



DSP Design Flow

User Guide



101 Innovation Drive
San Jose, CA 95134
www.altera.com

Document Date:

June 2009

Copyright © 2009 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

Chapter 1. Introduction

Digital Signal Processing	1-1
FPGA Architecture Features	1-1
Software Design Flow with DSP Processors	1-3
DSP Design Flow in FPGAs	1-4
Software Flow in FPGAs	1-4
Software Combined with Hardware Acceleration	1-4
Hardware Design Flow	1-5
Benefits of FPGAs in DSP Designs	1-5

Chapter 2. DSP Design Building Blocks

MegaCore Functions	2-1
Design Flows	2-2

Chapter 3. DSP Builder

DSP Builder Design Flow	3-1
Standard and Advanced Blocksets	3-1
Standard Blockset	3-2
Advanced Blockset	3-3
Installing DSP Builder	3-3
DSP Builder Standard and Advanced Blockset Interoperability	3-4
Combined Blockset Example	3-6
Archiving Combined Blockset Designs	3-10
Tool Integration	3-10
Simulink	3-10
ModelSim	3-11
Quartus II	3-12
SOPC Builder	3-12

Appendix A. Using Hardware in the Loop with the DSP Builder Advanced Blockset

Combined Blockset Example	A-1
Advanced Blockset Example	A-6

Additional Information

Revision History	Info-1
How to Contact Altera	Info-1
Typographic Conventions	Info-1

This document is an introduction to DSP design for implementation in Altera® FPGAs, using Altera DSP Builder. It introduces the DSP Builder standard and advanced blockset tools, and the DSP IP libraries that are provided with these tools.

Digital Signal Processing

The digital signal processing (DSP) market includes rapidly evolving applications such as 3G Wireless, voice over Internet protocol (VoIP), multimedia systems, radar and satellite systems, medical systems, image-processing applications, and consumer electronics. These applications cover a broad spectrum of performance and cost requirements.

Specialized DSP processors are used for implementing many of these applications. Although these DSP processors are programmable through software, their hardware architecture is not flexible. Therefore, DSP processors are limited by fixed hardware architecture such as bus performance bottlenecks, a fixed number of multiply accumulate (MAC) blocks, fixed memory, fixed hardware accelerator blocks, and fixed data widths. The DSP processor's fixed hardware architecture is not suitable for certain applications that might require customized DSP function implementations.

FPGAs provide a reconfigurable solution for implementing DSP applications as well as higher DSP throughput and raw data processing power than DSP processors. Because FPGAs can be reconfigured in hardware, they offer complete hardware customization while implementing various DSP applications. Therefore, DSP systems implemented in FPGAs can have customized architecture, customized bus structure, customized memory, customized hardware accelerator blocks, and a variable number of MAC blocks.

FPGA Architecture Features

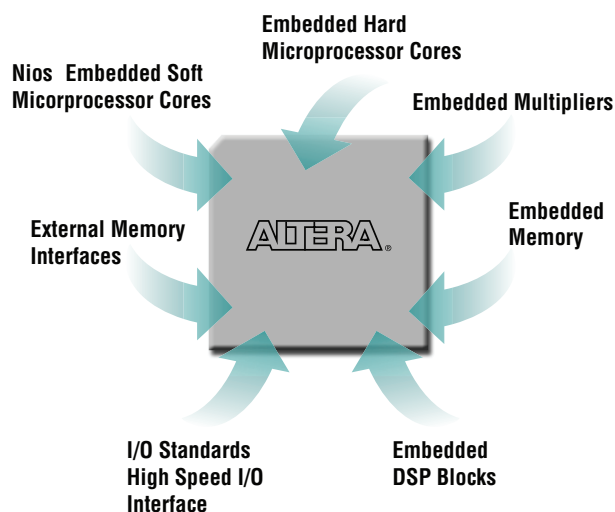
FPGA devices consist of logic elements (LEs) and memory that can be configured to operate in different modes corresponding to a required functionality. This hardware flexibility allows you to implement any hardware design described using a suitable hardware description language (HDL) such as VHDL or Verilog HDL. Thus, the same FPGA can implement a DSL router, a DSL modem, a JPEG encoder, a digital broadcast system, or a backplane switch fabric interface.

High-density FPGAs, such as Altera's Stratix® families, incorporate embedded silicon features that can implement complete systems inside an FPGA, creating a system on a programmable chip (SOPC) implementation. Embedded silicon features such as embedded memory, DSP blocks, and embedded processors are ideally suited for implementing DSP functions such as finite impulse response (FIR) filters, fast Fourier transforms (FFTs), correlators, equalizers, encoders, and decoders.

The embedded DSP blocks also provide other functionality such as accumulation, addition/subtraction, and summation that are common arithmetic operations in DSP functions. For example, Stratix device DSP blocks offer up to 224 multipliers that can perform 224 multiplications in a single clock cycle. Compared to DSP processors that only offer a limited number of multipliers, Altera FPGAs offer much more multiplier bandwidth.

One determining factor of the overall DSP bandwidth is the multiplier bandwidth, therefore the overall DSP bandwidth of FPGAs can be much higher using FPGAs than with a DSP processors. For example, Stratix device DSP blocks can deliver 70 GMACS of DSP throughput while typical DSP processors available can deliver only up to 4.8 GMACS. [Figure 1-2](#) highlights the DSP-related features available in Altera FPGA device families.

Figure 1-1. DSP Related Features in Altera FPGA Devices



Many DSP applications use external memory devices to manage large amounts of data processing. The embedded memory in FPGAs meets these requirements and also eliminates the need for external memory devices in certain cases. For example, the Stratix device family offers up to 10 Mbits of embedded memory through the TriMatrix™ memory feature.

Embedded processors in FPGAs provide overall system integration and flexibility while partitioning the system between hardware and software. You can implement the system's software components in the embedded processors and implement the hardware components in the FPGA's general logic resources. Altera devices provide a choice between embedded soft core processors and embedded hard core processors.

You can implement soft core processors such as the Nios® II embedded processor in FPGAs and add multiple system peripherals. The Nios II processor supports a user-determinable multi-master bus architecture that optimizes the bus bandwidth and removes potential bottlenecks found in DSP processors. You can use multi-master buses to define as many buses and as much performance as needed for a particular application. Off-the-shelf DSP processors make compromises between size and performance when they choose the number of data buses on the chip, potentially limiting performance.

Soft embedded processors in FPGAs provide access to custom instructions such as the *MUL* instruction in Nios II processors that can perform a multiplication operation in two clock cycles using hardware multipliers. FPGA devices provide a flexible platform to accelerate performance-critical functions in hardware because of the configurability of the device's logic resources. Unlike DSP processors that have predefined hardware accelerator blocks, FPGAs can implement hardware accelerators for each application, allowing the best achievable performance from hardware acceleration. You can implement hardware accelerator blocks by designing such blocks using parameterizable IP functions or from scratch using HDL.

Altera offers the MegaCore® functions for DSP design that support error detection and correction (Reed Solomon and Viterbi), filters (CIC and FIR), signal generation (NCO), transforms (FFT) and video and image processing (Alpha Blending Mixer, Chroma Resampler, Clipper, Clocked Video Input, Clocked Video Output, Color Plane Sequencer, Deinterlacer, 2D FIR Filter, 2D Median Filter, Frame Buffer, Gamma Corrector, Line Buffer Compiler, Scaler).

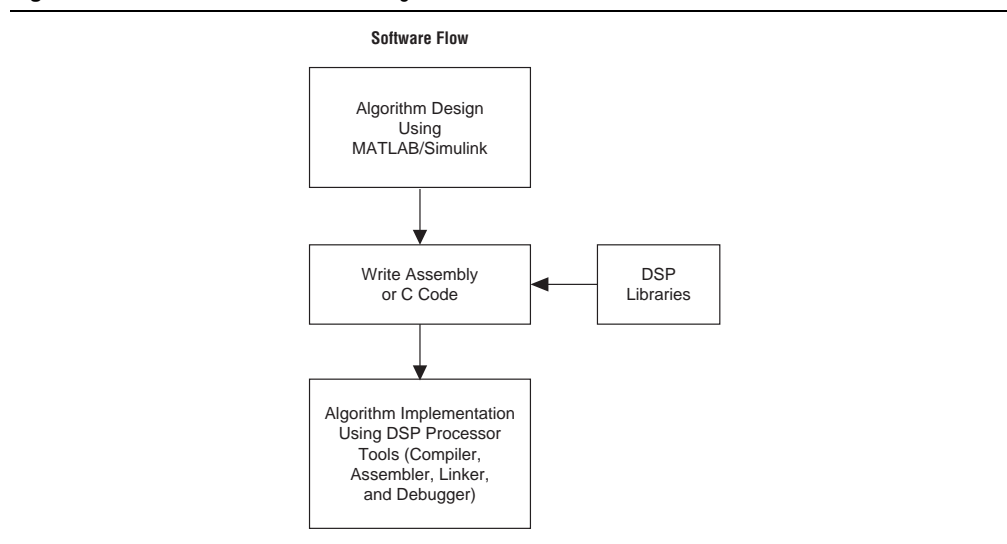
For more information about these MegaCore functions, refer to [Chapter 2, DSP Design Building Blocks](#).

Each of these functions can be parameterized (using the MegaWizard™ interface) to design the most efficient hardware implementation for a given set of parameters. This provides maximum flexibility, allowing you to customize IP without changing a design's source code. You can integrate a parameterized IP core in any hardware description language (HDL) or netlist file generated using any EDA tool. You can easily port the IP to new FPGA families, leading to higher performance and lower cost. The flexibility of programmable logic and soft IP cores allows you to quickly adapt your designs to new standards without waiting for long lead times usually associated with DSP processors.

Software Design Flow with DSP Processors

[Figure 1-2](#) shows the typical software design flow that DSP programmers follow.

Figure 1-2. Software-Based DSP Design Flow



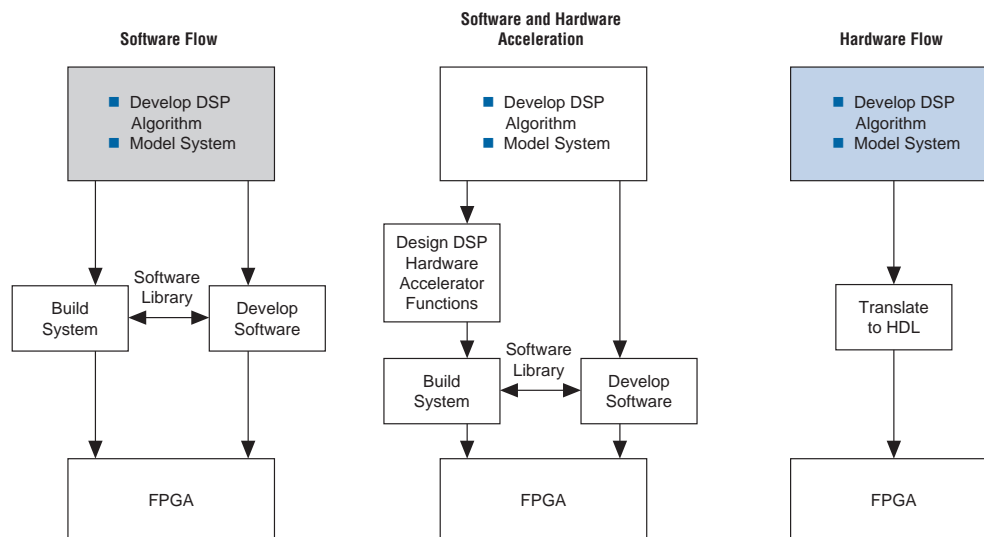
Algorithm development tools such as MATLAB are often used to optimize DSP algorithms and Simulink for system-level modeling. The algorithms and the system-level models are then implemented in C/C++ or Assembly code using an integrated development environment, that provides design, simulation, debug, and real-time verification tools. You can use standard C-based DSP libraries to shorten design cycles and derive the benefits of design re-use.

DSP Design Flow in FPGAs

Traditionally, DSP systems were implemented in FPGAs using the hardware flow based on a HDL language such as Verilog HDL and VHDL. Altera DSP tools such as DSP Builder, SOPC Builder, and a complete software development platform now enable you to follow a software-based design flow while targeting FPGAs.

Figure 1-3 outlines the various design-flow options available for FPGAs.

Figure 1-3. FPGA-Based DSP Design Flow Options



Software Flow in FPGAs

Altera FPGAs with embedded processors support a software-based design flow. Altera provides software development tools for compiling, debugging, assembling and linking software designs. These software designs can then be downloaded to an FPGA using either on-chip RAM or an external memory device.

Software Combined with Hardware Acceleration

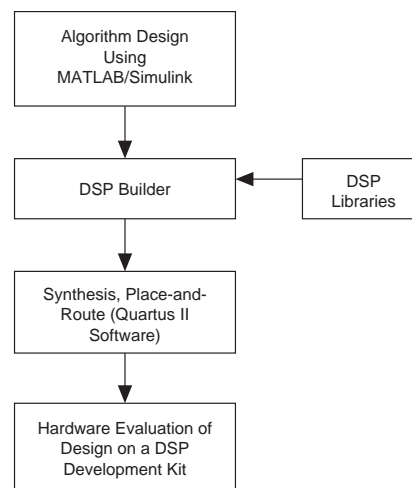
Embedded processors and hardware acceleration offer the flexibility, performance, and cost effectiveness in a development flow that is familiar to software developers. You can combine a software design flow along with hardware acceleration. In this flow, you first profile C code and identify the functions that are the most performance-intensive. Then, you can use Altera's DSP IP or develop your own custom instructions to accelerate those tasks in the FPGA.

The system control code along with the other low-performance DSP algorithms can be run on a Nios II embedded processor.

Altera also provides system-level tools such as SOPC Builder for system-level partitioning and integration. You can use SOPC Builder to build entire hardware systems by combining the embedded processor, such as a Nios II embedded processor, system peripherals, as well as IP MegaCore functions.

Altera's DSP Builder tool provides an interface from Simulink directly to the FPGA hardware (Figure 1-4). The DSP Builder tool simplifies hardware implementation of DSP functions, provides a system-level verification tool to the system engineer who is not necessarily familiar with HDL design flow, and allows the system engineer to implement DSP functions in FPGAs without learning HDL. Additionally, you can incorporate the designs created by DSP Builder into a SOPC Builder system for a complete DSP system implementation.

Figure 1-4. DSP Builder Design Flow for Altera FPGAs



Hardware Design Flow

You can also develop a pure hardware implementation of a DSP system using a HDL-based design flow. Altera provides a complete set of FPGA development tools including the Quartus® II software and interfaces to other EDA tools such as Synopsys, Synplify, and Precision Synthesis. These tools enable hardware design, simulation, debug, and in-system verification of the DSP system. The suite of pre-optimized DSP MegaCore functions can simplify this development process. You can also follow the DSP Builder design flow shown in Figure 1-4 and implement hardware-only DSP systems in FPGAs without learning HDL.

Benefits of FPGAs in DSP Designs

FPGA devices provide a reconfigurable DSP solution for various DSP applications. FPGA devices incorporate a variety of embedded features such as embedded processors, DSP blocks, and memory blocks. These device features provide very high DSP capability in FPGAs compared to DSP processors. Using FPGAs, you can customize their hardware for optimal implementation of their applications.

Using embedded processors such as the Nios II embedded processor, FPGAs also offer a software-based design flow similar to the traditional DSP software design flow. In addition, FPGAs offer a design flow using SOPC Builder that enables a software design flow to be combined with hardware acceleration. Using this design flow, you can implement a complete DSP system in an FPGA and thereby develop a cost-effective, high-performance DSP system.

Altera FPGAs also offer a design flow based on the DSP Builder tool that allows a DSP system engineer designing in MATLAB/Simulink to implement a system on an FPGA without learning HDL. The DSP Builder tool flow can also be combined with the SOPC Builder tool flow for implementing a complete DSP system.

Hence FPGAs from Altera provide the benefits of system integration, flexibility while partitioning the system, lower system costs, and higher performance when compared to pure DSP processor based implementations.

For information about the DSP Builder design flow, refer to [Chapter 3, DSP Builder](#).



For information about the SOPC Builder design flow, refer to [Volume 4: SOPC Builder](#) of the *Quartus II Handbook*.

MegaCore Functions

The Altera MegaCore IP library includes the following MegaCore functions that can be used for DSP design:

- Error Detection and Correction
 - The Reed-Solomon (RS) Compiler implements a fully parameterizable encoder and decoder for forward error correction applications. RS codes are widely used for error detection and correction in a wide range of DSP applications for storage, retrieval, and transmission of data.
 - For more information, refer to the [Reed-Solomon Compiler User Guide](#).
 - The Viterbi Compiler implements high-performance, soft-decision Viterbi MegaCore functions that implement a wide range of standard Viterbi decoders. Viterbi decoding (also known as maximum likelihood decoding or forward dynamic programming) is the most common way of decoding convolutional codes by using an asymptotically optimum decoding technique. In its basic form, Viterbi decoding is an efficient, recursive algorithm that performs an optimal exhaustive search.
 - For more information, refer to the [Viterbi Compiler User Guide](#).
- Filters
 - Cascaded integrator-comb (CIC) filters, also known as Hogenauer filters, are computationally efficient for extracting baseband signals from narrow band sources using decimation, and for constructing narrow-band signals from processed baseband signals using interpolation. CIC filters use only adders and registers, and require no multipliers to handle large rate changes. Therefore, CIC is a suitable and economical filter architecture for hardware implementation, and is widely used in sample rate conversion designs such as digital down converters (DDC) and digital up converters (DUC).
 - For more information, refer to the [CIC MegaCore Function User Guide](#).
 - The FIR Compiler provides a fully integrated finite impulse response (FIR) filter development environment. You can use the FIR Compiler to implement a variety of filter architectures, including fully parallel, serial, or multibit serial distributed arithmetic, and multicycle fixed/variable filters. The FIR Compiler includes a coefficient generator.
 - For more information, refer to the [FIR Compiler User Guide](#).

- Signal Generation
 - The NCO MegaCore function generates numerically controlled oscillators (NCOs) customized for Altera devices. You can implement a variety of architectures, including ROM-based, CORDIC-based, and multiplier-based NCOs.
 -  For more information, refer to the *NCO MegaCore Function User Guide*.
- Transforms
 - The FFT MegaCore function is a high performance, highly parameterizable fast Fourier transform (FFT) processor. The FFT MegaCore function implements a complex FFT or inverse FFT (IFFT) for high-performance applications using a fixed transform size or variable streaming architecture.
 -  For more information, refer to the *FFT MegaCore Function User Guide*.
- Video and Image Processing
 - The Video and Image Processing Suite is a collection of MegaCore functions that facilitate the development of video and image processing designs. The MegaCore functions are suitable for use in a wide variety of image processing and display applications and include:
Alpha Blending Mixer, Chroma Resampler, Clipper, Clocked Video Input, Clocked Video Output, Color Plane Sequencer, Deinterlacer, 2D FIR Filter, 2D Median Filter, Frame Buffer, Gamma Corrector, Line Buffer Compiler, Scaler, Clipper, Test Pattern Generator.
 -  For more information, refer to the *Video and Image Processing Suite User Guide*.

Design Flows

Altera DSP IP MegaCore® functions can be used in several design flows:

- The MegaWizard™ Plug-In Manager can be used with any of the DSP IP MegaCore functions when you want to create a MegaCore function variation that you can instantiate manually in your design.
- The CIC, FFT, FIR, NCO, Reed-Solomon, and Viterbi MegaCore functions can be included as a block in a DSP Builder design. If required, the DSP Builder design can then be instantiated in a SOPC Builder system.
- The Video and Image Processing Suite MegaCore functions can be instantiated directly in a SOPC Builder system.

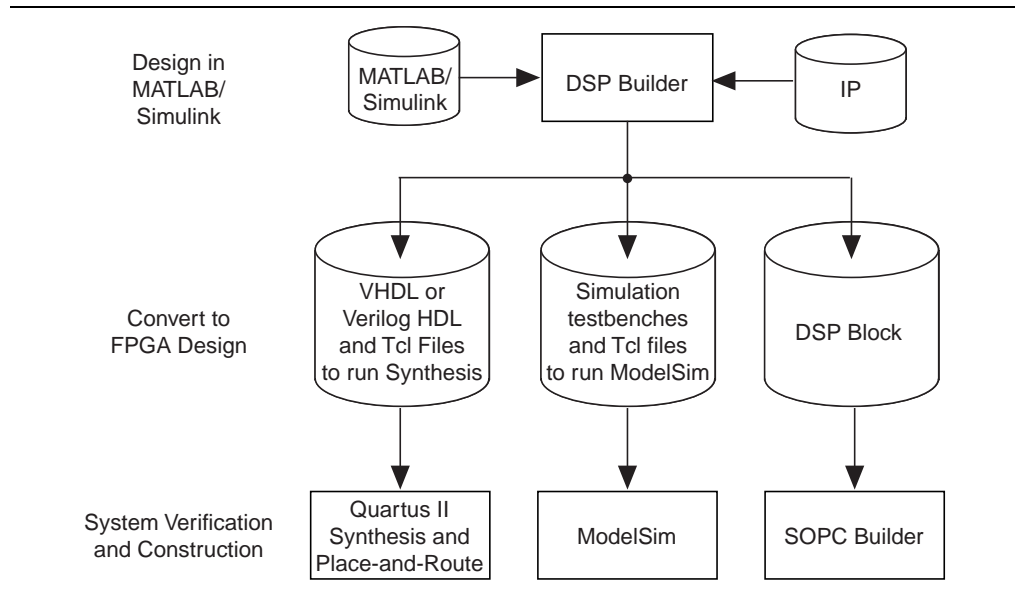
 For more information about using MegaCore functions in these design flows, refer to the appropriate MegaCore function user guide.

DSP Builder Design Flow

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

DSP Builder integrates the algorithm development, simulation, and verification capabilities of The MathWorks MATLAB® and Simulink® system-level design tools with the Altera Quartus II software and third-party synthesis and simulation tools. You can combine Simulink blocks with DSP Builder blocks and IP blocks to verify system level specifications and perform simulation (Figure 3-1).

Figure 3-1. DSP Builder Design Flow



Standard and Advanced Blocksets

When you install DSP Builder two separate blocksets (Altera DSP Builder Blockset and Altera DSP Builder Advanced Blockset) are added to the Simulink library browser and can be used separately or together.

The standard DSP Builder blockset includes libraries of design building and interface blocks and a library of blocks that represent each of the DSP MegaCore functions.

The advanced DSP Builder blockset does not interface directly with the DSP MegaCore functions but instead includes its own timing-driven IP blocks that can be used to generate high performance FIR, CIC, NCO, and FFT models.

Table 3-1 on page 3-2 summarizes the key features of the standard and advanced blocksets.

Table 3-1. Key Features of the Standard and Advanced Blocksets

Blockset	Key Features
Standard Blockset	<ul style="list-style-type: none"> Cycle accurate behavioral models Multiple clock domain management Control rich with backpressure support Access to specific hardware device features Hardware-in-the-loop support enables FPGA hardware cosimulation HDL import of VHDL or Verilog HDL design entities Tabular and graphical state machine support Rapid prototyping using Altera DSP development boards SignalTap II debug facilities DSP MegaCore function variations can be directly instantiated
Advanced Blockset	<ul style="list-style-type: none"> Specification driven design with automatic pipelining and folding High level synthesis technology High performance timing-driven IP models Multichannel designs with automatically vectorized inputs Automatic generation of memory-mapped interfaces Uses Simulink fixed-point types Single system clock for the main data path logic Feed forward data path with minimum control Portability across different device families High level resource trade-offs such as hard versus soft multipliers

You can use both blocksets in subsystems of the same design when you want to combine features from each blockset. For example, when you want to combine an IP model from the advanced blockset with development board support or hardware-in-the-loop from the standard blockset.

Standard Blockset

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

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

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



For more information about the advanced blockset, refer to the *DSP Builder Reference Manual* and the *DSP Builder User Guide*.

Advanced Blockset

The DSP Builder Advanced Blockset consists of a number of Simulink libraries that allow you to implement DSP designs quickly and easily. The blockset is based on a high level synthesis technology that optimizes the high level, untimed netlist into low level, pipelined hardware targeted to your chosen FPGA device and chosen clock rate. The hardware is written out as VHDL, along with scripts that integrate with the Quartus II software and the ModelSim® simulator.

The combinations of these features allows you to create a design without needing intimate device knowledge, and then generate a high quality implementation that runs on a variety of FPGA families with different hardware architectures.

After specifying your desired clock frequency, number of channels and other top-level design constraints, the generated RTL is automatically pipelined to achieve timing closure. By analyzing the system-level constraints, the tool also optimizes folding (that is, time division multiplexing) to achieve optimum logic utilization, with no need for manual RTL editing.

The synthesis technology also allows you to easily increase or decrease the number of channels—for example, in your FIR filter or digital up conversion signal chain—simply by using a parameter file within the Simulink design. DSP Builder adds the required time-division multiplexing control logic and generates the updated RTL in a matter of minutes.

The advanced blockset includes a library of basic control blocks and two component libraries - ModelIP and ModelPrim. These are useful in the following circumstances:

- **ModelIP:** The ModelIP library consists of a set of multichannel, multirate cycle-accurate filters, mixers and a numerically controlled oscillator (NCO) that allow you to quickly create designs for digital front end applications. Several implementation examples including up and down converters are provided.
- **ModelPrim:** The ModelPrim library allows you to create fast efficient designs captured in the behavioral domain rather than the implementation domain by combining zero latency primitive blocks. For example, you can use a delay block and let the tool decide how to implement that delay.

The advanced blockset is particularly suited for streaming algorithms characterized by continuous data streams and occasional control. (For example, RF card designs that comprise long filter chains.)

 For more information about the advanced blockset, refer to the *DSP Builder Advanced Blockset Reference Manual* and the *DSP Builder Advanced Blockset User Guide*.

Installing DSP Builder

DSP Builder can be installed by selecting an option in the install program for the Quartus II software. The installer allows you to install either or both blocksets.

 For more information, refer to the *DSP Builder Installation and Licensing* manual.

DSP Builder Standard and Advanced Blockset Interoperability

It is possible to combine blocks from the DSP Builder standard and advanced blocksets in the same design. The top-level model can contain the `Control` and `Signals` blocks from the advanced blockset. However, the `Device` block and the functional blocks in the advanced blockset design must be embedded in a lower level subsystem.

The `Run ModelSim` block from the advanced blockset does not work in a combined blockset design. However, you can use the `TestBench` block from the standard blockset to generate a testbench and compare the simulation results with `ModelSim`.

You can still use the `Run Quartus II` block from the advanced blockset in a combined blockset design but it only creates a Quartus II project for the advanced blockset subsystem containing the `Device` block. Use a `Signal Compiler` block from the standard blockset to create a Quartus II project for the whole combined blockset design.

The mechanism for embedding an advanced blockset subsystem within a top-level DSP Builder design is similar to that for embedding a HDL subsystem as a black box, with the `Device` block from the advanced blockset taking the place of the `HDL Entity` block from the standard blockset.



For information about the DSP Builder HDL subsystem flow, refer to the *Using Black Boxes for HDL Subsystems* section in the *DSP Builder User Guide*.



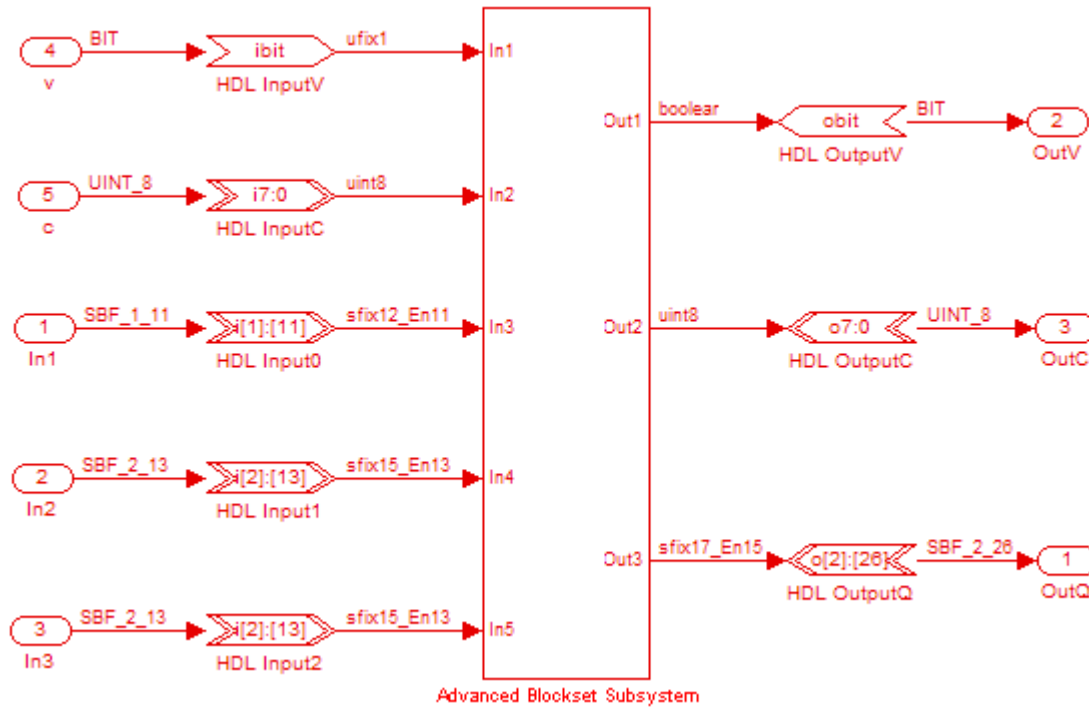
The advanced blockset design is generated when you simulate the design in Simulink. This simulation must be done before the top-level model is generated by running `Signal Compiler`.


There are a number of settings and parameters that must match across the two blocksets in an integrated design:

- The hardware destination directory specified in the `Control` block must be specified as an absolute path using forward slash (/) separator characters.
- The device family specified in the `Device` block must match that specified in the top level `Signal Compiler` block and the device used on your development board. However, the specific device can be set to **Auto**, or have different values. The specific target device in the generated Quartus II project is that specified in the `Signal Compiler` block. HIL specifies its own Quartus II project which can have a different device provided that the device family is consistent.
- The reset type specified in the advanced blockset `Signals` block must be active **High**.
- When you run the `TestBench` for a combined blockset design, mismatches when the `valid` signal is low are expected.
- The standard blockset does not support vector signals. Any vectors in the advanced blockset design must be converted using multiplexer and demultiplexer blocks.

The subsystem that contains the advanced blockset design must be connected using HDL Input and HDL Output blocks as shown in Figure 3–2 on page 3–5. The signal dimensions on the boundary between the advanced blockset subsystem and the HDL Input/HDL Output blocks must match.

Figure 3–2. Advanced Blockset Subsystem Enclosed by HDL Input and Output Blocks




 The signal types are shown on either side of the HDL Input/HDL Output blocks after you have simulated the subsystem. If the signal types are not displayed, check that **Port Data Types** is turned on in the Simulink Format menu.

If the signal types do not match, there may be error messages of the form:

```
"Error (10344): VHDL expression error at <subsystem>_<HDL I/O name>.vhd
(line no.): expression has N elements, but must have M elements"
```

For example in Figure 3–2, an error is issued because the signal type for the HDL OutputQ block is incorrect and should be changed from **Signed Fractional [2].[26]** to **Signed Fractional [2].[15]**.

After this change, the signal type is shown as **SBF_2_15** (representing a signed binary fractional number with 2 integer bit and 15 fractional bits) in the standard blockset part of the design (before the HDL Input block). The same signal is shown as **sfix17_En15** (representing a Simulink fixed-point type with word length 17 and 15 fractional bits) in the advanced blockset design (after the HDL Input block).

 For more information about the fixed-point notation used by the standard blockset, refer to the *Fixed-Point Notation* section in the *DSP Builder User Guide*. For more information about Simulink fixed-point types, refer to the MATLAB help.

Combined Blockset Example

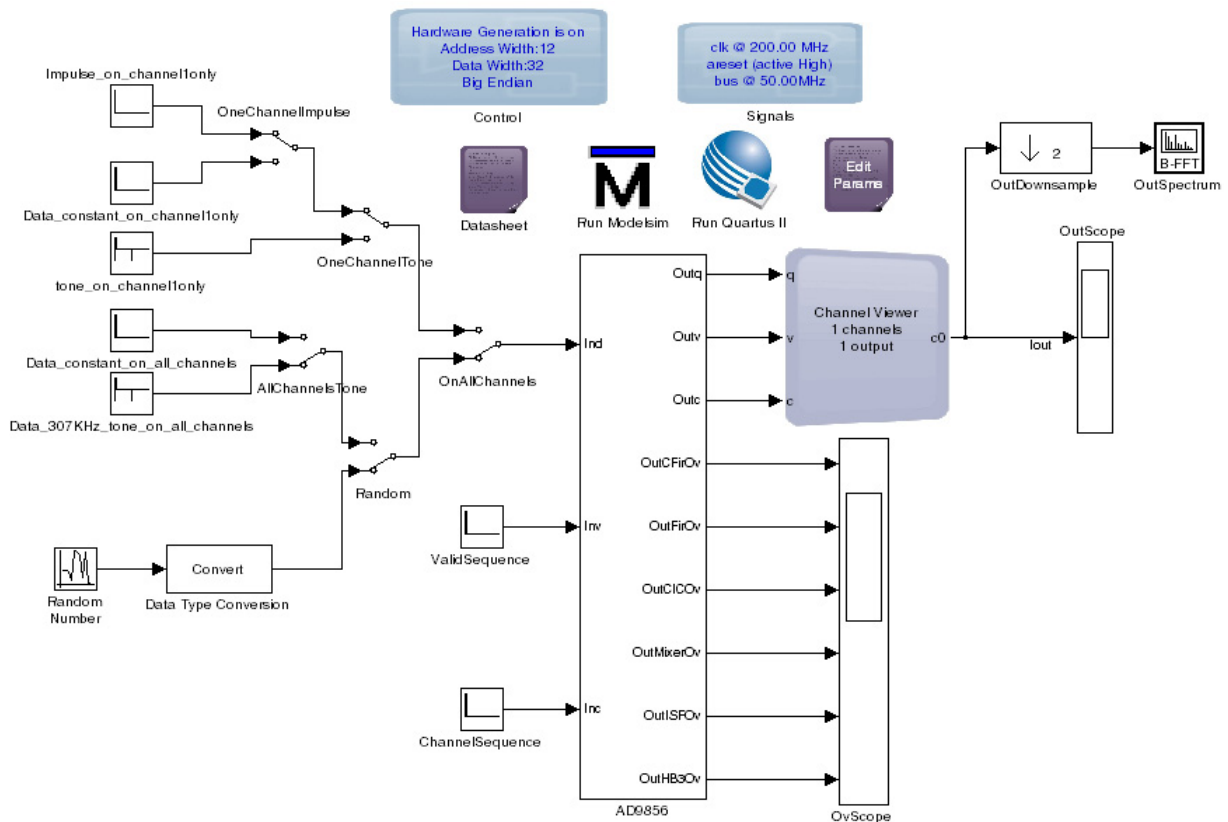
Figure 3–3 shows the 2-Channel Digital Up Converter advanced blockset demonstration design (**demo_AD9856**).

The synthesizable system in this design is the AD9856 subsystem. Its inputs are *Ind*, *Inv* and *Inc*. There are nine outputs with three feeding a *Channel Viewer* and six feeding a *Scope*.

To make the design interoperable with a *Signal Compiler* block in the top level, perform the following steps:

1. Select the subsystem, AD9856, the scope, *OvScope*, and the *Channel Viewer*. Click **Create Subsystem** on the popup menu to create a *Subsystem* block that contains them.

Figure 3–3. AD9856 Design Example

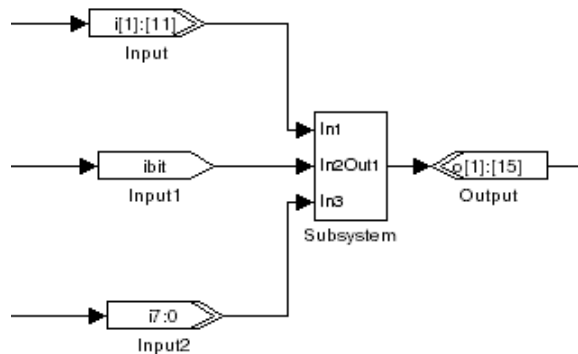


2. Add three Input blocks from the DSP Builder **IO & Bus** library immediately before the new *Subsystem* block: Input (with **Signed Fractional [1][11]** type), Input1 (with **Single Bit** type), and Input2 (with **Unsigned Integer 8** type).
3. Add an Output block immediately after the *Subsystem* block with **Signed Fractional [1][15]** type.




Steps 2 and 3 specify the boundaries between Simulink blocks and DSP Builder blocks.

Figure 3-4. Input, Output and Subsystem Blocks in Top-Level Model

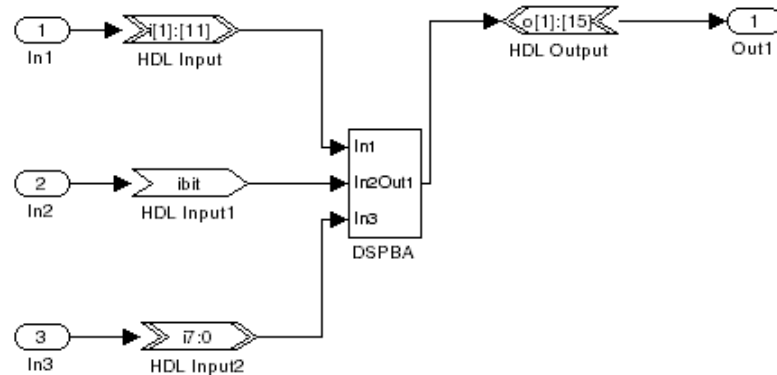


4. Open the Subsystem block and select the AD9856, OvScope, and Channel Viewer blocks inside the subsystem. Click **Create Subsystem** on the popup menu to push these blocks down another level. Rename this subsystem (to for example DSPBA).
5. Add three HDL Input blocks from the DSP Builder **AltLab** library between the Simulink input ports and the DSPBA subsystem.
 - a. These should have the same types as in Step 2: HDL Input (**Signed Fractional [1][11]**), HDL Input1 (**Single Bit**), and HDL Input2 (**Unsigned Integer 8**).
 - b. On the signed fractional HDL Input, set the **External Type** parameter to **Simulink Fixed Point Type**.
6. Add a HDL Output block between the subsystem and the subsystem output port with the same type as in Step 3 (**Signed Fractional [1][15]**).


 Steps 5 and 6 specify the boundaries between blocks from the standard and advanced blocksets. The HDL Input and HDL Output blocks must be in a lower level subsystem than the Input and Output blocks. If they are at the same level, a “NullPointerException” error is issued when you run Signal Compiler.

The new subsystem should appear similar to [Figure 3-5](#).

Figure 3-5. HDL Input, HDL Output and DSPBA Blocks in Subsystem

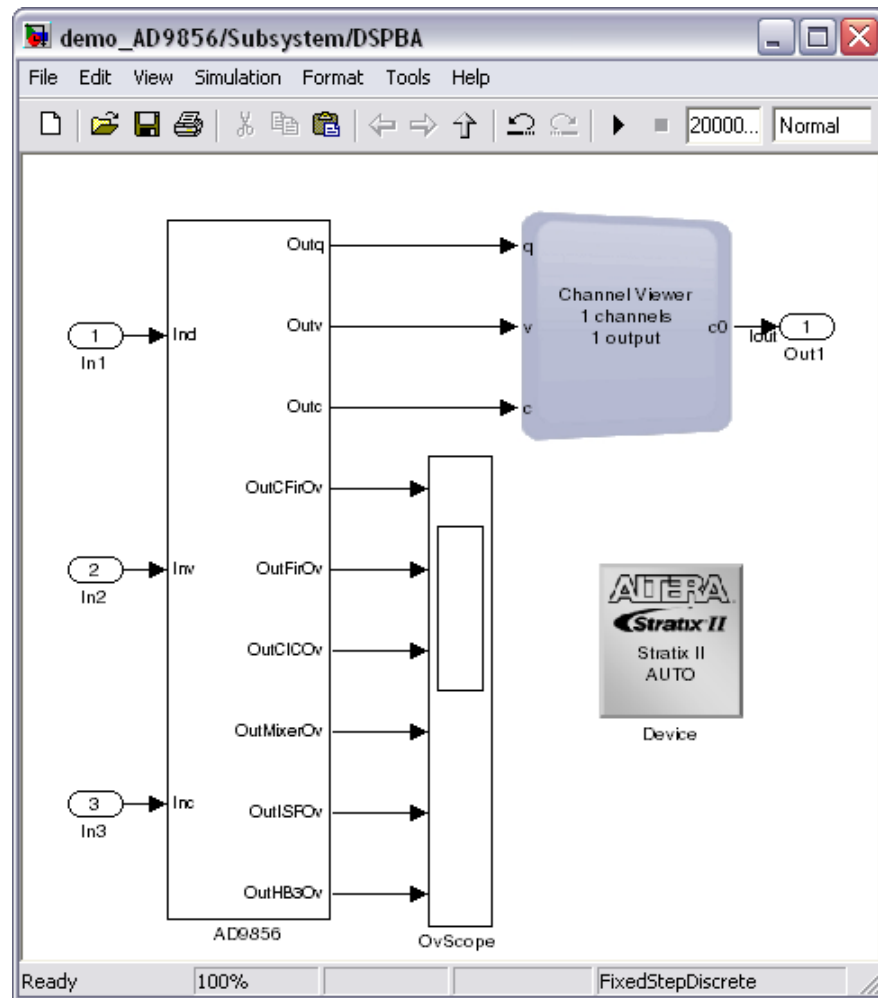


7. Open the DSPBA subsystem and the AD9856 subsystem below it.
8. Move the Device block from the AD9856 subsystem up a level into the DSPBA subsystem you have created by making a copy of the existing Device block and then deleting the old one.


 The Device block is used to detect the presence of a DSP Builder advanced subsystem and should be in the highest level of the advanced blockset design that does not contain any blocks from the standard blockset.

The DSPBA subsystem should appear similar to Figure 3-6.

Figure 3-6. DSPBA Subsystem



9. Open the Control block in the top level of the design and change the **Hardware Destination Directory** to an absolute path. For example: **C:/rtl**

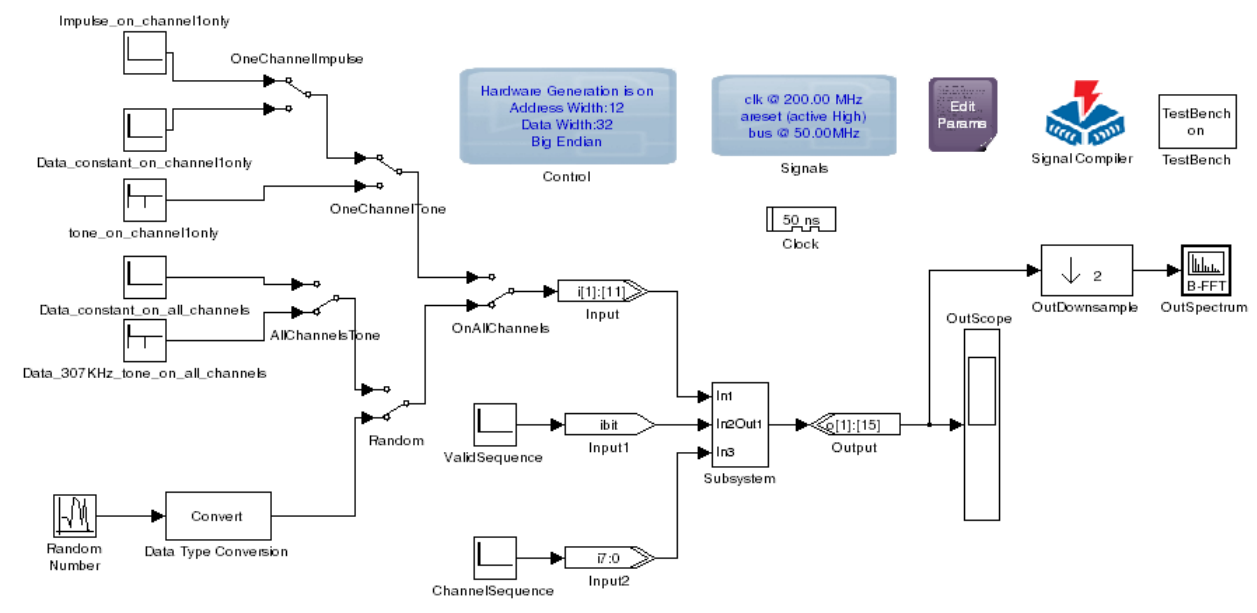
 You must use forward slashes (/) in this path.

10. Add Signal Compiler, TestBench and Clock blocks from the DSP Builder AltLab library to the top-level model.

11. In the `Signal Compiler` block, set the **Family** to **Stratix II** to match the family specified in the `Device` block.
12. In the `Clock` block, set the **Real-World Clock Period** to **50 ns** so that it matches the **Clock Frequency** specified in the `Signals` block. Set the **Reset Name** to **aclr Active Low**.
13. Remove the `Run ModelSim` and `Run Quartus II` blocks that are no longer required in the combined blockset design.

The updated design should look similar to [Figure 3-7](#).


Figure 3-7. Updated AD9856 Design Example



14. Simulate the design to generate HDL for the advanced subsystem.
15. Compile the system using `Signal Compiler`. It should compile successfully with no errors.

You can also use the `Testbench` block to compare the Simulink simulation with `ModelSim`. However, note that there are several cycles of delay in the `ModelSim` output that are not present in Simulink because the advanced blockset simulation is not cycle-accurate.

The subsystem containing the HDL `Input`, HDL `Output` blocks and the advanced blockset subsystem is treated as a black box. You can only add additional blocks from the advanced blockset to the subsystem inside this black box. However, you can add blocks from the standard blockset in the top-level design or in additional subsystems outside this black box.

 For an example of how you can use Hardware in the loop in a design combining the standard and advanced blocksets, refer to [“Using Hardware in the Loop with the DSP Builder Advanced Blockset”](#) on page A-1.

Archiving Combined Blockset Designs

To create an archive for a combined design using both the standard and advanced blockset, you should first simulate the design to generate HDL for the advanced blocks in the design. Then open the `Signal Compiler` interface, select the **Export** tab and click **Export**.

This exports HDL to the specified directory including a copy of all the standard blockset HDL and a Quartus II IP file (`.qip`) which contains all the assignments and other information required to process the standard blockset model in the Quartus II compiler.

This `.qip` file does not include the HDL files for the advanced blockset. To include these files, it is necessary to add the directories containing the generated HDL for the advanced block as user libraries in the Quartus II software.

You can do this by choosing **Add/Remove Files in Project** from the Project menu, select the Libraries category and add all directories that contain generated HDL from the advanced blocks. (There are separate directories for each level of hierarchy in the Simulink design).

You can archive this Quartus II project to produce a portable Quartus II archive file (`.qar`) that contains all the source HDL for the combined standard and advanced blocksets in the DSP Builder system. This is achieved by choosing **Archive Project** from the Project menu in the Quartus II software.

Tool Integration

The DSP Builder standard and advanced blocksets are designed to operate with the Simulink, ModelSim, Quartus II and SOPC Builder software.

Simulink

DSP Builder is interoperable with other Simulink blocksets. In particular, the basic Simulink blockset is very useful for creating interactive testbenches.

The DSP Builder standard blockset uses signed integer, unsigned integer or signed fractional signal types can be connected to other Simulink blocks using type casting Input and Output blocks.



For information about the internal signal types used by the standard blockset, refer to the *DSP Builder User Guide*.

The VHDL model for standard blockset subsystems is generated when you compare the Simulink simulation results with ModelSim using the `TestBench` block.

The DSP Builder Advanced blockset uses Simulink fixed-point types for all operations and requires licensed versions of the Simulink Fixed Point blockset and Fixed-Point Toolbox. The Signal Processing Blockset and Communications Blocksets are also recommended and are used in the demonstration designs.



For information about Simulink Fixed Point Types, the Signal Processing Blockset and the Communications Blockset, refer to the MATLAB Help.

A VHDL model is generated for subsystems described using the advanced blockset when you run a Simulink simulation.

There are many examples of using Simulink blocks in the tutorial and demonstration designs. In particular, Simulink scopes are used in the advanced blockset example to identify signals that should be added to the ModelSim Wave view for display as a digital or analog signal.

- For information about the tutorials and demonstration designs, refer to the *Example Designs* appendix in the *DSP Builder Reference Manual* and the *Example Designs* chapter in the *DSP Builder Advanced Blockset User Guide*.

ModelSim

ModelSim can be invoked from within a DSP Builder standard or advanced blockset design if the ModelSim executable (**vsim.exe**) is available on your path.

Integration between the DSP Builder standard blockset and ModelSim is performed using the **TestBench** block. A VHDL testbench is generated when you click **Compile against HDL** and a **tb_<model name>.tcl** script that can be used to load the testbench into ModelSim. You can optionally load the ModelSim GUI for interactive simulation.

- For more information, refer to the description of the *TestBench* block in the *DSP Builder Reference Manual*.

Integration between the DSP Builder Advanced blockset and ModelSim is performed using a number of generated scripts:

- **Control.do**. This script is named after the control block (normally `Control` unless renamed by the user). It is used to compile the entire design with RTL equivalents of the testbench structures, add all relevant signals to the Wave window and run for the same period as the Simulink simulation.

The script relies on some subordinate scripts that recursively compile library files, all the RTL files in the project, and add the signals to the Wave window, before the simulation is started. These scripts are easy to follow, and may be useful when building custom flows. The design can be automatically loaded in ModelSim by clicking on the `Run ModelSim` block in the top-level model.

- Only a subset of the Simulink blocks are translated into RTL that can be used for simulation in ModelSim. For a list of compatible blocks, refer to the *Run ModelSim* block description in the *DSP Builder Advanced Blockset Reference Manual*.

- **<block name>_atb.do**. This script runs the automatic testbench flow for a block. It relies on reading some stimulus files at run time to verify a hardware block. The automatic testbench flow runs a rigorous test and returns a result whether or not the outputs match.

- For more information, refer to the *Comparison with RTL* section in the *DSP Builder Advanced Blockset Reference Manual*.

Quartus II

The standard blockset is tightly integrated with the Quartus II software using the `Signal Compiler` block. You can choose the device family and device, synthesize your design, run the Quartus II fitter and program your selected device. You can also enable the SignalTap II logic analyzer or export synthesizable model of your design.


The advanced blockset is designed for building high speed, high performance DSP data paths. In most production designs there will be an RTL layer surrounding this data path to perform interfacing to processors, high speed I/O, memories, and so on. However, there is little emphasis on the generation of complete designs with PLLs and pinouts.

To complete the design, board level components can be assigned using the DSP Builder standard blockset, SOPC Builder, or RTL. The Quartus II software can then complete the synthesis and place and route process.

You can automatically load a design into the Quartus II software by clicking on the `Run Quartus II` block in the top-level model.

SOPC Builder

The DSP Builder standard blockset includes a library of Avalon® Memory-Mapped (Avalon-MM) and Avalon Streaming (Avalon-ST) interface blocks. A Tcl script `<model name>_add.tcl` is created by `Signal Compiler` which can be used to add your design to a Quartus II project. Any design that includes the Avalon interface blocks is automatically available for connection to other Avalon components in SOPC Builder.

 For an example of the standard blockset integration with SOPC Builder, refer to the *Using the Interface Library* chapter in the *DSP Builder User Guide*.

A memory-mapped interface and `class.ptf` file is created for each advanced blockset design. These can be used to expose the processor bus for connection in SOPC Builder. A DSP Builder Advanced Blockset subsystem is automatically available from the **System Contents** tab in SOPC Builder after the path to the `class.ptf` file has been added to the SOPC Builder IP search path.

 For an example of the advanced blockset integration with SOPC Builder, refer to the *ModelIP Tutorial* chapter in the *DSP Builder Advanced Blockset User Guide*.

This appendix describes how you can use Hardware in the Loop (HIL) with a design that uses the DSP Builder Advanced Blockset.

Combined Blockset Example

This example shows how you can use HIL in a design that combines blocks from the standard and advanced blocksets.

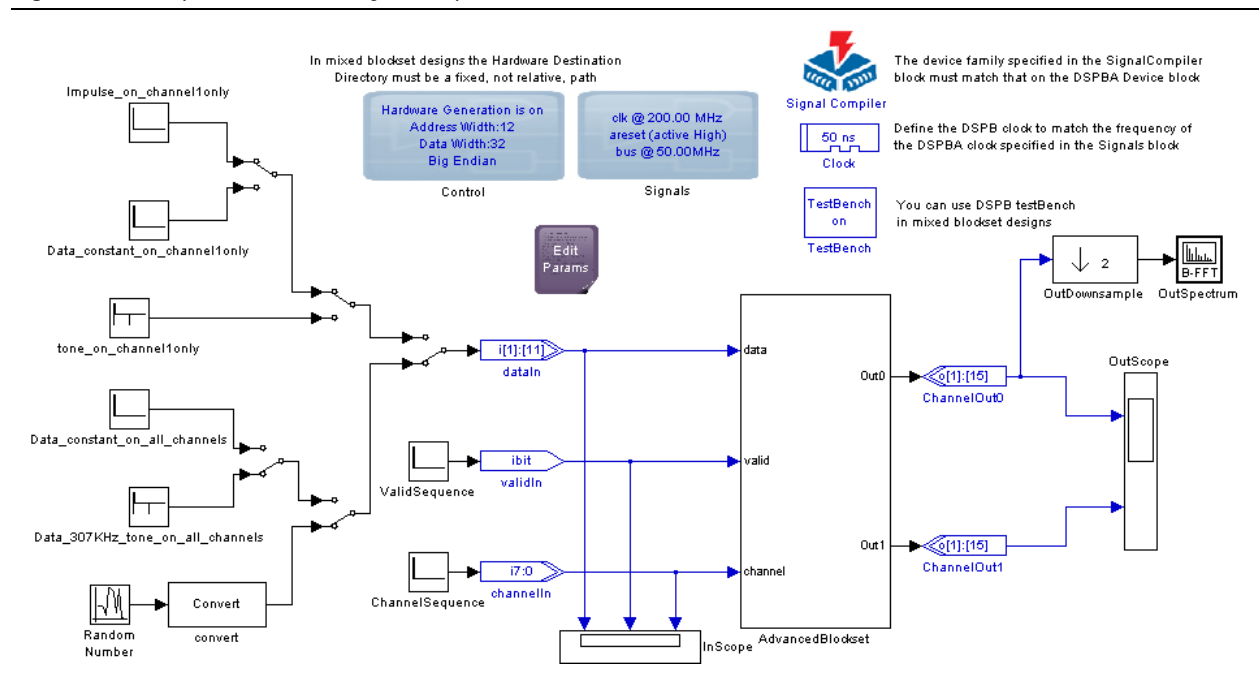
There are a number of settings and parameters that must match across the two blocksets in an integrated design. Refer to [on page 3–4](#) for full details.

Perform the following steps:

1. Open the **Adapted_AD9856.mdl** model from the **Demos\Combined Blockset** directory in the installed design examples for the standard blockset ([Figure A–1](#)).


Ensure that the **Hardware Destination Directory** specified in the **Control** block is specified as an absolute path using forward slash (/) separator characters.

Figure A–1. Adapted_AD9856 Design Example



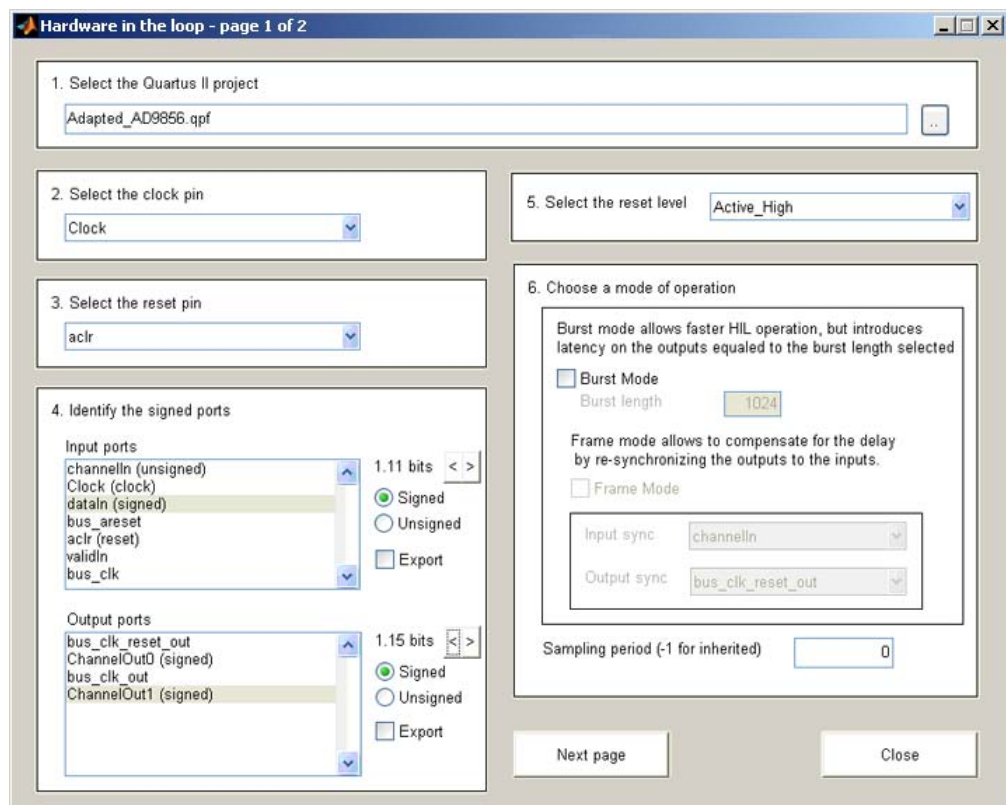
2. Simulate the design to generate hardware.

- Run Signal Compiler to create a Quartus II project for the combined design.

 The device family in the Signal Compiler block is set to **Stratix II**, which should match the device family specified in the Device block inside the advanced blockset subsystem. If you are using a different board, change the device family in both the Signal Compiler and the Device block and repeat steps 2 and 3.

- Save a copy of the model as **Adapted_AD9856_HIL.mdl** and delete the AdvancedBlockset subsystem in this model.
- Replace the AdvancedBlockset subsystem by a HIL block from the AltLab library in the DSP Builder standard blockset.
- Double-click on the HIL block to open the **Hardware in the loop** dialog box (Figure A-2). In the first page of the dialog box, select the Quartus II project (**Adapted_AD9856_dspbuilder/Adapted_AD9856.qpf**) that was created in step 3. Select the clock pin (Clock) and reset pin (aclr) names. Set the channelIn signal type to **unsigned 8 bits**, dataIn to **signed [1].[11] bits**, and ChannelOut0 and ChannelOut1 to **signed [1].[15] bits**.

Figure A-2. Hardware in the Loop Parameter Settings Page 1 for the AD9856 Example



- Close the **Hardware in the loop** dialog box and connect the dataIn, validIn, channelIn, ChannelOut0, and ChannelOut1 ports to your model as shown in Figure A-3 on page A-3.


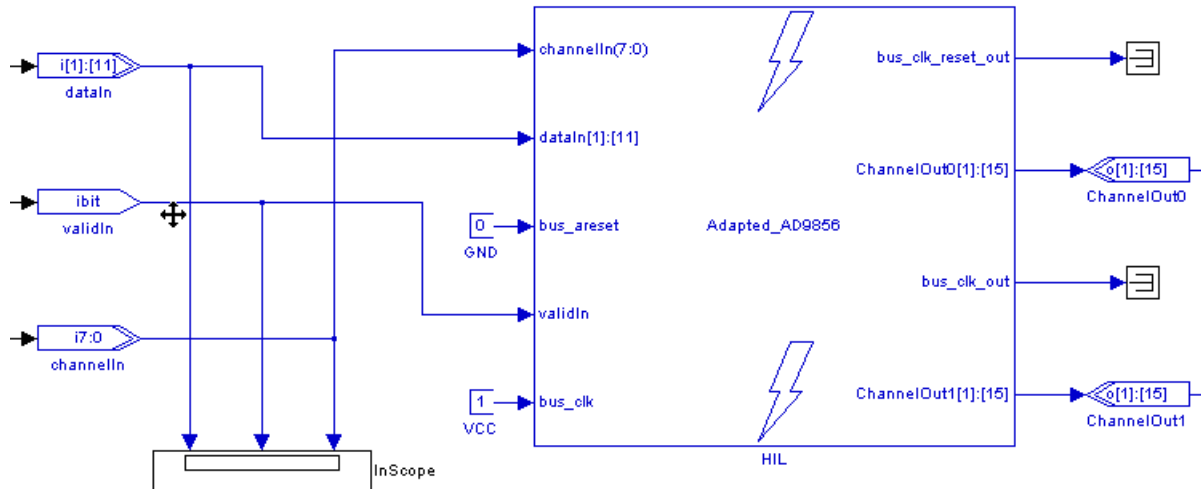
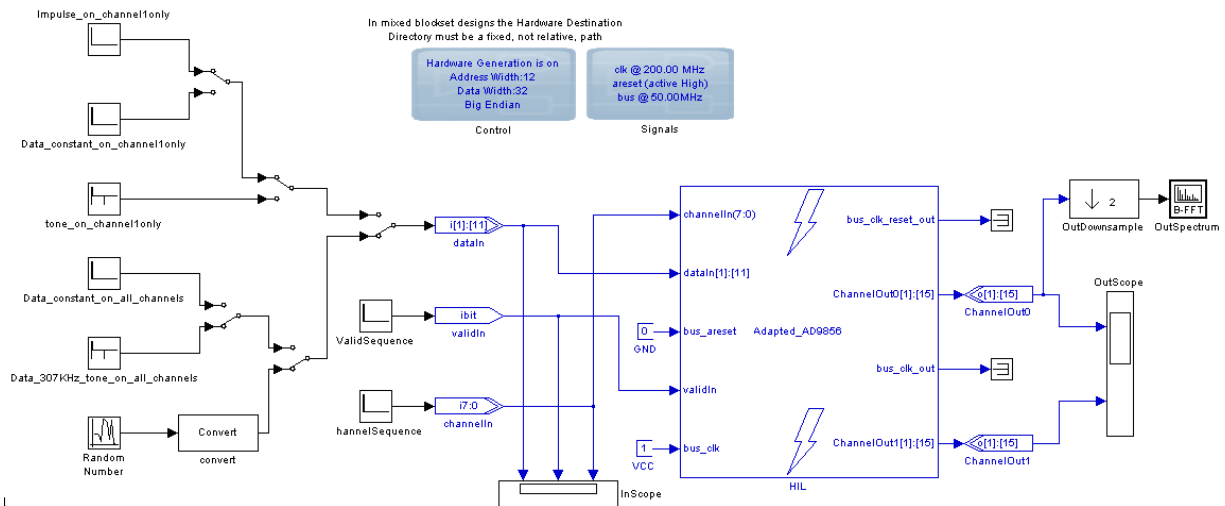
 The bus interface is currently not used in HIL simulation. Connect the bus_areset signal to a GND block and the bus_clk port to a VCC block from the standard blockset IO & Bus library. Connect the bus_clk_out and bus_clk_reset_out signals to Simulink Terminator blocks.

Figure A-3. HIL Block in the Adapted_AD9856 Model




- Clean up the HIL model by removing those blocks which are not related to the HIL, as shown in Figure A-4.

Figure A-4. Cleaned Up HIL Design



- Save the **Adapted_AD9856_HIL.mdl** model.
- Connect the DSP development board and ensure that it is switched on.

11. Re-open the **Hardware in the loop** dialog box and set the reset level as **Active_High**.

 The reset level must match the level specified in the Signals block for the original model.

12. Click on **Next** to display the second page of the **Hardware in the loop** dialog box (Figure A-5). Enter a full device name into the **FPGA device** field. Verify that the device name matches the device on the DSP development board and is compatible with the device family set in the original model.


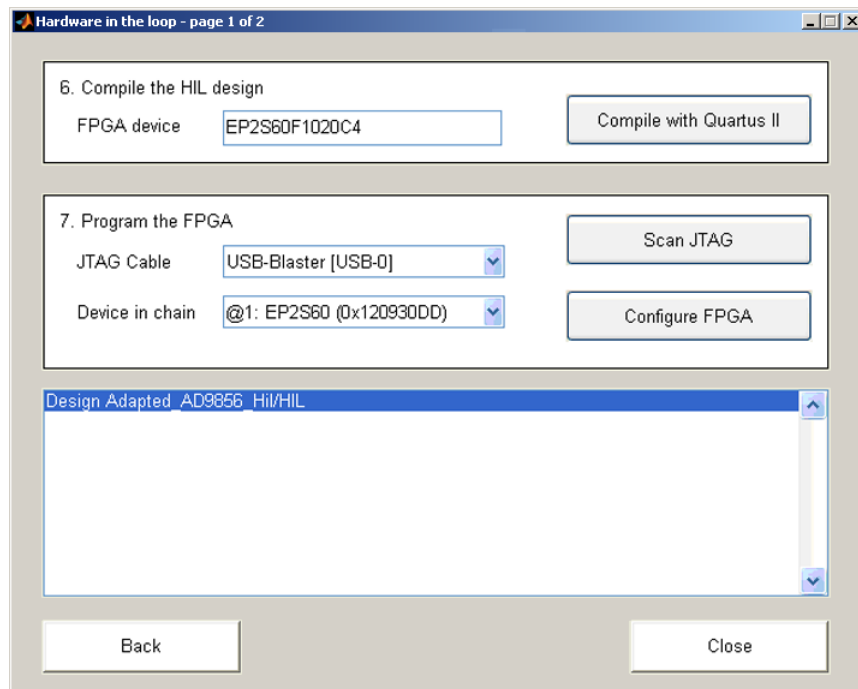
 If you need to change the device family to use a different board, then you must repeat steps 2, 3, 6, and 11.

Figure A-5. Hardware in the Loop Parameter Settings Page 2



13. Click **Compile with Quartus II** to compile the HIL model.
14. Click **Scan JTAG** to find all the hardware connected to your computer and select the required **JTAG Cable** and **Device in chain**.
15. Click **Configure FPGA** to download the compiled programming file (.sof) to the DSP development board.
16. Close the **Hardware in the loop** dialog box and save the model.
17. Simulate the HIL model. Compare the displays of the OutputSpectrum and OutScope blocks to the waveforms in the original model. They should be identical.

Figure A-6 on page A-5 shows the output spectrum waveform and Figure A-7 on page A-5 shows the output scope waveform.

Figure A-6. Output Spectrum Waveform

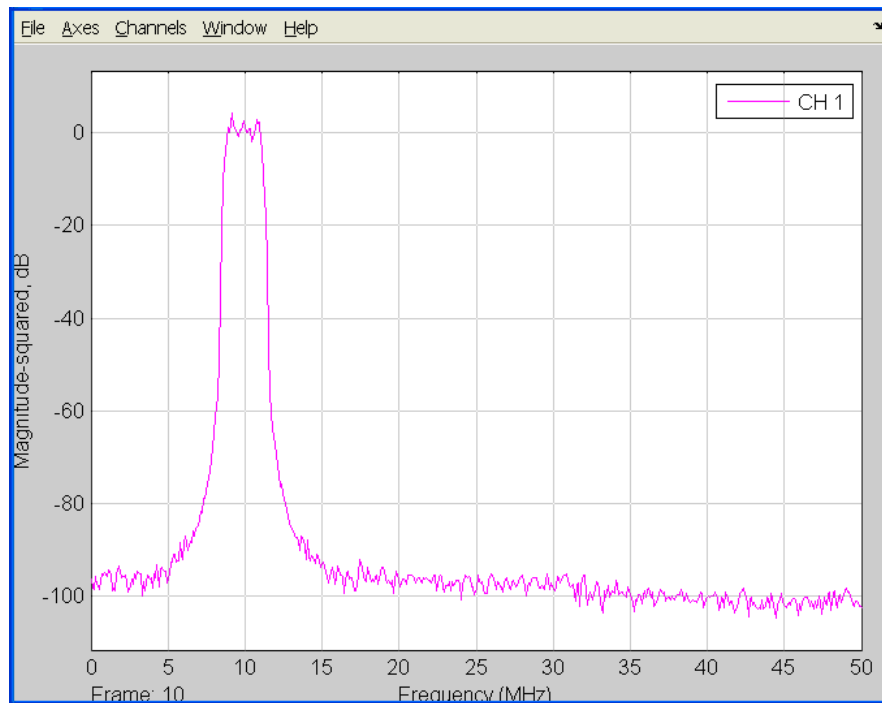
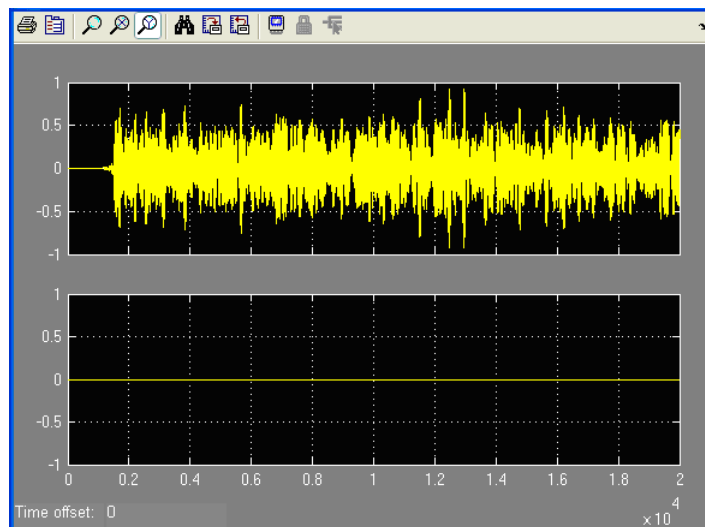



Figure A-7. Output Scope Waveform



 You can speed up the HIL simulation using burst mode. To use burst mode, open the **Hardware in the loop** dialog box and turn on **Burst Mode**. Then repeat step 15 to download an updated programming file into the device on the DSP development board. (This action resets the memory and registers to their starting value). When you simulate the HIL model again the simulation is much faster. The `OutputSpectrum` display should be identical, but extra delays (equal to the burst length) can be observed on signals in the `OutScope` display.

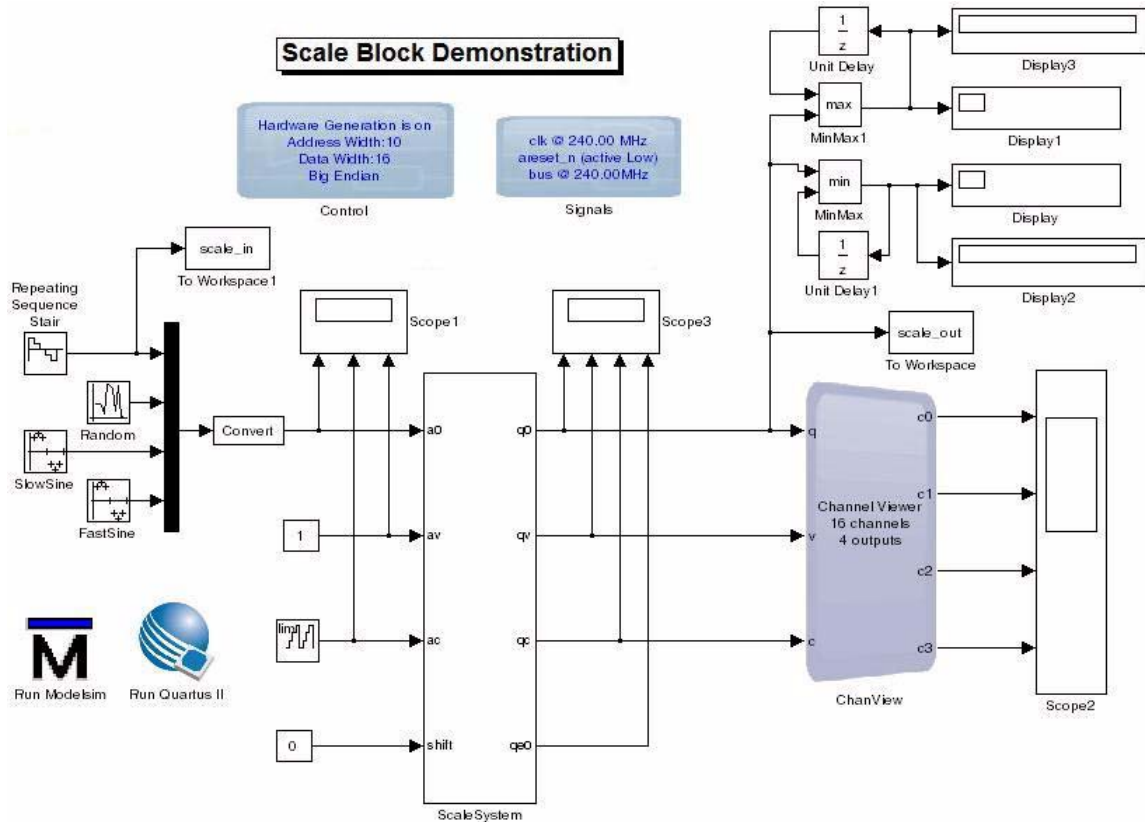
Advanced Blockset Example

This example shows how you can use HIL in a design that uses blocks from the advanced blockset only.

Perform the following steps:

1. Open the **demo_scale.mdl** model from the **Examples\Basesblocks** directory in the installed design examples for the advanced blockset (Figure A-8).

Figure A-8. Scale Block Design Example

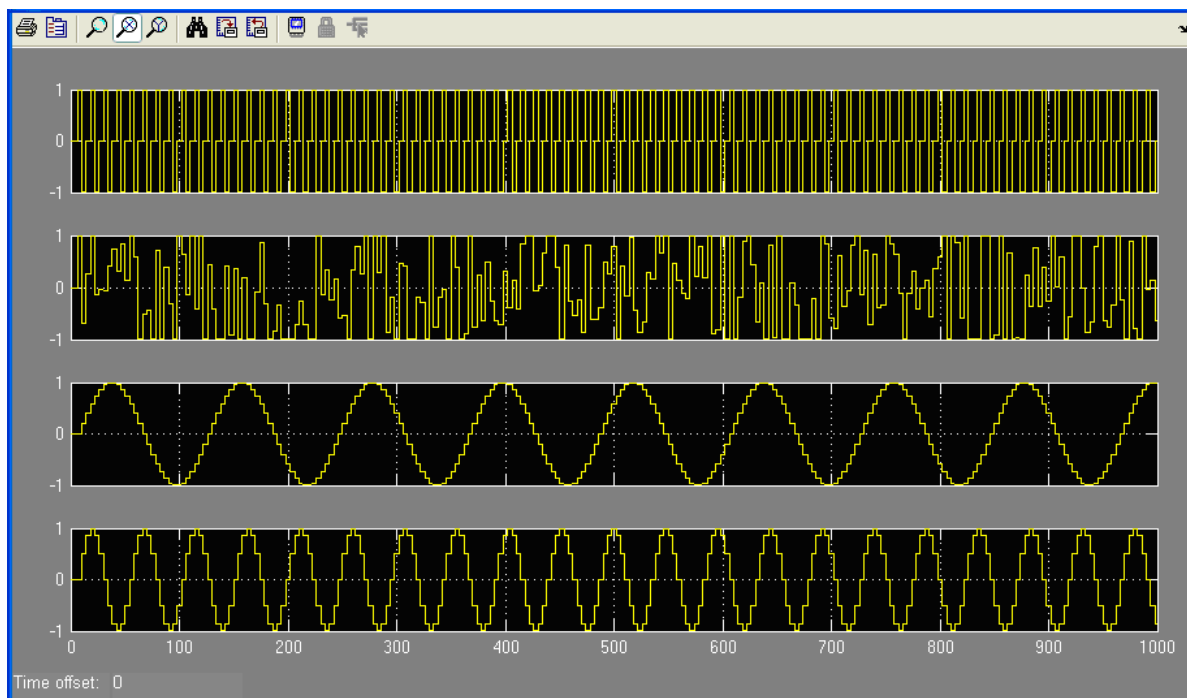


2. Simulate the design to generate HDL.


Ensure that the **Hardware Destination Directory** specified in the **Control** block uses an absolute, not relative, path.

This design demonstrate the use of the **Scale** block from the advanced blockset to scale four-channel data from signed fractional [2].[30] to signed fractional [1].[15]. The four inputs are repeating square stairs of [1 0.5 -0.5 -1 0], randomly generated numbers, a slow sine wave, and a fast sine wave, as shown in the [Figure A-9 on page A-7](#).

Figure A-9. Input Waveforms for the Scaler Example



3. Open the `Device` block inside the `ScaleSystem` subsystem to verify that the device family specified is compatible with the DSP development board to be used with the HIL simulation.
4. Double-click on the `Run Quartus II` block to start the Quartus II software and create a Quartus II project with the HDL generated in step 2.

 The Quartus II project obtains its name from the subsystem which contains the `Device` block and is named `ScaleSystem.qpf`.

5. In the Quartus II Tools menu, click **Start Compilation** and verify that the project compiles successfully.
6. Save a copy of the model as `demo_scale_HIL.mdl` and delete the `ScaleSystem` subsystem in this model.
7. Replace the `ScaleSystem` subsystem by a HIL block from the `AltLab` library in the DSP Builder standard blockset.
8. Double-click on the HIL block to open the **Hardware in the loop** dialog box (Figure A-10 on page A-8). In the first page of the dialog box, select the Quartus II project (`ScaleSystem.qpf`) that was created in step 4. Select the clock pin (`clk`) and reset pin (`areset_n`). Set the `a0`, `a1`, `a2`, and `a3` input port types as **signed [2].[30]**, the `q0`, `q1`, `q2`, and `q3` output port types as **signed [1].[15]**.


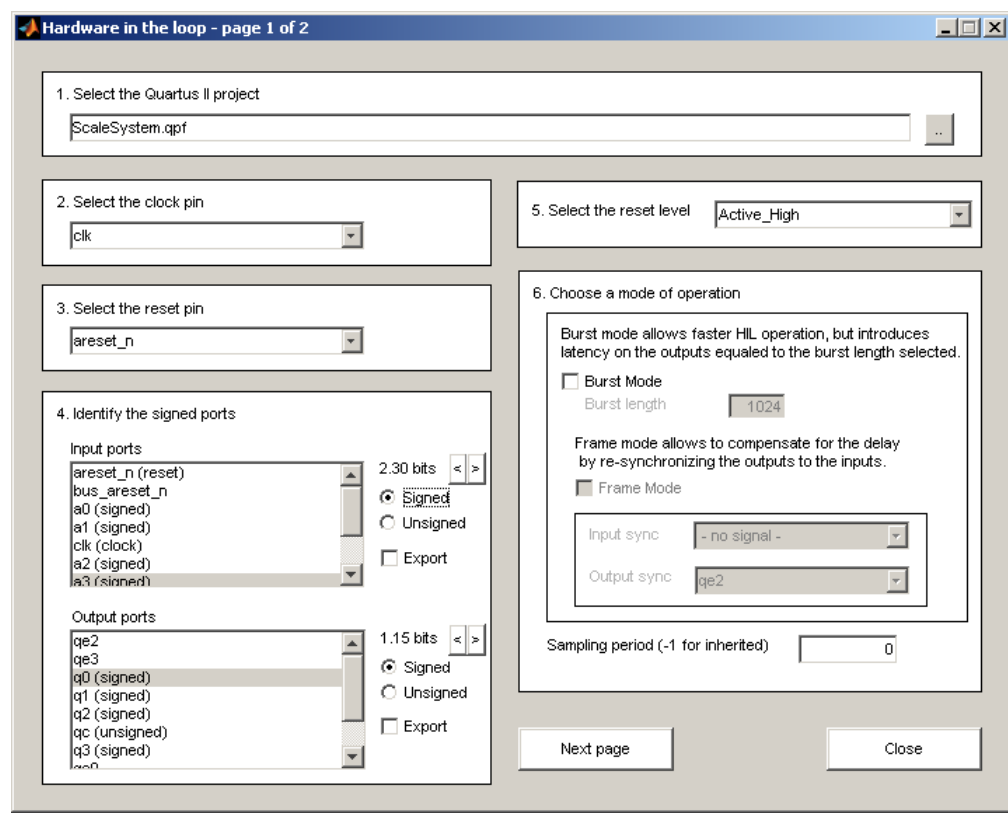

 Leave the `ac` and `shift` input ports and the `qc` output port as unsigned. The reset level must match the level specified in the `Signals` block for the original model.

Figure A-10. Hardware in the Loop Parameter Settings Page 1 for the Scale Block Example



9. Close the **Hardware in the loop** dialog box.
10. Remove the Scope1, Scope3, Unit Delay, Unit Delay1, MinMax, MinMax1, To Workspace, Display, Display1, Display2, and Display3 blocks from the model.
11. Connect all the inputs and outputs to the **HIL** block using Input and Output blocks from the standard blockset. Verify that the data types for all the inputs and outputs are set correctly, matching those set in the **Hardware in the loop** dialog box by step 8.

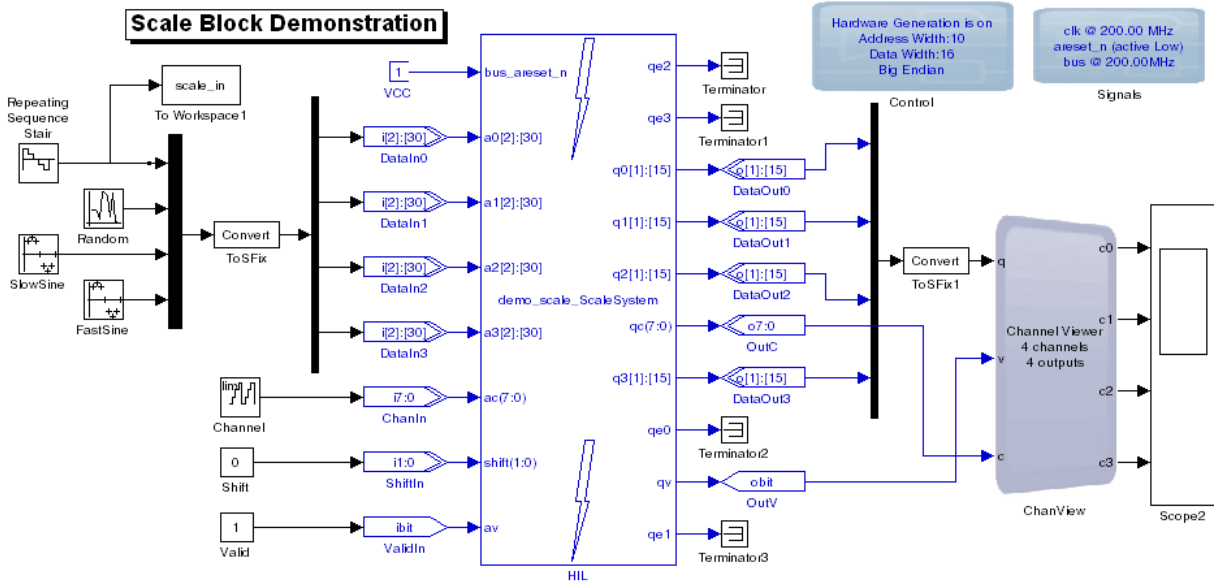
 The HIL block expands the channel data input bus into four individual inputs (a0, a1, a2, and a3) and the channel data output bus into four separate outputs (q0, q1, q2, and q3).

Use Simulink Demux and Mux blocks to separate the inputs into individual inputs and multiplex the four outputs together into one data bus.

Connect a VCC block from the standard blockset **IO & Bus** library to the signal input bus_areset_n. Terminate the output signals qe0, qe1, qe2, and qe3 using Simulink Terminator blocks.

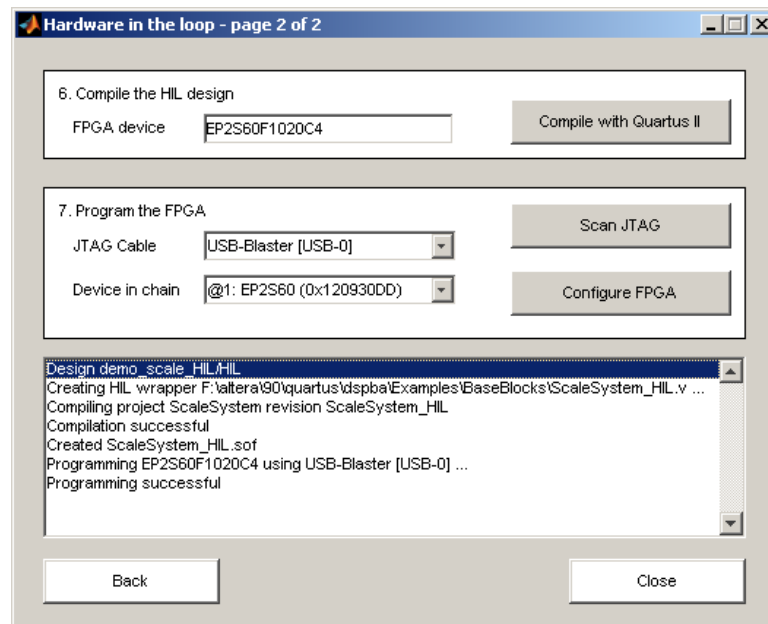
Figure A-3 on page A-3 shows the updated model.

Figure A-11. HIL Version of the Scale Block Example Design



12. Connect the DSP development board and ensure that it is switched on.
13. Re-open the **Hardware in the loop** dialog box and set the reset level as **Active_High**.
14. Click on **Next** to display the second page of the **Hardware in the loop** dialog box (Figure A-12). Enter a full device name into the **FPGA device** field. Verify that the device name matches the device on the DSP development board and is compatible with the device family set in the original model.

Figure A-12. Hardware in the Loop Parameter Settings Page 2



15. Click **Compile with Quartus II** to compile the HIL model.
16. Click **Scan JTAG** to find all the hardware connected to your computer and select the required **JTAG Cable** and **Device in chain**.
17. Click **Configure FPGA** to download the compiled programming file (**.sof**) to the DSP development board.
18. Close the **Hardware in the loop** dialog box and save the model.
19. Simulate the HIL model. Compare the waveform of the `DataOutScope` block to the results for the original model. They should be identical.

Revision History

The following table displays the revision history for this user guide.

Date	Version	Changes Made
April 2009	3.1	Added cross-reference to SOPC Builder and corrected the reset level required for a combined blockset design.
March 2009	3.0	Updated examples for the combined blockset and using Hardware in the Loop.
November 2008	2.0	Restructured as an overview for DSP design using new style and moved install information to new <i>DSP Builder Installation and Licensing</i> manual.
May 2008	1.0	First release of this user guide as an introduction to the DSP Builder Advanced Blockset.

How to Contact Altera

For the most up-to-date information about Altera products, refer to the following table.

Contact <small>Note 1</small>	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Non-technical support (General) (Software Licensing)	Email	nacomp@altera.com
	Email	authorization@altera.com







Note to Table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

This document uses the typographic conventions shown in the following table.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicates command names, dialog box titles, dialog box options, and other GUI labels. For example, Save As dialog box.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, and software utility names. For example, \qdesigns directory, d: drive, and chiptrip.gdf file.
<i>Italic Type with Initial Capital Letters</i>	Indicates document titles. For example, <i>AN 519: Stratix IV Design Guidelines</i> .

Visual Cue	Meaning
<i>Italic type</i>	Indicates variables. For example, $n + 1$. Variable names are enclosed in angle brackets (< >). For example, <file name> and <project name>. .pcf file.
Initial Capital Letters	Indicates keyboard keys and menu names. For example, Delete key and the Options menu.
“Subheading Title”	Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, “Typographic Conventions.”
Courier type	Indicates signal, port, register, bit, block, and primitive names. For example, data1, tdi, and input. Active-low signals are denoted by suffix n. Example: resetn. Indicates command line commands and anything that must be typed exactly as it appears. For example, c:\qdesigns\tutorial\chiptrip.gdf. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword SUBDESIGN), and logic function names (for example, TRI).
1., 2., 3., and a., b., c., and so on.	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	A warning calls attention to a condition or possible situation that can cause you injury.
	The angled arrow instructs you to press the Enter key.
	The feet direct you to more information about a particular topic.