Controls user LED 1 (D7)
- LED2—Controls user LED 1 (D8)
- BUTTON SW0—Controls push-button switch 0 (SW0)
- BUTTON SW1—Controls push-button switch 1 (SW1)
- BUTTON SW2—Controls push-button switch 2 (SW2)
- SW3—8 dipswitches
- DEBUG_A—Controls debugging port A
- DEBUG_B—Controls debugging port B
- PROTO—Controls the prototyping area I/O
- EVALIO_IN
- EVALIO_OUT
- RS2323 TIN—RS-232 serial port transmit input
- RS2323 ROUT—RS-232 serial port receive output
- 7 Segment Display 0
- 7 Segment Display 1

Figure 5–2. Stratix EP1S25 DSP Board Blocks

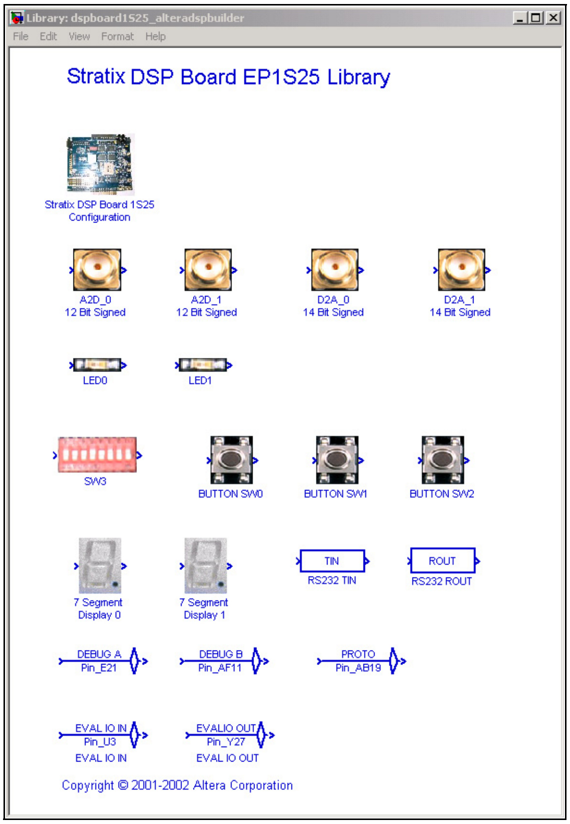
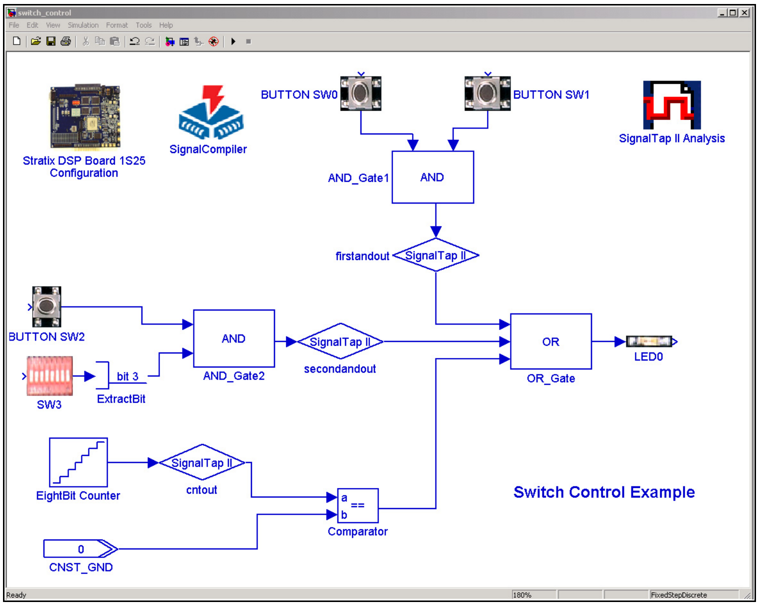


Figure 5–3 shows an example switch controller design using the SignalTap® II and board blocks. Based on the user-controlled switches and the value of the incrementer, an LED on the Stratix EP1S25 DSP development board turns on or off.

Figure 5–3. Switch Controller Using Stratix DSP Board Blocks



Stratix II EP2S60 DSP Development Boards Library

The Stratix II EP2S60 DSP development Boards library is a prototyping platform that provides system designers with an economical solution for hardware and software verification. This rapid prototyping board enables the user to debug and verify both functionality and design timing. With two analog input and output channels per board and the ability to combine boards easily with right angle connectors, the board can be used to construct an extremely powerful processing system. Combined with DSP IP from Altera and AMPP™ partners, you can solve design problems that formerly required custom hardware and software solutions.

☞ For information on setting up the board and connecting it to your PC, see the *DSP Development Kit, Stratix II Edition Getting Started User Guide*.

The blocks in this library provide a connection to analog-to-digital (A/D) converters, digital-to-analog (D/A) converters, LEDs, and switches on the Stratix II EP2S60 DSP development board. With these blocks, you do not have to make pin assignments to connect to these board components. When targeting the DSP development board EP2S60F1020C4, the design must contain the Stratix II EP2S60 DSP development board configuration block at the top hierarchical level, to specify global clock and reset board connections. This library contains the following blocks (see Figure 5–4):

- DSP Board EP2S60F1020C4 Configuration—Board configuration
- A2D_0—Controls A/D converter 0 (J1)

  ☞ Note: the A2D_0 is labeled A2D_A on the Stratix II EP2S60 DSP Board

- A2D_1—Controls A/D converter 1 (J2)
  Note: the A2D_1 is labeled A2D_B on the Stratix II EP2S60 DSP Board
- D2A_0—Controls D/A converter 0 (J15)
- D2A_1—Controls D/A converter 1 (J7)
- LED0 through LED7—Controls user LEDs (D1-D8)
- BUTTON SW0—Controls push-button switch 0 (SW4)
- BUTTON SW1—Controls push-button switch 1 (SW5)
- BUTTON SW2—Controls push-button switch 2 (SW6)
- BUTTON SW3—Controls push-button switch 2 (SW7)
- PROTO—2 Santa Cruz connectors controlling the prototyping area I/O (J23-J25, J26-J28)
- Mictor Connector (J20)
- ADI Connectors (J5, J6)
- 7 Segment Display 0
- 7 Segment Display 1

Figure 5–4. Stratix II EP2S60 DSP Board Blocks

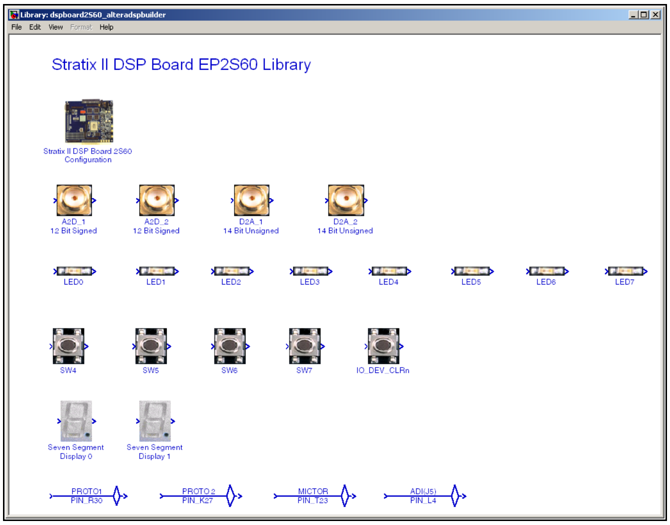
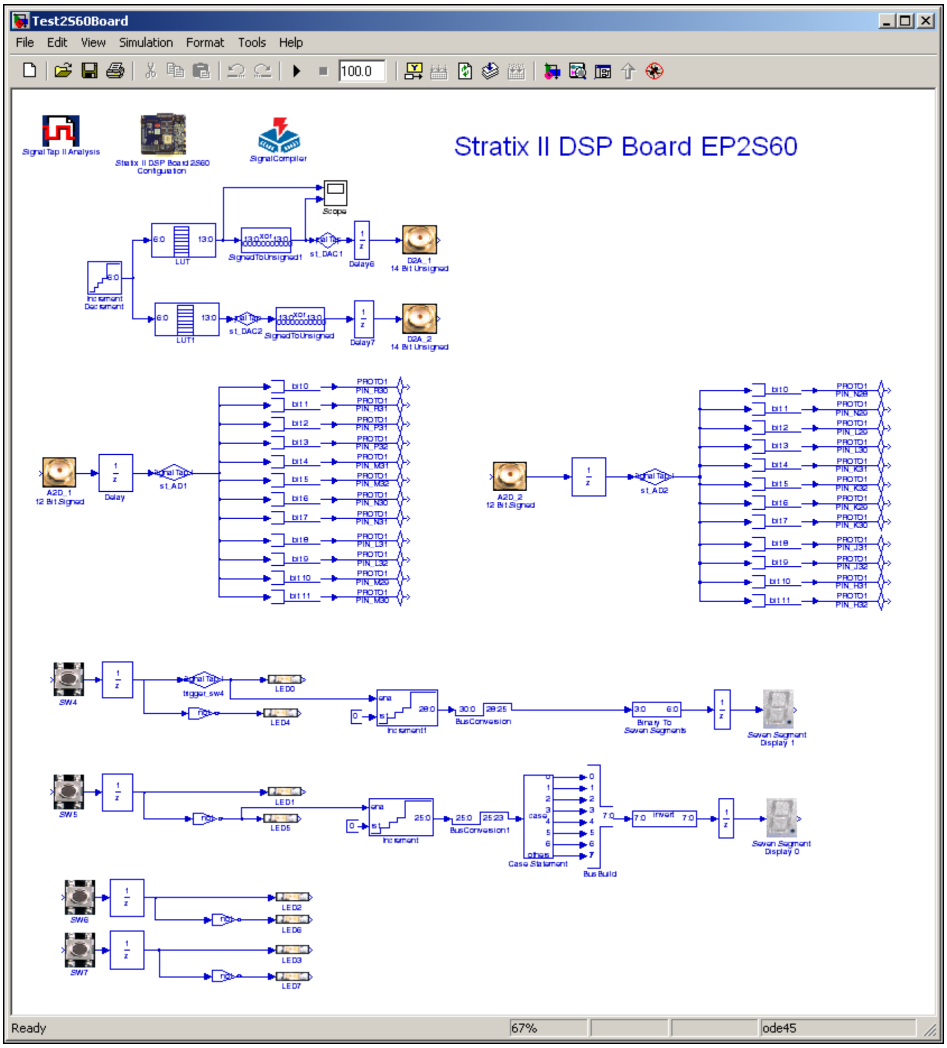


Figure 5–5 shows a test design using the SignalTap® II and EP2S60 DSP board blocks. The 7-segment display and 8 LEDs on the Stratix II EP2S60 DSP development board turn on or off in response to user-controlled switches and the value of the incrementer.

Figure 5–5. Test Design Using Stratix II DSP Board Blocks



Chapter 6. Gate & Control Library

Binary to Seven Segments Display Block

The Binary to Seven Segments Display block converts a 4-bit unsigned input bus into a 7-bit bus used for seven segment displays.

Table 6–1 shows the block I/O formats.

Table 6–1. Binary to Seven Segments Display Block I/O Formats Note (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(7 DOWNTO 0)	Explicit

Notes to Table 6–1:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Case Statement Block

This block contains boolean operators, which you can use for combinatorial functions.

The Case Statement block compares the input signal (which must be a signed or unsigned integer) with a set of case values. For each case, a single-bit output is generated. You can implement as many cases as you wish; append a comma (,) after each case. Additionally, you can have multiple conditions for each case; use a pipe (|) to separate the conditions. For example, for four cases with the first of which having two conditions, you would enter `1|2,3,4,5,` in the **Case Values** box. Table 6–2 shows the Case Statement block parameters.

Table 6–2. Case Statement Block Parameters

Name	Value	Description
Default Case	On or Off	Turn on this option if you want the `default` output signal to go high when the other outputs are false.
Case Values	User Specified	Indicate the values with which you want to compare the input. Include a comma after each value.
Register Outputs	On or off	Indicate whether you would like to register the output result.

Table 6–3 shows the block I/O formats.

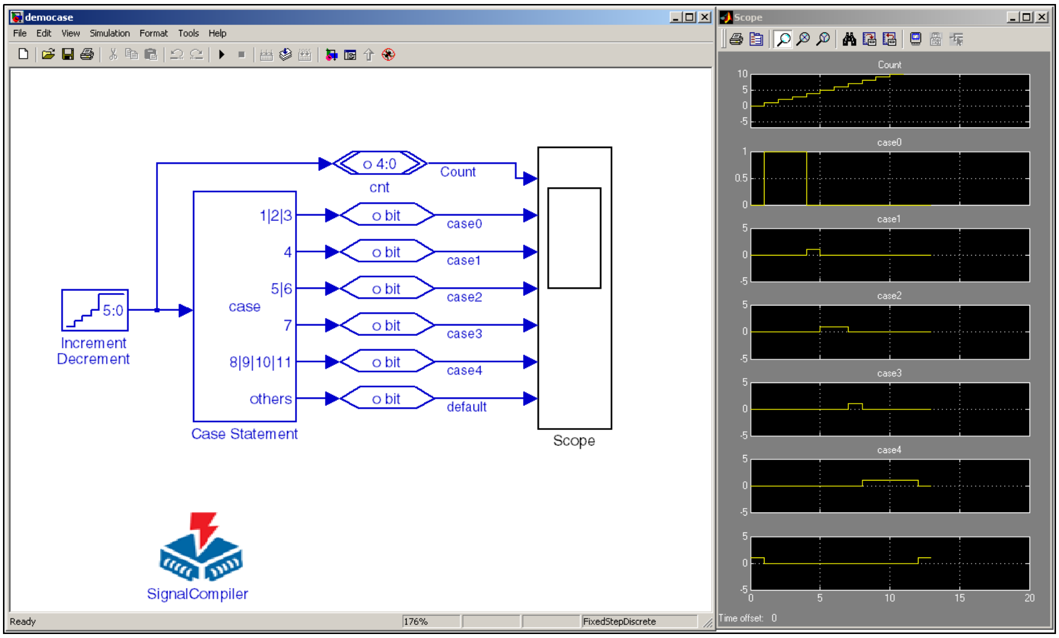
Table 6–3. Case Statement Block I/O Formats *Note (1)*

I/O	Simulink *(2), (3)*	VHDL	Type *(4)*
I	$I1_{[L1].[R1]}$	I1: in STD_LOGIC_VECTOR({LP1 + RP1 - 1} DOWNTO 0)	Implicit
O	$O1_{[1]}$ … $OUTi_{[1]}$ …. $OUTn_{[1]}$	O1: out STD_LOGIC … OUTi: out STD_LOGIC …. OUTn: out STD_LOGIC	Explicit

Notes to Table 6–3:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 6–1 shows an example model using the Case Statement block.

Figure 6–1. Case Statement Block Example



The following code is an excerpt of the VHDL that SignalCompiler generates from the model:

```
p0:process(A1W)
    begin
        case A1W is
            when "000001"|"000010"|"000011" =>
                    A0W <= '1';
                    A2W <= '0';
                    A6W <= '0';
                    A5W <= '0';
                    A4W <= '0';
                    A3W <= '0';
            when "000100" =>
                    A0W <= '0';
                    A2W <= '1';
                    A6W <= '0';
                    A5W <= '0';
                    A4W <= '0';
                    A3W <= '0';
.
.
.
```

```
            when "001000"|"001001"|"001010"|"001011" =>
                    A0W <= '0';
                    A2W <= '0';
                    A6W <= '0';
                    A5W <= '0';
                    A4W <= '1';
                    A3W <= '0';
            when others=>
                    A0W <= '0';
                    A2W <= '0';
                    A6W <= '0';
                    A5W <= '0';
                    A4W <= '0';
                    A3W <= '1';
        end case;
end process;
```

☞ In Simulink, each wire is named `A`<*number*>`W` where <*number*> is auto-generated.

Decoder Block

The Decoder block is a bus decoder that reads the input and outputs as a single bit. The outputs returns a one when the data input equals the decoded value. Table 6–4 shows the block parameters.

Table 6–4. Decoder Block Parameters

Name	Value	Description
Input Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Indicate the bus number format that you want to use for the counter.
[number of bits].[]	1 to 51	Select the number of bits to the left of the binary point.
[].[number of bits]	0 to 51	Select the number of bits to the right of the binary point for the gain. This option is zero (0) unless Signed Fractional is selected.
Decoded Value	User defined	Specify the decoded value.
Register Outputs	On or off	Indicate whether you would like to register the output result.

Table 6–5 shows the block I/O formats.

Table 6–5. Decoder Block I/O Formats *Note (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 6–5:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Flip Flop Block

The Flip Flop block is a 1-bit DFFE or TFFE. In DFFE mode:

```
if (0 == clrn)    q = 0;
else if (0 == prn)    q = 1;
else if (1 == ena)    q = d
```

In TFFE mode,

```
if (0 == clrn)    q = 0;
else if (0 == prn)    q = 1;
else if (1 == ena) and (1 == T)    q = toggle
```

(`clrn == 0`) and (`prn == 0`) are not supported.

Table 6–6 shows the Flip Flop block parameters.

Table 6–6. Flip Flop Block Parameters

Name	Value	Description
Mode	DFFE or TFFE	Indicate which type of flip flop to implement.

Table 6–7 shows the block I/O formats.

Table 6–7. Flip Flop Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[1].[0]}$ $I2_{[1].[0]}$ $I3_{[1].[0]}$	I1: in STD_LOGIC I2: in STD_LOGIC I3: in STD_LOGIC	Explicit
O	$O1_{[1].[0]}$	O1: in STD_LOGIC	Explicit

Notes to Table 6–7:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

If Statement Block

The If Statement block returns a boolean result based on the IF condition equation. The input arguments—*a, b, c, d, e, f, g, h, i,* or *j*—must be signed or unsigned integers. You can use any number of parentheses. The operators are given in Table 6–8.

Table 6–8. Supported If Statement Block Operators

Operator	Operation
+	OR
&	AND
$	XOR
=	Equal To
~	Not Equal To
>	Greater Than
<	Less Than

Table 6–9 shows the If Statement block parameters.

Table 6–9. If Statement Block Parameters

Name	Value	Description
Number of Inputs	2 - 10	Indicate the number of inputs to the If Statement.
IF	User Defined	Indicate the if condition using any of the following operators: +, &, $, =, ~, >, or < and the variables *a, b, c, d, e, f, g, h, i,* or *j*.
ELSE Output	On or Off	This option turns on the `false` output signal, which goes high if the condition evaluated by the If Statement is false.
ELSE IF Input	On or Off	This option enables the `n` input signal (where `n` is the ELSE IF input), which you can use to cascade multiple IF Statement blocks together.

Table 6–10 shows the block I/O formats.

Table 6–10. If Statement Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	I1$_{[L1].[R1]}$ .... INPi$_{[LI].[RI]}$ ... INPn$_{[LN].[RN]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) … INPi: in STD_LOGIC_VECTOR({LI + RI - 1} DOWNTO 0) …. INPn: 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 6–10:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

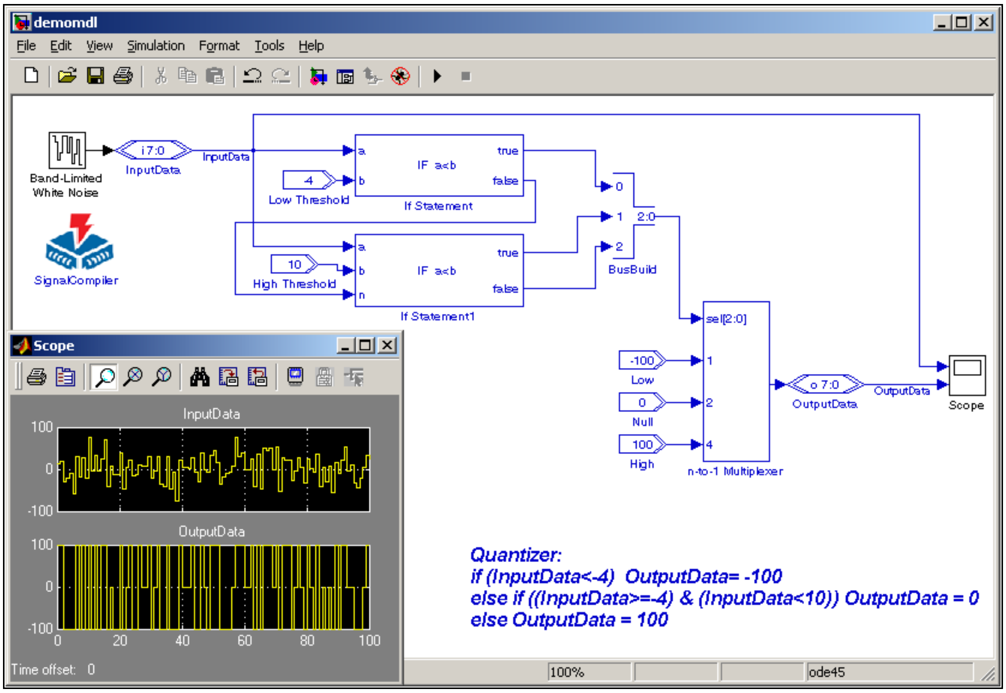
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 6–2 shows an example using the If Statement block.

Figure 6–2. If Statement Block Example



Quantizer:
if (InputData<-4) OutputData= -100
else if ((InputData>=-4) & (InputData<10)) OutputData = 0
else OutputData = 100

Logical Bit Operator Block

This block performs logical operation 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 6–11 shows the block parameters.

Table 6–11. Logical Bit Operator Block Parameters

Name	Value	Description
Logical Operator	AND, OR, XOR, NAND, NOR, or NOT	Indicate which operator you wish to use.
Number of Inputs	1 - 16	Indicate the number of inputs. This parameter defaults to 1 if the NOT logical operator is selected.

Table 6–12 shows the block I/O formats.

Table 6–12. Logical Bit Operator Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	I1[1] … INPi[1] …. INPn[1]	I1: in STD_LOGIC … INPi: in STD_LOGIC …. INPn: in STD_LOGIC	Explicit
O	O1[1]	O1: out STD_LOGIC	Explicit

Notes to Table 6–12:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

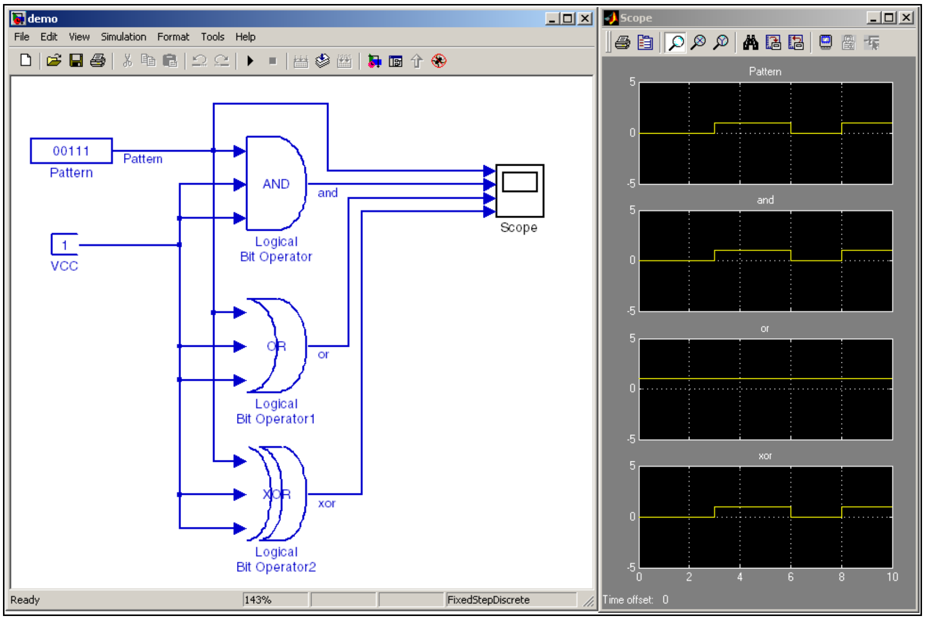
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 6–3 shows an example using the Logical Bit Operator block.

Figure 6–3. Logical Bit Operator Block Example



Logical Bus Operator Block

This block performs logical operations on a bus such as AND, OR, XOR, or invert. You can perform masking by entering in decimal notation or shifting. Note that the shifting operation is a logical shift and not a arithmetic shift. that is, the shifting operation does not preserve the input data sign. Table 6–13 shows the block parameters.

Table 6–13. Logical Bus Operator Block Parameters

Name	Value	Description
[number of bits].[]	1 - 51	Indicate the number of bits to the left of the binary point, including the sign bit.
[].[number of bits]	0 - 51	Indicate the number of bits to the right of the binary point.
Logic Operation	AND, OR, XOR, Invert, Shift Left, Shift Right, Rotate Left, Rotate Right	Indicate the logical operation to perform.
Number of Bits to Shift	User Defined	Indicate how many bits you wish to shift.

Table 6–14 shows the block I/O formats.

Table 6–14. Logical Bus Operator Block I/O Formats Note (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: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit

Notes to Table 6–14:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

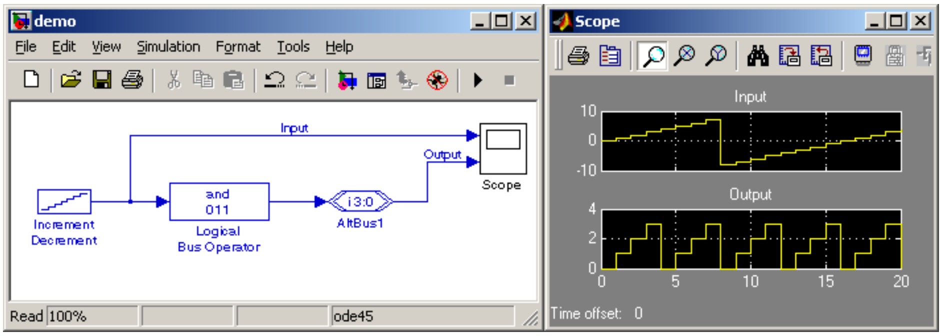
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 6–4 shows an example using the Logical Bus Operator block.

Figure 6–4. Logical Bus Operator Block Example



Bitwise Logical Bus Operator Block

This block performs logical operations such as AND, OR, or XOR across two buses. Table 6–15 shows the block parameters.

Table 6–15. Bitwise Logical Bus Operator Block Parameters

Name	Value	Description
Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Indicate the bus number format that you want to use for the counter.
[number of bits].[]	1 - 51	Indicate the number of bits to the left of the binary point, including the sign bit.
Input [].[number of bits]	0 - 51	Indicate the number of bits to the right of the binary point.
Logic Operation	AND, OR, XOR	Indicate the logical operation to perform.

Table 6–16 shows the block I/O formats.

Table 6–16. Bitwise Logical Bus Operator Block I/O Formats Note (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: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit

Notes to Table 6–16:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

LUT Block

The LUT block stores data as 2 $^{(address\ width)}$ 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)/TriMatrix memory. You should use a manual synthesis and compilation flow (see *Manual Synthesis & Compilation* in the *DSP Builder Tutorial* chapter in the *DSP Builder User Guide*) if you want to control the memory implementation.

☞ If you want to use a Hexadecimal File (**.hex**) to store data, use the ROM EAB block not the LUT block.

Table 6–17 shows the block parameters.

Table 6–17. LUT Block Parameters

Name	Value	Description
Output Data [number of bits].[]	1 - 51	Indicate the number of data bits stored on the left side of the binary point including the sign bit.
Output Data [].[number of bits]	0 - 51	Indicate the number of data bits stored on the right side of the binary point.
Address Width	2 - 16	Indicate the address width.
MATLAB Array	User Defined	This field must be a one-dimensional MATLAB array with a length smaller than 2 to the power of the address width. You can specify the array in the MATLAB **Command Window** using a variable, or you can specify it directly in the **MATLAB Array** box.
Use LPM	On or Off	If you turn on the **Use LPM** option, SignalCompiler will implement the look-up table using the `lpm_rom` library of parameterized modules (LPM) function. If you turn off this option, SignalCompiler implements the look-up table using Case conditions. Altera recommends that you turn on the **Use LPM** option for large look-up tables, for example, greater than 8 bits.
Register Address	On of Off	When register address is on, the input address bus is generated. If you are targeting either Stratix™ II, Stratix, Stratix GX, or Cyclone™ devices and using the Use LPM option, you must turn on the register address option.
Register Outputs	On or off	Indicate whether you would like to register the output result.

Table 6–18 shows the block I/O formats.

Table 6–18. LUT Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	I1$_{[L1].[0]}$	I1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0)	Explicit
O	O1$_{[LPO].[RPO]}$	O1: out STD_LOGIC_VECTOR({LPO + LPO - 1} DOWNTO 0)	

Notes to Table 6–18:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

1-to-n Demultiplexer Block

The 1-to-n Demultiplexer block uses full encoded binary values. The `sel` input is an unsigned bus.

Table 6–19 describes the parameters for the 1-to-n Demultiplexer block.

Table 6–19. 1-to-n Demultiplexer Block Parameters

Name	Value	Description
Number of Output Data Lines	An integer greater than 1	Choose how many outputs you want the demultiplexer to have.
Use Control Inputs	On or Off	Indicate whether you want to use additional control inputs (clock enable and reset).

Table 6–20 shows the block I/O formats.

Table 6–20. 1-to-n Demultiplexer Block I/O Formats Note (2)

I/O	Simulink (3), (4)	VHDL	Type (5)
I	I[L].[R[	I1: in STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0) SEL: in STD_LOGIC_VECTOR({L - 1} DOWNTO 0)	Implicit Explicit
O	OI[L].[R] ... On[L].[R] (1)	O: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0) ... On: out STD_LOGIC_VECTOR({L + R - 1} DOWNTO 0)	Implicit

Notes to Table 6–20:

(1) Where I is the number of outputs to the demultiplexer.

(2) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

n-to-1 Multiplexer Block

The n-to-1 Multiplexer block is a *n*-to-1 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 6–21 shows the block parameters.

Table 6–21. n-to-1 Multiplexer Block Parameters

Name	Value	Description
Number of Input Data Lines	an integer greater than 1	Choose how many inputs you want the multiplexer to have.
One Hot Select Bus	On or Off	Indicate whether you want to use one-hot selection for the bus select signal instead of full binary.
Pipeline	0-6	Specify the number of levels of pipeline
Use Control Inputs	On or Off	Indicate whether you would like to use additional control inputs (clock enable and reset).

Table 6–22 shows the block I/O formats.

Table 6–22. n-to-1 Multiplexer Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$IS_{[LS].[0]}$ (select input) $I1_{[L1].[R1]}$ .... $Ii_{[LI].[RI]}$ ... $In_{[LN].[RN]}$ (n = number of inputs)	IS: in STD_LOGIC_VECTOR({LS - 1} DOWNTO 0) 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	$O1_{[max(LI)].[max(RI)]}$ with (0 < I < N + 1)	O1: out STD_LOGIC_VECTOR({max(LI)) + max(RI) - 1} DOWNTO 0)	Implicit

Notes to Table 6–22:

(1) For signed integers and signed binary fractional numbers, the MSB is the sign bit.

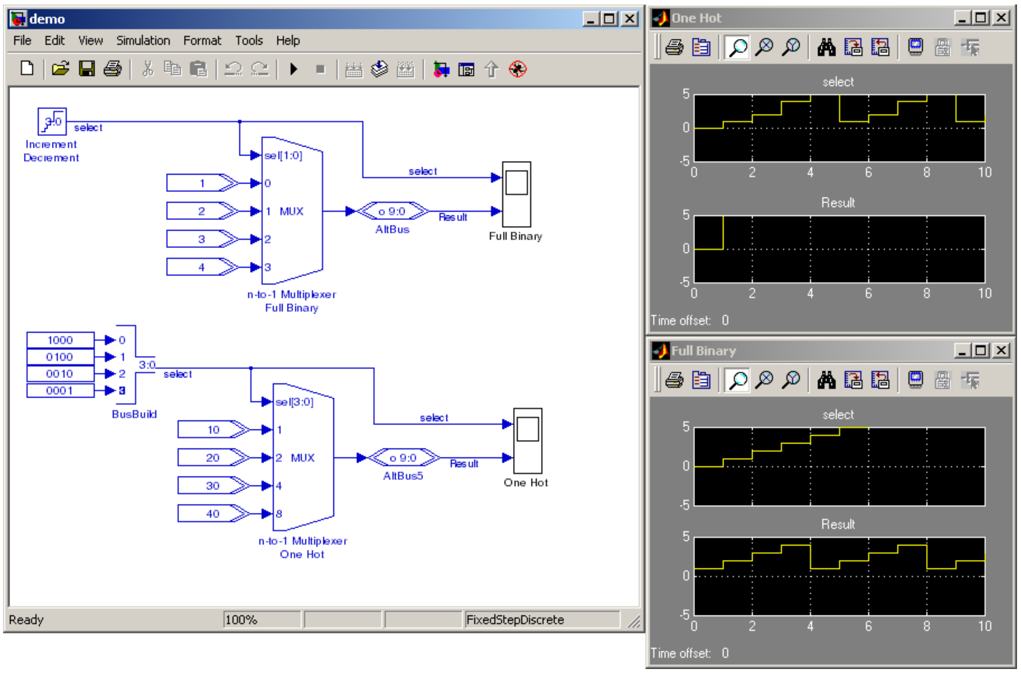
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 6–5 shows an example using the n-to-1 Multiplexer block.

Figure 6–5. n-to-1 Multiplexer Block Example



NOT Block

The NOT block is a single-bit inverter.

Table 6–23 shows the block I/O formats.

Table 6–23. n-to-1 Multiplexer Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[1].[0]}$	I1: in STD_LOGIC	Explicit
O	$O1_{[1].[0]}$	O1: out STD_LOGIC	Explicit

Notes to Table 6–23:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Single Pulse Block

This 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 could be an impulse, a step (0 to 1), or a step (1 to 0) as shown in Figure 6–6. Table 6–24 shows the block parameters.

Table 6–24. Single Pulse Block Parameters

Name	Value	Description
Signal Generation Type	Step (0 to 1) or Step (1 to 0) or Impulse	Select the type of single pulse.
Use Control Inputs	On or Off	Indicate whether you would like to use additional control inputs: Trigger: trigger the signal generation when equal to one. rst: reset the output.
Step Delay	Integer	Specify the number of sampling periods which occur before the step transition. This parameter is only valid when the Signal Generation Type is set to Step (0 to 1) or Step (1 to 0).
Impulse Width	Integer	Specify the number of sampling periods for which the output signal is transitional from 0 to 1. This parameter is only valid when the Signal Generation Type is set to Impulse.
Impulse Delay	Integer	Specify the number of sampling periods for which the output signal is high. This parameter is only valid when the Signal Generation Type is set to Impulse.
Sampling Period	Integer	Specify the sample time period in seconds.

Table 6–25 shows the block I/O formats.

Table 6–25. Single Pulse Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[1]}$ $I2_{[1]}$	I1: in STD_LOGIC I2: in STD_LOGIC	Explicit optional Explicit optional
O	$O1_{[1]}$	O1: out STD_LOGIC	Explicit

Notes to Table 6–25:

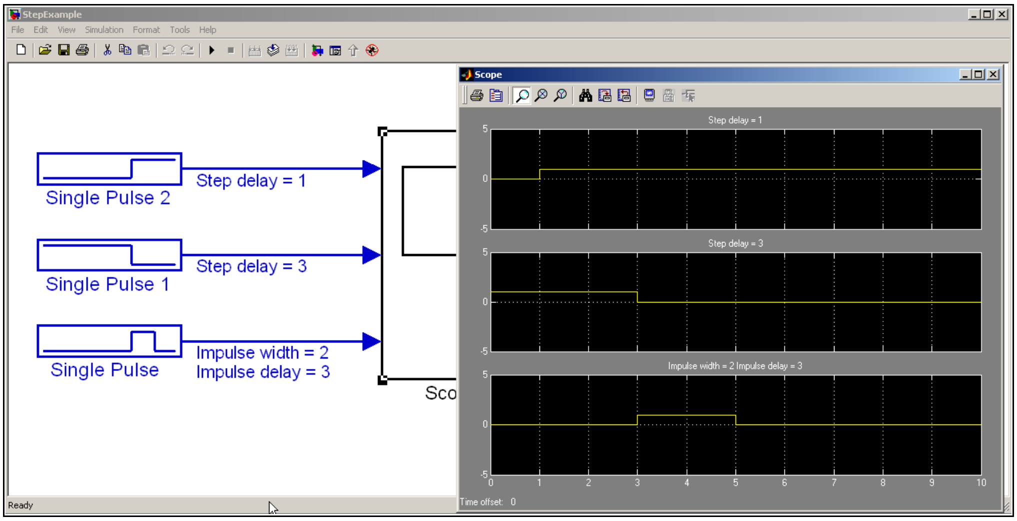
(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 6–6. Single Pulse Output Signal Types



Chapter 7. Rate Change Library

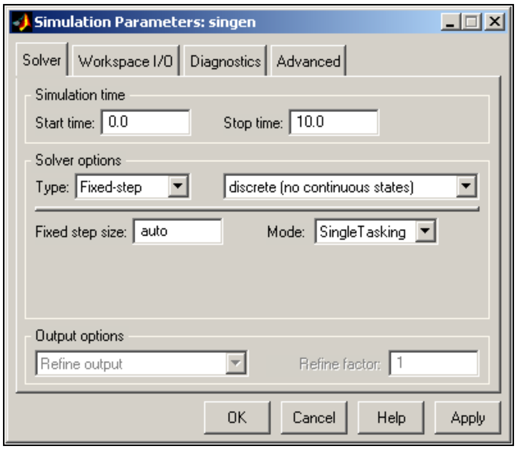
Introduction

This library contains the blocks that allow you to control the clock assignment to registered DSP Builder blocks, such as Delay or Increment Decrement blocks.

To maintain cycle accuracy between Simulink and the VHDL domain, make the following settings in the **Solver** tab (in Figure 7–1) of the Simulink block:

- Choose **Fixed-step** from the **Type** list under **Solver options**.
- Choose **discrete (no continuous state)** under **Solver options**.
- Choose **Single Tasking** from the **Mode** list.

Figure 7–1. Set the Simulation Parameters


Simulation Parameters: singen

Simulation time
Start time: 0.0
Stop time: 10.0

Solver options
Type: Fixed-step
discrete (no continuous states)
Fixed step size: auto
Mode: SingleTasking

Output options
Refine output
Refine factor: 1

ClockAltr Block

Use the ClockAltr block to set additional hardware clock domains. To create new clock domains when the source clock is set to Input Pin:

- If the clock divider ratio is equal to 1, the ClockAltr block creates one additional clock domain. The Simulink variable name is the same as the block name.
- If the clock divider ratio is greater than 1, the ClockAltr block creates two additional clock domains. The Simulink variable name is <*block name*> **_div** when there are two additional clock domains.

The input clock period must be positive and non-null. Specify its value in seconds.

When the source clock is set to `Pll output clock`<*number*> ClockAltr creates one additional clock domain if the clock divider ratio is greater than 1. The Simulink variable name is <*block name*> **_div**.

Table 7–1 lists the parameters for the **ClockAltr** block:

Table 7–1. ClockAltr Block Parameters

Name	Value	Description
Source	Input Pin, PLL output Clock 0, (through 5)	Clock source, either directly from a pin or from a PLL output.
Input Clock Period Value (second)	Double	Period value specified in seconds.
Addition Clock divider	Integer	Clock divider ratio value.

PLL Block

DSP Builder uses the PLL block to synthesize a clock signal that is based on a reference clock. PLLs 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. Cyclone™ II, Cyclone, Stratix® II, Stratix, and Stratix GX device families offer the most advanced on-chip PLL features, which 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, which gives you complete control over your clocking needs. The PLLs offer full frequency synthesis capability (the ability to multiply up or divide down the clock frequency) 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 that operate at frequencies that are multiples of the frequency of the system clock. Cyclone II, Cyclone, Stratix II, Stratix, and Stratix GX PLLs can simultaneously multiply and divide the reference clock. The PLL block checks the validity of the parameters.

When you add the PLL block to your Simulink design, DSP Builder creates global MATLAB variables in the base MATLAB workspace. For example:

Global Variable Name	= Period value of PLL output clock
pll_output_clock0	= 6.666e-009
pll_output_clock1	= 4.9995e-007

You can reuse these global variables to specify the sampling period of DSP Builder blocks, such as Constant or Increment Decrement blocks. When you change the PLL parameters, these global variables, as well as the sampling period of the blocks using these variables, are updated dynamically.

The PLL block supports the Cyclone II, Cyclone, Stratix II, Stratix, and Stratix GX device families.

For more information on the built-in PLLs, see the device handbook for the device family you are targeting.

When you add the PLL block to your design, SignalCompiler has an additional option, **PLL Options**, on the **Main Clock** tab under **Project Setting Options**. You use this option to specify whether to keep the clock internal or whether to output it to the pins.

Table 7–2 shows the number of clock outputs that are available for each family.

Table 7–2. Device Clock Outputs

Device	Number of Clock Outputs
Stratix, Stratix II, Stratix GX devices	6
Cyclone, Cyclone II devices	2

If you use the PLL block, the following restrictions apply:

- The design must contain a single PLL instance at the top level
- Each output clock of the PLL has a zero degree phase shift and 50% duty cycle
- All DSP Builder block Simulink sample times must equal one of the PLL's output clock periods

Table 7–3 shows the PLL block parameters.

Table 7–3. PLL Block Parameters

Name	Value	Description
Input Clock Frequency	*(1)*	Input reference clock.
Number of Output Clocks	1 to 6	Number of PLL clock outputs.
Clock Frequency multiplication factor	*(1)*	Multiply the reference clock by this value.
Clock Frequency division factor	*(1)*	Divide the reference clock by this value.

Note to Table 7–3:

(1) See the device documentation for the device family you are targeting.

Multi-Rate DFF Block

DSP Builder uses the multi-rate DFF block to synchronize data path intersections involving multiple rates.

Table 7–4 shows the multi-rate DFF block parameters.

Table 7–4. Multi-Rate DFF Block Parameters

Name	Value	Description
Clock Source	1 to 6	Specify PLL output clock source to the clock pin of the multi-rate DFF block. When the design does not contain PLLs, DSP Builder connects the multi-rate DFF block clock pin to the main single clock.

Table 7–5 shows the block I/O formats.

Table 7–5. Multi-Rate DFF Multiplexer Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L].[R]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit
O	$O1_{[L].[R]}$	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit

Notes to Table 7–5:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Tsamp Block

DSP Builder uses the Tsamp block to specify the sample time period of the internal data path source.

Table 7–6 shows the Tsamp block parameters.

Table 7–6. Tsamp Block Parameters

Name	Value	Description
Sample Time Period	Any	Specified the sample time period in seconds.

Table 7–7 shows the block I/O formats.

Table 7–7. Tsamp Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L].[R]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit
O	$O1_{[L].[R]}$	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Implicit

Notes to Table 7–7:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

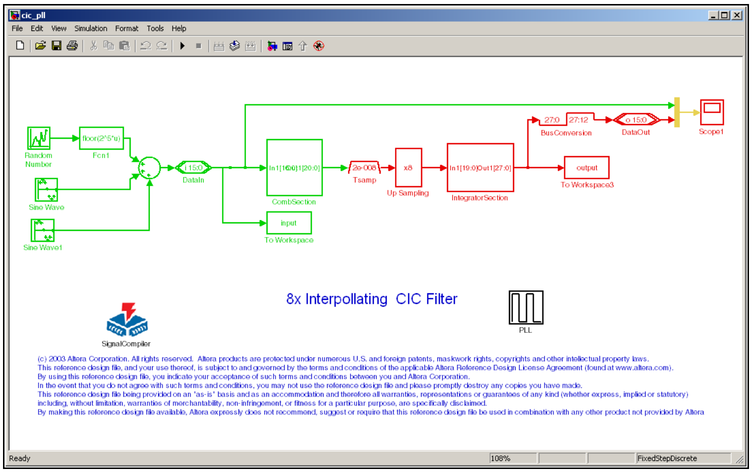
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 7–2 shows an example design with the Tsamp block. The design is available in the <*path*>**\DSPBuilder\designexamples\Demos\Filters\CicFilter** directory.

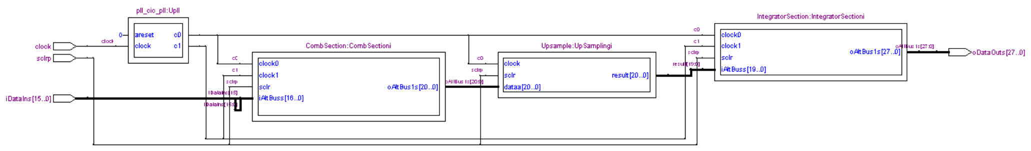
Figure 7–2. Tsamp Block Example



(c) 2003 Altera Corporation. All rights reserved. Altera products are protected under numerous U.S. and foreign patents, maskwork rights, copyrights and other intellectual property laws.
This reference design file, and your use thereof, is subject to and governed by the terms and conditions of the applicable Altera Reference Design License Agreement (found at www.altera.com).
By using this reference design file, you indicate your acceptance of such terms and conditions between you and Altera Corporation.
In the event that you do not agree with such terms and conditions, you may not use the reference design file and please promptly destroy any copies you have made.
This reference design file being provided on an 'as-is' basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.
By making this reference design file available, Altera expressly does not recommend, suggest or require that this reference design file be used in combination with any other product not provided by Altera

Figure 7–3 shows the Quartus® II Tsamp block example design in the LeonardoSpectrum™ software.

Figure 7–3. Representation of the Tsamp Block Example in Quartus II RTL Viewer



Chapter 8. SOPC Builder Links Library

The Nios® II embedded processor is an industry-leading soft-core embedded processor optimized for the high-performance, high-bandwidth needs of network, telecommunications, and mass storage applications. The Nios II embedded processor includes a RISC CPU and the industry-standard GNUPro software from Cygnus, a Red Hat company.

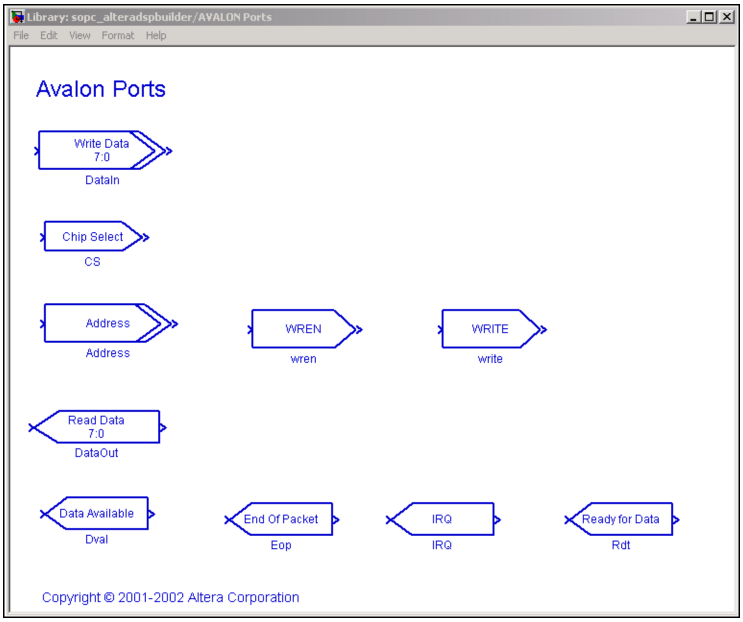
You can use the blocks in the SOPC Ports library to build a custom logic block that works with the SOPC Builder and optionally Nios II embedded processor designs. The SOPC Builder supports two general types of custom logic blocks:

- Peripherals that use the Avalon™ bus
- Custom instructions that extend the function of the Nios II arithmetic logic unit (ALU) and instruction set

Avalon Ports

You use the blocks in the Avalon Ports library to build Nios II peripherals that connect to the Avalon bus. The Avalon Ports blocks automate the process of specifying slave ports that are compatible with the Avalon bus. After you build a model of your DSP Builder peripheral, add blocks from the Avalon Ports library to control the peripheral's inputs and outputs. Figure 8–1 shows the blocks in the Avalon Ports library.

Figure 8–1. Avalon Ports Library



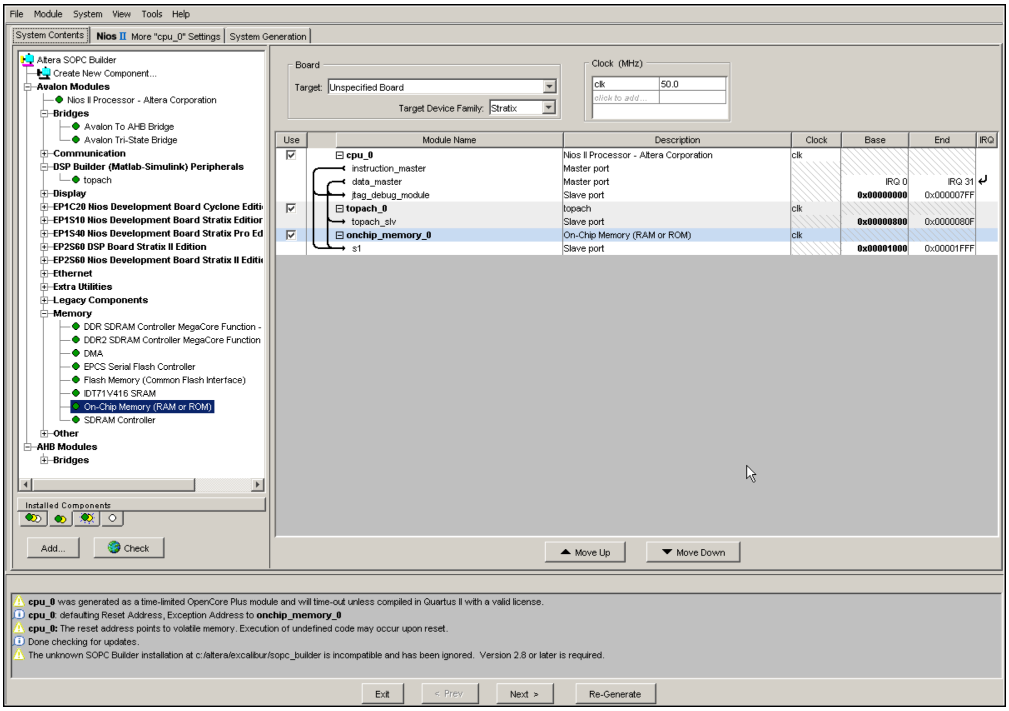
When you convert your model to VHDL, SignalCompiler creates a file, **class.ptf**, in your working directory. The **class.ptf** file is a text file that describes:

- Parameters that define the peripheral's structure and/or functionality
- The peripheral's slave role
- The peripheral's ports (such as read enable, read data, write enable, write data)
- The arbitration mechanism for each slave port that can be accessed by multiple master ports

For more information on the Avalon bus and PTF files, see the *Avalon Bus Specification Reference Manual*.

After you synthesize your model and compile it in the Quartus® II software, you can add it to your Nios II system using the SOPC Builder. Your peripheral appears in the SOPC Builder peripherals listing. Figure 8–2 shows the SOPC Builder with a Nios II CPU, a DSP Builder created peripheral named **top**, and a DMA peripheral that connects to **top**.

Figure 8–2. SOPC Builder with DSP Builder Peripheral


File Module System View Tools Help
System Contents
Nios II More "cpu_0" Settings
System Generation
Altera SOPC Builder
Create New Component...
Avalon Modules
Nios II Processor - Altera Corporation
Bridges
Avalon To AHB Bridge
Avalon Tri-State Bridge
Communication
DSP Builder (Matlab-Simulink) Peripherals
topach
Display
EP1C20 Nios Development Board Cyclone Editi
EP1S10 Nios Development Board Stratix Editior
EP1S40 Nios Development Board Stratix Pro Ed
EP2S60 DSP Board Stratix II Edition
EP2S60 Nios Development Board Stratix II Editi
Ethernet
Extra Utilities
Legacy Components
Memory
DDR SDRAM Controller MegaCore Function -
DDR2 SDRAM Controller MegaCore Function
DMA
EPCS Serial Flash Controller
Flash Memory (Common Flash Interface)
IDT71V416 SRAM
On-Chip Memory (RAM or ROM)
SDRAM Controller
Other
AHB Modules
Bridges
Installed Components
Add...
Check
Move Up
Move Down
Exit
< Prev
Next >
Re-Generate

Board

Target: Unspecified Board

Target Device Family: Stratix

Clock (MHz)

clk	50.0
click to add ...	

Use	Module Name	Description	Clock	Base	End	IRQ
✓	cpu_0	Nios II Processor - Altera Corporation	clk			
	instruction_master	Master port				
	data_master	Master port		IRQ 0	IRQ 31	
	jtag_debug_module	Slave port		0x00000000	0x000007FF	
✓	topach_0	topach	clk			
	topach_slv	Slave port		0x00000800	0x0000080F	
✓	onchip_memory_0	On-Chip Memory (RAM or ROM)	clk			
	s1	Slave port		0x00001000	0x00001FFF	

```
⚠ cpu_0 was generated as a time-limited OpenCore Plus module and will time-out unless compiled in Quartus II with a valid license.
ⓘ cpu_0: defaulting Reset Address, Exception Address to onchip_memory_0
⚠ cpu_0: The reset address points to volatile memory. Execution of undefined code may occur upon reset.
ⓘ Done checking for updates.
⚠ The unknown SOPC Builder installation at c:/altera/excalibur/sopc_builder is incompatible and has been ignored.  Version 2.8 or later is required.
```

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

See the *Nios II Hardware Development Tutorial* for information on using the SOPC Builder to create Nios II designs.

Figure 8–3 shows the design flow using DSP Builder and SOPC Builder.

Figure 8–3. DSP Builder & SOPC Builder Design Flow

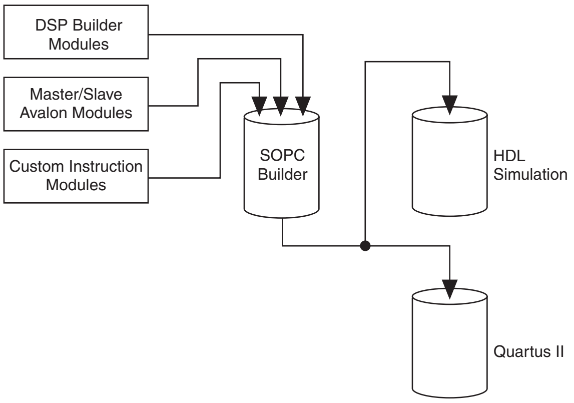
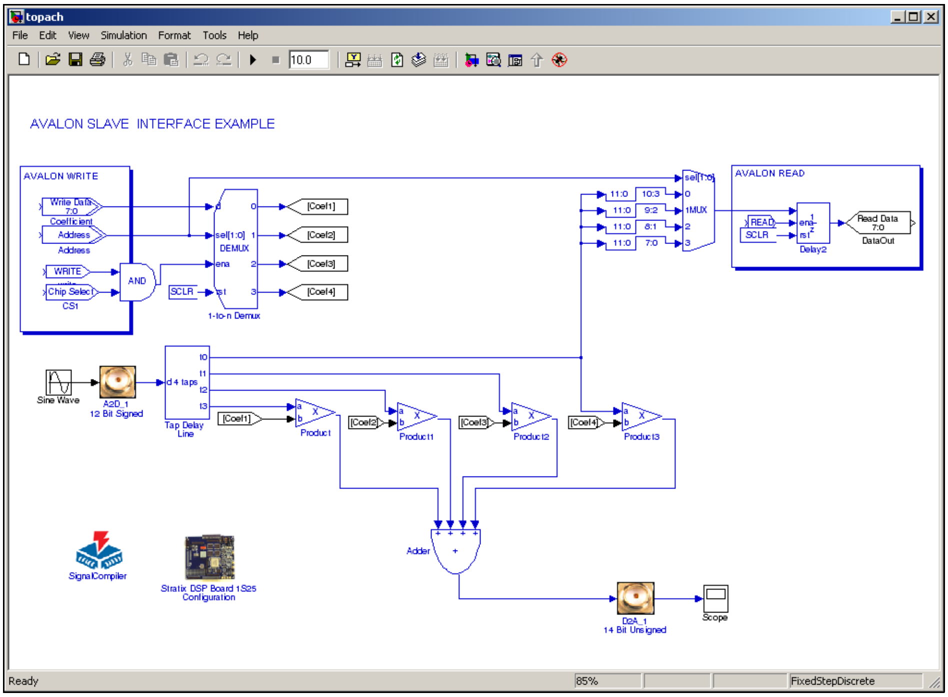


Figure 8–4 shows an example model using blocks from the Avalon Ports library.

Figure 8–4. Avalon Ports Library Example

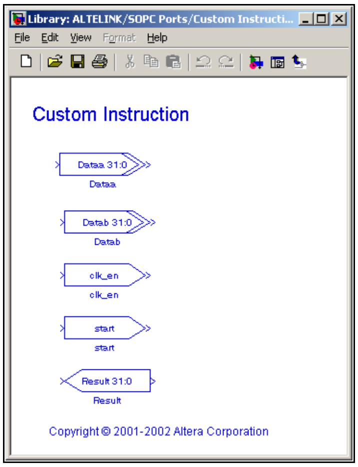


Custom Instruction

You use the blocks in the Custom Instruction library to build custom instructions for Nios II systems. The Custom Instruction blocks automate the process of specifying ports that are compatible with SOPC Builder: the blocks correspond to the custom instruction ports used by the SOPC Builder. After you build a model of your custom instruction, add blocks from the Custom Instruction library to control the instruction's inputs and outputs. Figure 8–5 shows the blocks in the Custom Instruction library.

DSP Builder does not support the Nios II custom instruction `prefix` port. All other ports are supported.

Figure 8–5. Custom Instruction Library


Library: ALTELINK/SOPC Ports/Custom Instructi...
File Edit View Format Help
Custom Instruction
Dataa 31:0
Dataa
Datab 31:0
Datab
clk_en
clk_en
start
start
Result 31:0
Result
Copyright © 2001-2002 Altera Corporation

For non-combinatorial custom instructions, you must include the start and `clk_en` blocks as inputs at the top level

With custom instructions, system designers can add custom-defined functionality to the Nios II embedded processor's arithmetic logic unit (ALU) and instruction set. Custom instructions consist of two essential elements:

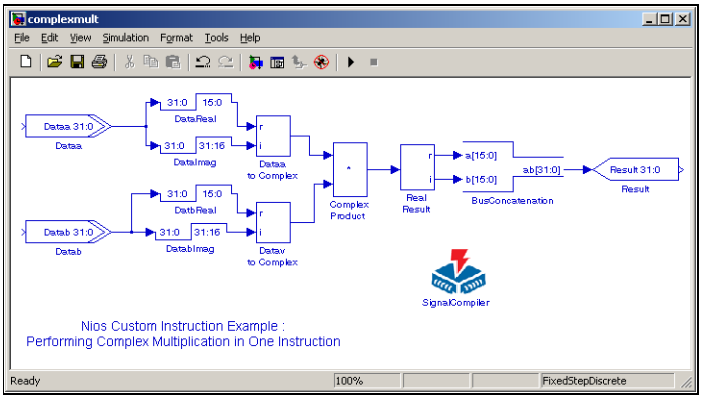
- *Custom logic block*—Hardware that performs the operation. The Nios II embedded processor can include up to five user-defined custom logic blocks. The blocks become part of the Nios II microprocessor's ALU.

- *Software macro*—Allows the system designer to access the custom logic through software code.

For more information on Nios II embedded processor custom instructions, see *AN 188: Custom Instructions for the Nios Embedded Processor*.

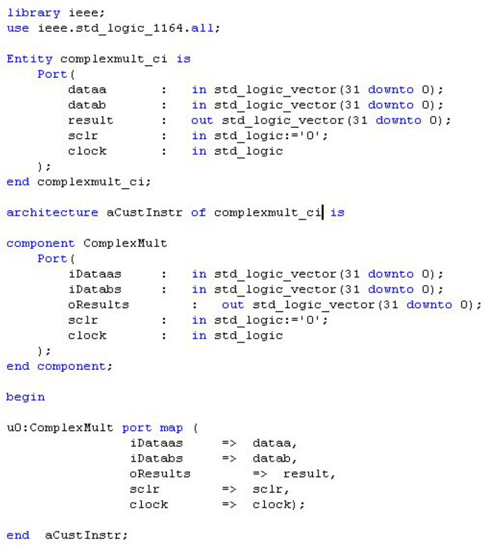
Figure 8–6 shows an example model using blocks from the Avalon Ports library.

Figure 8–6. Custom Instruction Block Example



When you generate VHDL of your model that uses Custom Instruction blocks, SignalCompiler creates a wrapper file for use with SOPC Builder. See Figure 8–7 for an example wrapper file.

Figure 8–7. Example Wrapper File for Use with SOPC Builder

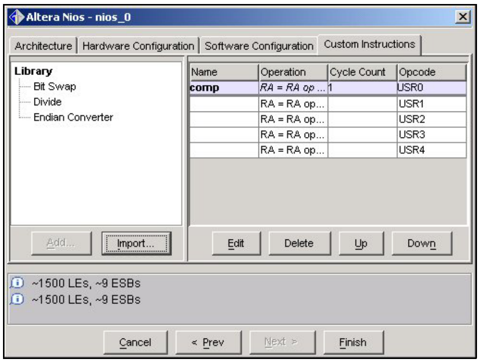


```
library ieee;
use ieee.std_logic_1164.all;

Entity complexmult_ci is
    Port(
        dataa       :   in std_logic_vector(31 downto 0);
        datab       :   in std_logic_vector(31 downto 0);
        result      :   out std_logic_vector(31 downto 0);
        sclr        :   in std_logic:='0';
        clock       :   in std_logic
    );
end complexmult_ci;

architecture aCustInstr of complexmult_ci is

component ComplexMult
    Port(
        iDataas     :   in std_logic_vector(31 downto 0);
        iDatabs     :   in std_logic_vector(31 downto 0);
        oResults        :   out std_logic_vector(31 downto 0);
        sclr        :   in std_logic:='0';
        clock       :   in std_logic
    );
end component;

begin

u0:ComplexMult port map (
                iDataas     =>  dataa,
                iDatabs     =>  datab,
                oResults        =>  result,
                sclr        =>  sclr,
                clock       =>  clock);

end  aCustInstr;
```

You can specify that the Nios II embedded processor use this wrapper file as a custom instruction. See *AN 188: Custom Instructions for the Nios Embedded Processor* for instructions on how to specify the file as a custom instruction. Figure 8–8 shows the Nios II CPU **Custom Instruction** tab and the **Design Import Wizard**.

☞ When you import the custom instruction in SOPC Builder, you must specify the number of clock cycles that your design uses.

Figure 8–8. Importing a Nios II Custom Instruction



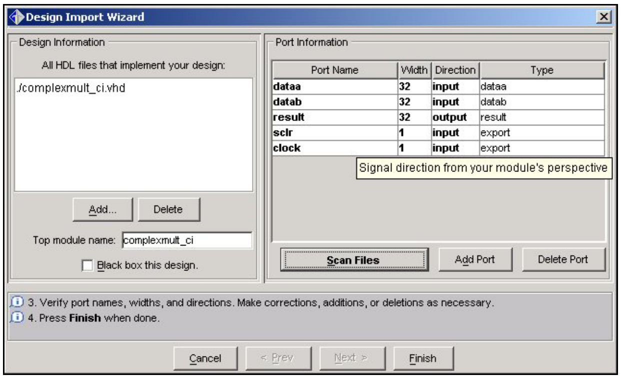
Altera Nios - nios_0

Architecture | Hardware Configuration | Software Configuration | Custom Instructions

Library
- Bit Swap
- Divide
- Endian Converter

Name	Operation	Cycle Count	Opcode
comp	RA = RA op ...	1	USR0
	RA = RA op...		USR1
	RA = RA op...		USR2
	RA = RA op...		USR3
	RA = RA op...		USR4

~1500 LEs, ~9 ESBs
~1500 LEs, ~9 ESBs



Design Import Wizard

Design Information

All HDL files that implement your design:

./complexmult_ci.vhd

Top module name: complexmult_ci

Black box this design.

Port Information

Port Name	Width	Direction	Type
dataa	32	input	dataa
datab	32	input	datab
result	32	output	result
sclr	1	input	export
clock	1	input	export

Signal direction from your module's perspective

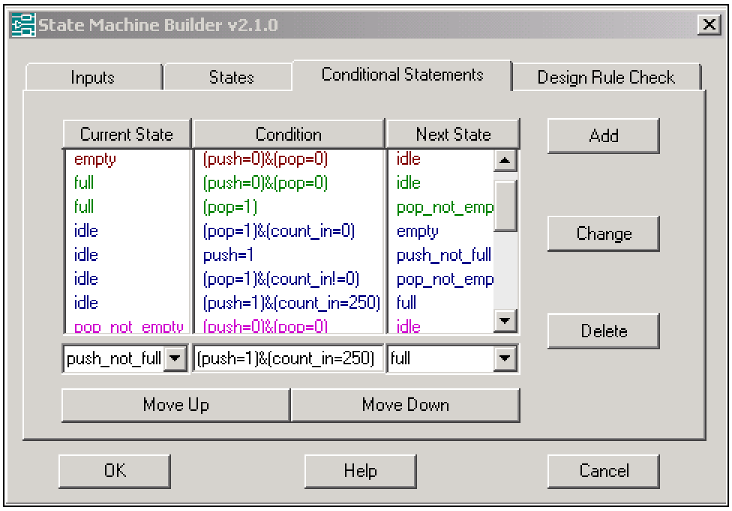
3. Verify port names, widths, and directions. Make corrections, additions, or deletions as necessary.
4. Press **Finish** when done.

Chapter 9. State Machine Functions Library

State Machine Table Block

The State Machine Table block allows you to use a state transition table to build a one-hot Moore state machine where the output is equal to the current state (see Figure 9–1).

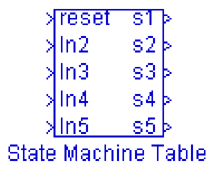
Figure 9–1. State Transition Table Block for a FIFO Controller



Current State	Condition	Next State
empty	(push=0)&(pop=0)	idle
full	(push=0)&(pop=0)	idle
full	(pop=1)	pop_not_emp
idle	(pop=1)&(count_in=0)	empty
idle	push=1	push_not_full
idle	(pop=1)&(count_in!=0)	pop_not_emp
idle	(push=1)&(count_in=250)	full
pop_not_empty	(push=0)&(pop=0)	idle
push_not_full	(push=1)&(count_in=250)	full

The default State Machine Table symbol is shown in Figure 9–2. The default state machine has five inputs and five states. Each state is represented by an output. You must connect each output to a DSP Builder Output block (from the IO/Bus library). While the state machine is operating, an output is assigned a logic level 1 if its respective state is equal to the current state. All other outputs get assigned a logic level 0. In Simulink, the inputs and outputs are represented as signed integers. In VHDL, the input and output are represented as standard logic vectors.

Figure 9–2. Default State Machine Table Block



Chapter 10. Storage Library

Delay Block

The Delay block delays the incoming data by the amount specified by the `Depth` parameter. The block accepts any data type as inputs. Table 10–1 shows the block parameters.

Table 10–1. Delay Block Parameters

Name	Value	Description
Depth	User Defined	Indicate the delay length of the block.
Use Control Inputs	On or Off	Indicate that you wish to use the additional control inputs, clock enable, and resets.
Clock Phase Selection	User Defined	Phase selection. Indicate the phase selection with a binary string, where a 1 indicates the phase in which the block is enabled. For example: `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 second phase out of 4 and only the second data out of 4 (sampled at the rate 1) passes through. In other words, the data on phases 1, 3, and 4 do not pass through the delay block.

Table 10–2 shows the block I/O formats.

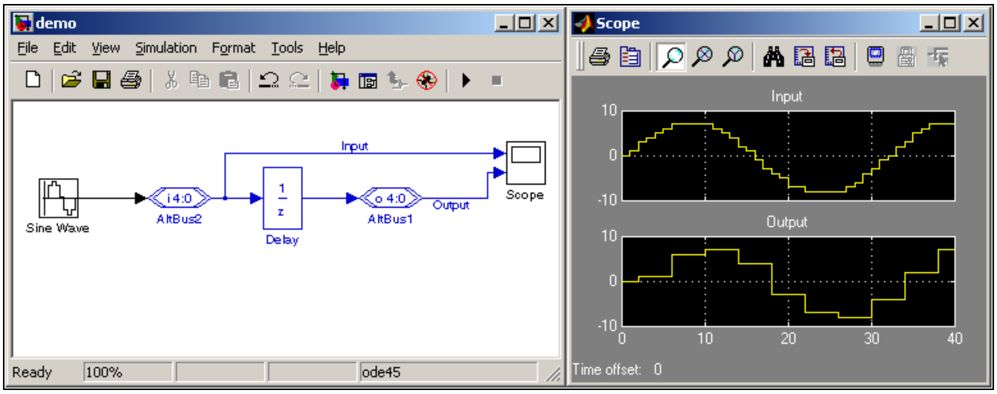
Table 10–2. Delay Block I/O Formats Note (1)

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 10–2:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 10–1 shows an example using the Delay block.

Figure 10–1. Delay Block Example



Down Sampling Block

The Down Sampling block decreases the output sample rate from the input sample rate. The output data is sampled at every *m*th cycle where *m* is equal to the down sampling rate. The input sample rate is normalized to one in Simulink (see "Frequency Design Rule" on page 1–5 for a description of how to normalize the frequency). Table 10–3 shows the block parameters.

Table 10–3. Down Sampling Block Parameters

Name	Value	Description
Down Sampling Rate	1 - 20	Indicate the down sampling rate.

Table 10–4 shows the block I/O formats.

Table 10–4. Down Sampling Block I/O Formats Note (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 10–4:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

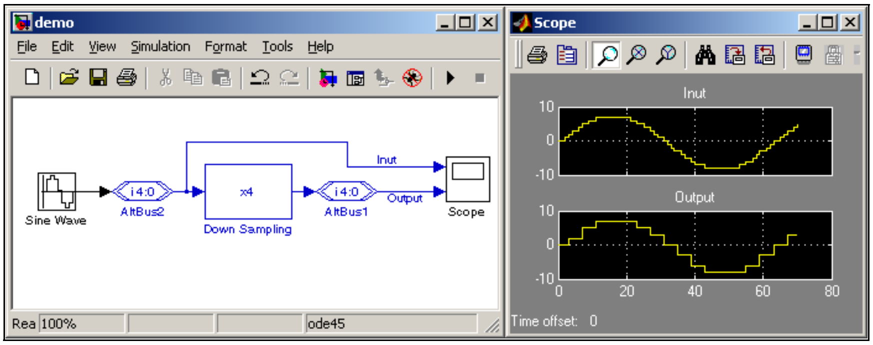
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 10–2 shows an example using the Down Sampling block.

Figure 10–2. Down Sampling Block Example



Dual-Port RAM Block

When you use the Dual-Port RAM block, SignalCompiler maps data to the embedded RAM (EABs or ESBs) in Altera® devices; the contents of the RAM are pre-initialized to zero. The Dual-Port RAM block accepts any data type as input. All input and output ports are registered. Table 10–5 shows the block parameters.

Table 10–5. Dual-Port RAM Block Parameters

Name	Value	Description
Address Width	2 - 20	Indicate the address width.
Clock Phase Selection	User Defined	Phase selection. 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 out of 4 and only the second data out of 4 (sampled at the rate 1) passes through. In other words, the data on phases 1, 3, and 4 do not pass through the delay block.
Ram Block Type	AUTO, M512, M4K, M-RAM	Specifies the RAM block type.

The Dual-Port RAM block has the following ports:

- `d`—Input data
- `q`—Output data
- `rdad`—Read address bus
- `wrad`—Write address bus
- `wren`—Write enable

Table 10–6 shows the block I/O formats.

Table 10–6. Dual-Port RAM Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[L1].[R1]}$ $I2_{[L2].[0]}$ $I3_{[L2].[0]}$ $I4_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC_VECTOR({L2 - 1} DOWNTO 0) I3: in STD_LOGIC_VECTOR({L3 - 1} DOWNTO 0) I4: in STD_LOGIC	Explicit
O	$O1_{[L1].[R1]}$	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0)	Explicit

Notes to Table 10–6:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

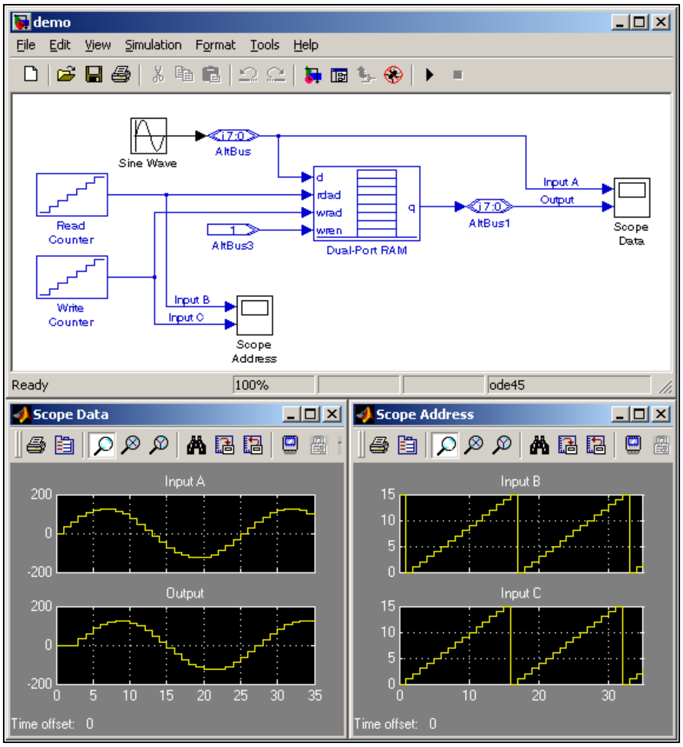
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 10–3 shows an example using the Dual-Port RAM block.

Figure 10–3. Dual-Port RAM Block Example



FIFO Block

The FIFO block implements a parameterized, single-clock FIFO buffer. The block contains the inputs and outputs shown in Table 10–7.

Table 10–7. FIFO Block Inputs & Outputs

Signal	Direction	Description
`d`	Input	Data input to the FIFO buffer.
`wrreq`	Input	Write request control. The `data[]` port is written to the FIFO buffer.
`rdreq`	Input	Read request control. The oldest data in the FIFO buffer goes to the `q[]` port.
`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 `rdreq` port.
`usedw`	Output	Indicates the number of words that are currently in the FIFO buffer.

Table 10–8 shows the block parameters.

Table 10–8. FIFO Block Parameters

Name	Value	Description
Number of Words in the FIFO Buffer	User Defined	Indicate how many words you would like in the FIFO buffer. The default is 64.
Input Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Choose the bus type format.
[number of bits].[]	1 - 51	Indicate the number of bits stored on the left side of the binary point including the sign bit.
[].[number of bits]	0-51	Indicate the number of bits stored on the right side of the binary point. This option only applies to signed fractional formats.

Table 10–9 shows the block I/O formats.

Table 10–9. FIFO Block I/O Formats Note (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 10–9:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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.

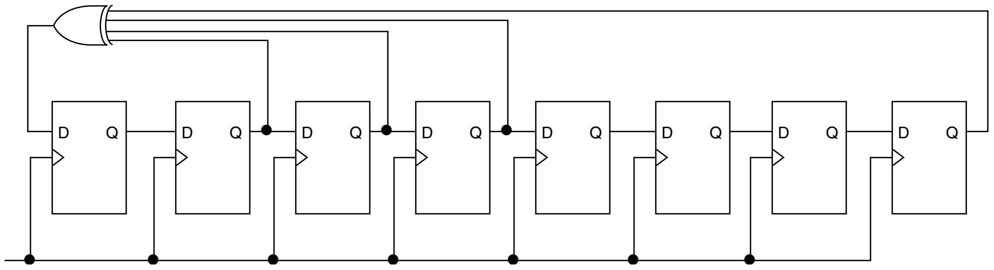
(3) $I1_{[L].[R]}$ is an input port. $O1_{[L].[R]}$ is an output port.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

LFSR Sequence Block

The LFSR Sequence block implements a linear feedback shift register, which shifts one bit across *L* registers. The register output bits are shifted from LSB to MSB. According to the LFSR polynomial, the register output bits are XOR'ed together. For example, when choosing an LFSR sequence of length eight, the default polynomial is $x^8 + x^4 + x^3 + x^2 + 1$ with the circuitry shown in Figure 10–4.

Figure 10–4. Default LFSR Sequence Block with Length 8 Circuitry

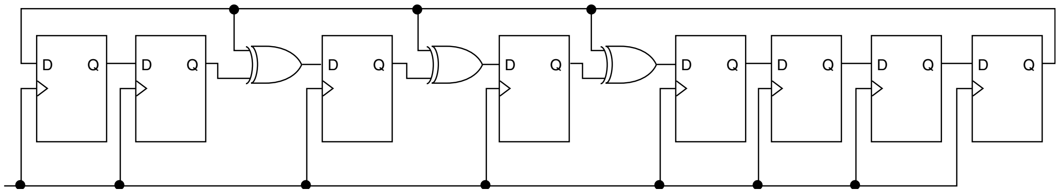


In this default structure:

- 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

To modify the LFSR sequence implemented by the block, you change the block's default parameters values. For instance, changing the feedback structure to an internal 2-input gate or one to many, DSP Builder implements the circuitry shown in Figure 10–5.

Figure 10–5. Internal 2-Input Gate Circuitry



This circuitry changes the sequence from

1 1 1 1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0 1

to

1 1 1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0 0

Table 10–10 shows the block parameters.

Table 10–10. LFSR Sequence Block Parameters

Name	Value	Description
LFSR Length	User Defined	Indicate the LFSR length as an integer.
Sampling Period	User Defined	Indicate the block sampling period.
Use Control Input	On or Off	When you turn on this option, DSP Builder implements the clock enable and reset input signals.
Default LSFR Setting	On or Off	When on, this option implements the default structure shown in Figure 10–4. When on, this option allows you to customize the setting by enabling the **Feedback Structure**, **Feedback Gate Type**, **Initial Register Values (Hex)**, and **Primitive Polynomial Tap Sequence** options.
Use Parallel Output	On or Off	When you turn on this option, DSP Builder implements an additional output signal, pout (parallel output).
Feedback Structure	Internal 2-Input gate (One-to-Many) or External n-Input Gate (Many-to-One)	Choose whether you want an internal or external structure.
Feedback Gate Type	XOR or XNOR	Indicate the type of feedback gate to implement.
Initial Register Values (Hex)	Any Hexadecimal Number	Indication the initial values in the register.
Primitive Polynomial Tap Sequence	A User-Defined Array of Polynomial Coefficients	Indicate the sequence. The numbers should be in brackets ([]).

Table 10–11 shows the block I/O formats.

Table 10–11. LFSR Sequence Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type
I	I1[1].[0]	ena: in STD_LOGIC	-
	I2[1].[0]	rst: in STD_LOGIC	-
O	O[1].[0]	sout: out STD_LOGIC	-
	O2[L].[0] – L LFSR Length	pout: out STD_LOGIC_VECTOR(L-1 DOWNTO 0)	-

Notes to Table 10–11:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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.

Memory Delay Block

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 10–12 shows the block parameters.

Table 10–12. Memory Delay Block Parameters

Name	Value	Description
Use Clock Enable	On or Off	Indicate that you wish to use the clock enable input.
Delay Value	User Defined	Indicate how much of a delay to implement.
RAM Block Type	AUTO, M512, M4K, M-RAM	Specifies the RAM block type.

Table 10–13 shows the block I/O formats.

Table 10–13. Memory Delay Block I/O Formats Note (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 10–13:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Multi Rate FIFO Block

The Multi Rate FIFO block implements a parameterized, dual-clock FIFO buffer. The block contains these inputs and outputs:

Inputs:

- `d`: Data input to the FIFO buffer.
- `wrreq`: Write request control. The `data[]` port is written to the FIFO buffer.
- `rdreq`: Read request control. The oldest data in the FIFO buffer goes to the `q[]` port.

Outputs:

- `q`: Data output from the FIFO buffer.
- `wrfull`: Indicates that the FIFO buffer is full and disables the `wrreq` port.
- `wrempty`: Indicates that the FIFO buffer is empty and disables the `rdreq` port.
- `wrusedw`: Indicates the number of words that are currently in the FIFO buffer with respect to the write side sampling rate.
- `rdfull`: Indicates that the FIFO buffer is full and disables the `wrreq` port.
- `rdempty`: Indicates that the FIFO buffer is empty and disables the `rdreq` port.
- `rdusedw`: Indicates the number of words that are currently in the FIFO buffer with respect to the read side sampling rate.

Table 10–14 shows the block I/O formats

Table 10–14. Multi Rate FIFO Block Parameters (Part 1 of 2)

Name	Value	Description
Number of Words in the FIFO	Integer	Set the FIFO depth
Input Bus Type	Signed Integer, Signed Fractional, or Unsigned Integer	Choose the bus type format.
Input Bus [number of bits].[]	1 - 51	Indicate the number of bits stored on the left side of the binary point.
Input Bus [].[number of bits]	0 - 51	Indicate the number of bits stored on the right side of the binary point. This option only applies to signed fractional formats.
Write Side Sampling Period (second)	Double	Specify the sample time period in seconds of the write side of the FIFO.

Table 10–14. Multi Rate FIFO Block Parameters (Part 2 of 2)

Name	Value	Description
Read Side Sampling Period (second)	Double	Specify the sample time period in seconds of the read side of the FIFO.
Use Outputs usedw[]	On or off	Indicate whether you would like to use additional control outputs: wrusedw and rdusedw.
Ram Block Type	AUTO, M512, M4K, or M-RAM	Choose the FPGA RAM type used for the Multi-rate FIFO.

Table 10–15 shows the block I/O formats.

Table 10–15. Multi Rate FIFO Block I/O Formats Note (1)

I/O	Simulink (2), (3)	VHDL	Type (4)
I	$I1_{[1].[R]}$ $I2_{[1]}$ $I3_{[1]}$	I1: in STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) I2: in STD_LOGIC I3: in STD_LOGIC	Explicit Explicit Explicit
O	$O1_{[1].[R]}$ $O2_{[1]}$ $O3_{[1]}$ $O4_{[1]}$ $O5_{[1]}$ $O6_{[12].[0]}$ $O7_{[12].[0]}$	O1: out STD_LOGIC_VECTOR({L1 + R1 - 1} DOWNTO 0) O2: out STD_LOGIC O3: out STD_LOGIC O4: out STD_LOGIC O5: out STD_LOGIC O6: out STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0) O7: out STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0)	Explicit Explicit Explicit Explicit Explicit Explicit-optional Explicit-optional

Notes to Table 10–15:

(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Parallel to Serial Block

The Parallel to Serial block implements a parallel (input `d`) to serial bus conversion (output `sd`). Table 10–16 shows the block parameters.

Table 10–16. Parallel to Serial Block Parameters

Name	Value	Description
Data Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Choose the bus type format.
[number of bits].[]	1 - 51	Indicate the number of bits stored on the left side of the binary point.
[].[number of bits]	0-51	Indicate the number of bits stored on the right side of the binary point. This option only applies to signed fractional formats.
Serial Bit Order	MSB First, LSB First	Indicate whether the most significant bit (MSB) or least significant bit (LSB) should be transmitted first.

Table 10–17 shows the block I/O formats.

Table 10–17. Parallel to Serial Block I/O Formats Note (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	Explicit
O	$O1_{[1]}$	O1: out STD_LOGIC	Explicit

Notes to Table 10–17:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

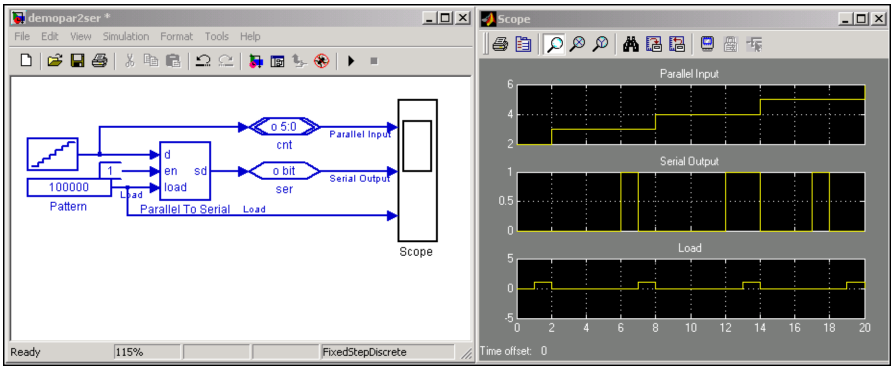
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 10–6 shows an example using the Parallel to Serial block.

Figure 10–6. Parallel to Serial Block Example



Pattern Block

The Pattern block generates a repeating periodic bit sequence in time. For example, `01100` yields `011000110001100011000110001100011000 1100` in time. You can change the output data rate for a registered block by feeding the clock enable input with the output of the Pattern block. Table 10–18 shows the block parameters.

Table 10–18. Pattern Block Parameters

Name	Value	Description
Binary Sequence	User Defined	Indicate the sequence that you wish to use.
Use Control Inputs	On or Off	Indicate that you wish to use additional control inputs (clock enable and reset).

Table 10–19 shows the block I/O formats.

Table 10–19. Pattern Block I/O Formats Note (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 10–19:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

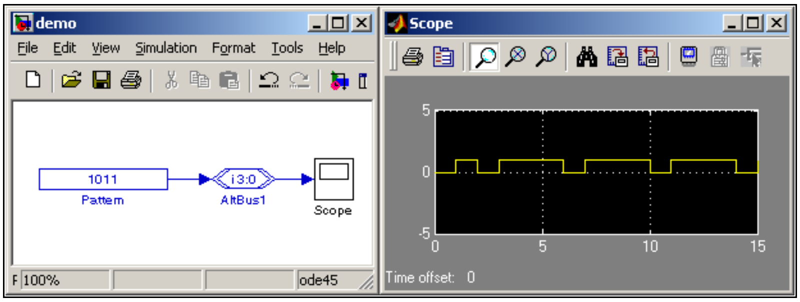
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 10–7 shows an example using the Pattern block.

Figure 10–7. Pattern Block Example



ROM EAB Block

The ROM EAB block is used to read out pre-loaded data. The data input must be specified as a hexadecimal file. To use the embedded memory in an Altera device as ROM, use the ROM EAB block to read in an Intel-format Hexadecimal File (**.hex**) containing the ROM data.

You can use the Quartus® II software to generate a Hex File. Search for "Creating a Memory Initialization File or Hexadecimal (Intel-Format) File" in Quartus II Help for instructions on creating this file.

☞ If you use the Quartus II software to generate a Hex File, you must save the Hex File in your DSP Builder working directory.

Table 10–20 shows the block parameters.

Table 10–20. ROM EAB Block Parameters

Name	Value	Description
Data Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Choose the bus type format.
[number of bits].[]	1 - 51	Indicate the number of bits stored on the left side of the binary point including the sign bit.
[].[number of bits]	1-51	Indicate the number of bits stored on the right side of the binary point. This option only applies to signed fractional formats.
Address Width	2 - 20	Indicate the address width.
Clock Phase Selection	User Defined	Phase selection. 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 out of 4 and only the second data out of 4 (sampled at the rate 1) passes through. In other words, the data on phases 1, 3, and 4 do not pass through the delay block.
Input Hex File	User Defined; <*filename*>**.hex**	Indicate the name of the HEX File to use.

Table 10–21 shows the block I/O formats.

Table 10–21. ROM EAB Block I/O Formats *Note (1)*

I/O	Simulink *(2), (3)*	VHDL	Type *(4)*
I	I1$_{[L1].[0]}$	I1: in STD_LOGIC_VECTOR({L1 - 1} DOWNTO 0)	Explicit
O	O1$_{[LPO].[RPO]}$	O1: out STD_LOGIC_VECTOR({LPO + LPO - 1} DOWNTO 0)	Explicit

Notes to Table 10–21:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

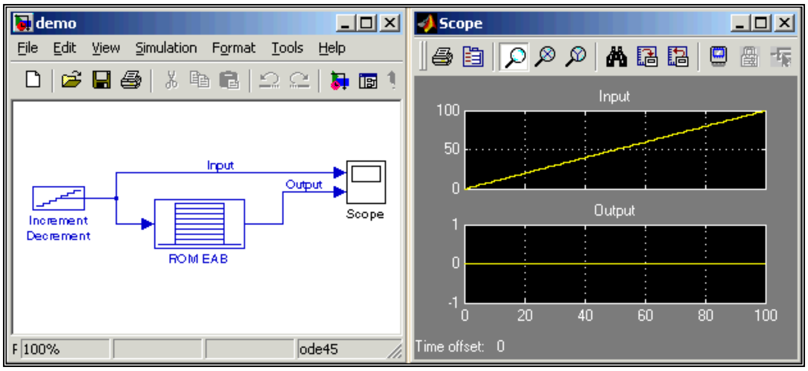
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 10–8 shows an example using the ROM EAB block.

Figure 10–8. ROM EAB Block Example



Serial to Parallel Block

The Serial to Parallel block implements a serial (input `sd`) to parallel bus conversion (output `d`). Table 10–22 shows the block parameters.

Table 10–22. Serial to Parallel Block Parameters

Name	Value	Description
Data Bus Type	Signed Integer, Signed Fractional, Unsigned Integer	Choose the bus type format.
[number of bits].[]	1 - 51	Indicate the number of bits stored on the left side of the binary point including the sign bit.
[].[number of bits]	0-51	Indicate the number of bits stored on the right side of the binary point. This option only applies to signed fractional formats.
Serial Bit Order	MSB First, LSB First	Indicate whether the most significant bit (MSB) or least significant bit (LSB) should be transmitted first.

Table 10–23 shows the block I/O formats.

Table 10–23. Serial to Parallel Block I/O Formats Note (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 10–23:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

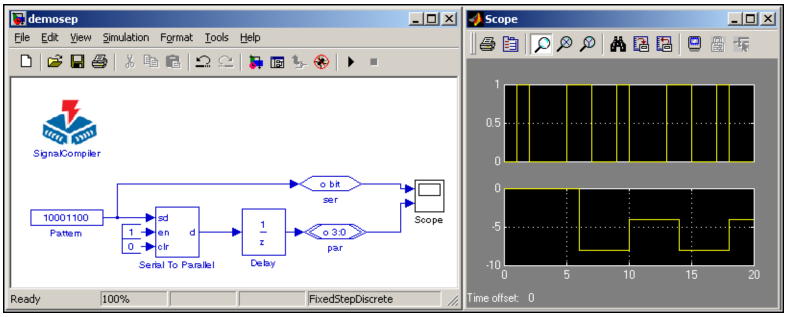
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 10–9 shows an example using the Serial to Parallel block.

Figure 10–9. Serial to Parallel Block Example



Shift Taps Block

The Shift Taps block implements a shift register that you can use for filters or convolution. In the Stratix® II, Stratix, 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 is implemented in the small memory. Table 10–24 shows the block parameters.

Table 10–24. Shift Taps Block Parameters

Name	Value	Description
Number of Taps	User Defined	Specifies the number of regularly spaced taps along the shift register
Distance Between Taps	User Defined	Specifies the distance between the regularly spaced taps in clock cycles. This number translates to the number of RAM words that will be used.
Use Shift Out	On or Off	Indicate whether you want to create an output from the end of the shift register for cascading.
Use Clock Enable	On or Off	Indicate whether you would like to use an additional control input (clock enable).
Use Dedicated Hardware	On or Off	If you are targeting Stratix II, Stratix, or Stratix GX devices, turn on this option to implement the functionality in Stratix small memory.

Table 10–25 shows the block I/O formats.

Table 10–25. Shift Taps Block I/O Formats Note (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	$O1_{[L1].[R1]}$ .... $Oi_{[L1].[R1]}$ ... $On_{[L1].[R1]}$ (n = number of taps)	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) (n = number of taps)	Implicit

Notes to Table 10–25:

(1) For signed integers and signed binary fractional numbers, the MSB bit is the sign bit.

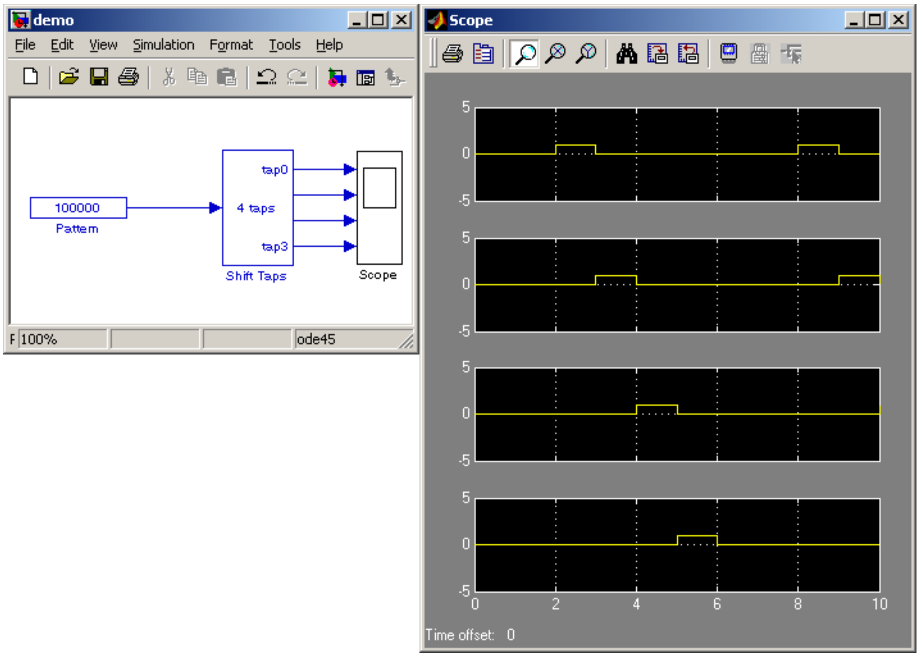
(2) [L] is the number of bits on the left side of the binary point; [R] is the number of bits on the right side of the binary point. For signed or unsigned integers R = 0, that is, [L].[0]. For single bits, R = 0, that is, [1] is a single bit.

(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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 10–10 shows an example using the Shift Taps block.

Figure 10–10. Shift Taps Block Example



Up Sampling Block

The Up Sampling block increases the output sample rate from the input sample rate. The output data is sampled every *l* cycles where *l* is equal to the up sampling rate. Table 10–26 shows the block parameters.

Table 10–26. Up Sampling Block Parameters

Name	Value	Description
Up Sampling Rate	An integer greater than 0	Indicate the up sampling rate.

Table 10–27 shows the block I/O formats.

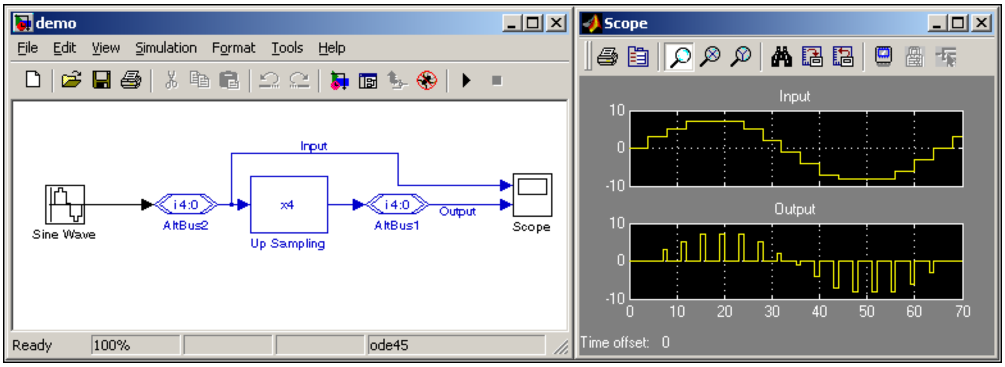
Table 10–27. Up Sampling Block I/O Formats Note (1)

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 10–27:
(1) For signed integers and signed binary fractional numbers, the MSB bit 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 data path bit with propagation mechanism. If you want to specify the bus format of an implicit input port, use a BusConversion block to set the width.

Figure 10–11 shows an example using the Up Sampling block.

Figure 10–11. Up Sampling Block Example



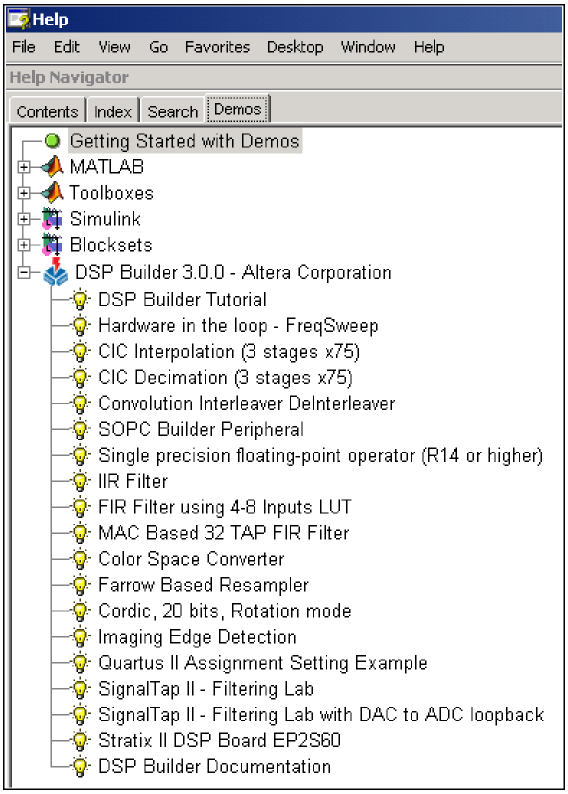
Chapter 11. Example Designs

Introduction

Altera® DSP Builder provides a variety of example designs, which you can use to learn from or as a starting point for your own design. This section describes the available designs.

To view the example designs, type `demo` at the MATLAB command prompt. The Help window opens up as shown in Figure 11–1. The demos are listed under DSP Builder 3.0.0 -Altera Corporation.

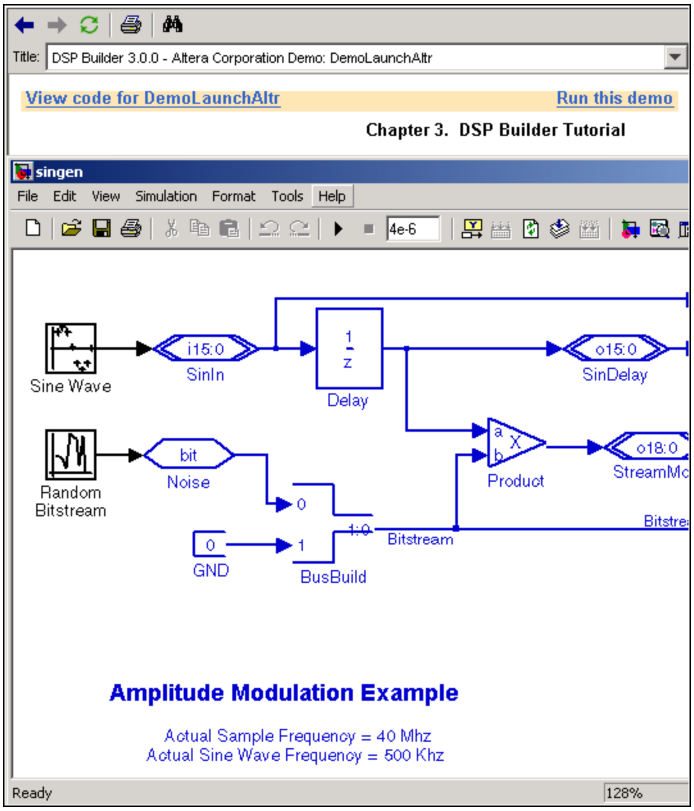
Figure 11–1. DSP Builder Design Example Demos



Getting Started with Demos
MATLAB
Toolboxes
Simulink
Blocksets
DSP Builder 3.0.0 - Altera Corporation
DSP Builder Tutorial
Hardware in the loop - FreqSweep
CIC Interpolation (3 stages x75)
CIC Decimation (3 stages x75)
Convolution Interleaver DeInterleaver
SOPC Builder Peripheral
Single precision floating-point operator (R14 or higher)
IIR Filter
FIR Filter using 4-8 Inputs LUT
MAC Based 32 TAP FIR Filter
Color Space Converter
Farrow Based Resampler
Cordic, 20 bits, Rotation mode
Imaging Edge Detection
Quartus II Assignment Setting Example
SignalTap II - Filtering Lab
SignalTap II - Filtering Lab with DAC to ADC loopback
Stratix II DSP Board EP2S60
DSP Builder Documentation

For example, if you select the DSP Builder Tutorial, the example model design window opens, and you may view the demo code or run the demo as shown in Figure 11–2.

Figure 11–2. Example of the DSP Builder Tutorial


Title: DSP Builder 3.0.0 - Altera Corporation Demo: DemoLaunchAltr
View code for DemoLaunchAltr
Run this demo
Chapter 3. DSP Builder Tutorial
singen
File Edit View Simulation Format Tools Help
4e-6
Sine Wave
i15:0
SinIn
1/z
Delay
o15:0
SinDelay
Random Bitstream
bit
Noise
a
b
X
Product
o18:0
StreamMc
0
1
1:0
Bitstream
Bitstre
0
GND
BusBuild
Amplitude Modulation Example
Actual Sample Frequency = 40 Mhz
Actual Sine Wave Frequency = 500 Khz
Ready
128%

DSP Builder Tutorial

This tutorial uses an example amplitude modulation design, **SinGen.mdl**, to demonstrate the DSP Builder design flow. The amplitude modulator design example is a modulator that has a sine wave generator, a quadrature multiplier, and a delay element. The example file is named **singen.mdl**.

For detailed information on this example, see the *DSP Builder Tutorial* chapter in the *DSP Builder User Guide*.

Frequency Sweep

This HIL tutorial uses an example of the implementation of a low-pass filter applied on the output of a modulated sine wave generation created using the Cordic algorithm.

CIC Interpolation (3 Stages x75)

CIC (cascaded integrator 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 example file is named **CiCInterpolator75.mdl**.

CIC Decimation (3 Stages x75)

CIC filters are 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. This design is typically used 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 example file is named **Cicdecimator75.mdl**.

Convolution Interleaver Deinterleaver

Convolutional interleaver deinterleavers are typically used 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 example file is named **top12x17.mdl**.

Single MAC 32 Tap FIR Filter

This example illustrates how to implement a 32 tap FIR filter using a single Multiply Accumulate Unit and a single memory element for the TAP delay line. This design requires The MathWorks Signal Processing ToolBox to calculate the coefficient using 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');
```

32 Tap Serial FIR Filter

This example illustrates how to implement a 32 tap FIR filter using a single a serial distributed arithmetic partition. This design requires The Mathworks Signal Processing ToolBox to calculate the coefficient using 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;
```

SOPC Builder Peripheral

The example consists of a 4-tap FIR filter with variable coefficients. The coefficients are loaded using the Nios embedded processor while the input data is supplied by an off-chip source through an A/D converter. The filtered output data is sent off-chip through a D/A converter. The **toparc.mdl** design is included as a peripheral to the Avalon bus. The example file is named **topach.mdl**.

For detailed information on this example, see the *Using the SOPC Builder Links Library* chapter in the *DSP Builder User Guide*.

IIR Filter

This example illustrates how to implement an order 2 IIR filter using a Direct Form two structure. In the design, the coefficients are computed using the MATLAB function butter, which is a Butterwork 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 using the Gain block. The example file is named **topiir.mdl**.

Farrow Based Decimating Sample Rate Converter

This example illustrates how to implement a Farrow Based Decimating Sample Rate Converter. Many integrated systems, such as Software Defined Radios (SDR), require data to be resampled so that a unit can comply with different communication standards, where the sample rates are different. In some cases, where one clock rate is a simple integer multiple of another clock rate, resampling can be accomplished using interpolating and decimating FIR filters. However, in most cases the interpolation and decimation factors are so high that this approach is impractical.

Farrow resamplers offer 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 is then used to combine the phases of a polyphase filter in such a way that a sample for the wanted output phase is generated. This design demonstrates a Farrow resampler. You can simulate its performance in MATLAB, change it as required for your application, generate a VDHL version and synthesize it to Altera devices. This demo is designed for 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 get a cost effective solution. The example file is named **FarrowResamp.mdl**.

Appendix A. Example Tcl Scripts

This appendix contains sample Tcl scripts created by the SignalCompiler block for the amplitude modulation design example. See the *DSP Builder Tutorial* chapter in the *DSP Builder User Guide* for information on how these files were generated.

The following is an example of a Tcl script for synthesis using Quartus® II synthesis:

```
# TCL Script for the Quartus II software

# Packages
package require ::quartus::project

# Directory Variables
set workdir "C:/DSPBuilder/designexamples/Tutorials/GettingStartedSinMdl"
set libdir "c:/DSPBuilder/Altlib"
set megadir "c:/DSPBuilder/MegaCoreLib"


# Change to working directory
cd $workdir

# Create Quartus II project
project_new singen -overwrite
set_global_assignment -name "VHDL_FILE" "$libdir/DSPBUILDERPACK.VHD"
set_global_assignment -name "VHDL_FILE" "$libdir/DSPBUILDER.VHD"
set_global_assignment -name "VHDL_FILE" "$workdir/singen.vhd";
set_global_assignment -name "SIMULATOR_SETTINGS" "singen"
set_global_assignment -name "COMPILER_SETTINGS" "singen"
set_global_assignment -name "USER_LIBRARIES" "$megadir"

# Set Compiler assignements
set_global_assignment -name "FAMILY" "stratix";
set_global_assignment -name "APEX20K_OPTIMIZATION_TECHNIQUE" "SPEED"
set_global_assignment -name "MERCURY_OPTIMIZATION_TECHNIQUE" "SPEED"
set_global_assignment -name "FLEX10K_OPTIMIZATION_TECHNIQUE" "SPEED"
set_global_assignment -name "FLEX6K_OPTIMIZATION_TECHNIQUE" "SPEED"
set_global_assignment -name "MAX7000_OPTIMIZATION_TECHNIQUE" "SPEED"
set_global_assignment -name "STRATIX_OPTIMIZATION_TECHNIQUE" "SPEED"
set_global_assignment -name "STRATIXII_OPTIMIZATION_TECHNIQUE" "SPEED"
set_global_assignment -name "DEVICE" "AUTO";
set_global_assignment -name "LOGICLOCK_INCREMENTAL_COMPILE_ASSIGNMENT" "ON";
set_global_assignment -name "LOGICLOCK_INCREMENTAL_COMPILE_FILE"
"atom_netlists/singen.vqm";


# Simulator Assignments for test
set_project_settings  -sim "singen"
```

```
set_global_assignment -name "USE_COMPILER_SETTINGS" "singen"
set_global_assignment -name "VECTOR_INPUT_SOURCE" "singen.vec"

project_close
```

The following is an example of a Tcl script for synthesis using the LeonardoSpectrum™ synthesis tool:

```
# TCL Script for LeonardoSpectrum

# Set synthesis parameters
set ec_tech stratix
load_library $ec_tech
set target $ec_tech
set chip TRUE
set macro FALSE
set_working_dir "C:/DSPBuilder/designexamples/Tutorials/GettingStartedSinMdl"

# Load VHDL design files
set DSPbuilderFileList [list "c:/DSPBuilder/Altlib/220pack.vhd"]
lappend DSPbuilderFileList "c:/DSPBuilder/Altlib/altera_mf_components.vhd"
lappend DSPbuilderFileList "c:/DSPBuilder/Altlib/dspbuilderpack.vhd"
lappend DSPbuilderFileList "c:/DSPBuilder/Altlib/dspbuilder.vhd"
lappend DSPbuilderFileList
"C:/DSPBuilder/designexamples/Tutorials/GettingStartedSinMdl/singen.vhd"
read $DSPbuilderFileList

# Synthesis optimization
optimize -ta $ec_tech -delay -effort standard -chip -hierarchy auto
set_attribute -port {clock} -name clock_cycle -value 40
optimize_timing -force

# Write-out edif netlist for Quartus II Compilation
auto_write "C:/DSPBuilder/designexamples/Tutorials/GettingStartedSinMdl/singen.edf"
```

The following is an example of a Tcl script for synthesis using the Synplicity synthesis tool:

```
TCL Script for Synplify

# Change to working directory
cd "C:/DSPBuilder/designexamples/Tutorials/GettingStartedSinMdl"

# Load VHDL libraries
add_file -vhdl -lib dspbuilder "c:/DSPBuilder/Altlib/dspbuilderpack.vhd"
add_file -vhdl -lib dspbuilder "c:/DSPBuilder/Altlib/dspbuilder.vhd"

# Load VHDL design files
add_file -vhdl -lib work
"C:/DSPBuilder/designexamples/Tutorials/GettingStartedSinMdl/singen.vhd"

# Set synthesis options
set_option -technology stratix
set_option -write_apr_constraint 1
set_option -resource_sharing 0
```

```
project -result_file
"C:/DSPBuilder/designexamples/Tutorials/GettingStartedSinMdl/singen.vqm"

# Run synthesis
project -run
project -close
```

The following is an example Tcl script for the ModelSim® simulator:

```
# TCL Script for ModelSim

# Set Simulation timing parameters
set SimTime 4000
set TimeResolution 1ps

# Directory Variables
set workdir "C:/DSPBuilder/designexamples/Tutorials/GettingStartedSinMdl"
set libdir "c:/DSPBuilder/Altlib"
set megadir "c:/DSPBuilder/MegaCoreSimLib"
set bForceRecompile 1


# Close existing ModelSim simulation
quit -sim

# Create ModelSim project
if {[file exist [project env]] > 0} {project close}
cd $workdir
if {[file exist "$workdir/singenDspBuilder.mpf"] == 0} {
    project new $workdir singenDspBuilder work
} else {
    project open singenDspBuilder
}

# Compile LPM VHDL library
if {([file exist lpm] ==0)||($bForceRecompile>0)} {
                exec vlib lpm
                vcom -93 -explicit -work lpm  "$libdir/220pack.vhd"
                vcom -93 -explicit -work lpm  "$libdir/220model.vhd"
                }
exec vmap lpm lpm

# Compile Megafunction VHDL library
if {([file exist altera_mf] ==0)||($bForceRecompile>0)} {
                exec vlib altera_mf
                exec vmap altera_mf altera_mf
                vcom -93 -explicit -work altera_mf  "$libdir/altera_mf_components.vhd"
                vcom -93 -explicit -work altera_mf  "$libdir/altera_mf.vhd"
                }
exec vmap altera_mf altera_mf
# Compile dspbuilder VHDL library
if {([file exist dspbuilder] ==0)||($bForceRecompile>0)}    {
                exec vlib dspbuilder
             vcom -93 -explicit -work dspbuilder  "$megadir/SignalTapNode.vhd"
             vcom -93 -explicit -work dspbuilder  "$libdir/dspbuilderpack.vhd"
             vcom -93 -explicit -work dspbuilder  "$libdir/dspbuilder.vhd"
                                     }
```

```
exec vmap dspbuilder dspbuilder

# Create work lib
if {[file exist work] ==0} {exec vlib work}


# Compile VHDL design files
vcom -93 -explicit  -work work "$workdir/singen.vhd"
vcom -93 -explicit  -work work "$workdir/tb_singen.vhd"

# load simulation
vsim -t $TimeResolution work.tb_singen

# Set waveform display
add wave -label clock /tb_singen/clock
add wave -label "global reset (sclrp)" /tb_singen/SystemReset
add wave /tb_singen/iNoises
add wave -radix dec /tb_singen/iSinIns
add wave  -radix dec /tb_singen/oSinDelays
add wave  -radix dec /tb_singen/oStreamMods


# Run simulation
run $SimTime ns
```