



DSP Builder Handbook Volume 1:

Introduction to DSP Builder



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

HB_DSPB_INTRO-2.2

Document Version:
Document Date:

2.2
July 2011

Copyright © 2010 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. Introducing DSP Design

DSP Systems in FPGAs	1-1
FPGA Architecture Features	1-1
Software Design Flow with DSP Processors	1-3
DSP Design Flow in FPGAs	1-3
Software Flow in FPGAs	1-4
Software with Hardware Acceleration Flow	1-5
Hardware Design Flow	1-5

Chapter 2. Introducing DSP Builder

Standard and Advanced Blocksets	2-1
Standard Blockset	2-2
Advanced Blockset	2-3
Tool Integration	2-3
Simulink	2-3
ModelSim	2-4
Quartus II Software	2-5
Qsys	2-5

Chapter 3. Installing DSP Builder

System Requirements	3-1
Obtaining and Installing DSP Builder	3-1
Using Previous Versions of DSP Builder	3-3
Previous Versions	3-4
DSP Builder Start Up Dependencies	3-4
MATLAB Procedures	3-4
Directory Path Names in MATLAB Scripts	3-4
Using Multiple Versions of MATLAB	3-4
Standard Blockset	3-4
Advanced Blockset	3-5
Licensing DSP Builder	3-5
Appending the License to Your license.dat File	3-5
Specifying the License File Location	3-6

Chapter 4. Updating From Earlier Versions

Updating Models From the Previous Version	4-1
Limitations of the Update Model Utility	4-1
Using a Simulink Library Forwarding Table	4-3
Updating the IP Cores in your Design	4-3

Additional Information

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

This chapter introduces using DSP Builder for digital signal processing (DSP) designs on Altera® FPGAs. It introduces the DSP Builder standard and advanced blocksets, and the Altera-provided DSP IP libraries.

DSP Systems in FPGAs

The DSP market includes the following rapidly evolving applications, which cover a broad spectrum of performance and cost requirements:

- 3G wireless
- Voice over Internet protocol (VoIP)
- Multimedia systems
- Radar and satellite systems
- Medical systems
- Image-processing applications
- Consumer electronics.

Specialized DSP processors can implement many of these applications. Although these DSP processors are programmable through software, their hardware architecture is not flexible. Therefore, 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 limit DSP processors. The DSP processor's fixed hardware architecture is not suitable for some applications that require customized DSP function implementations.

FPGAs provide a reconfigurable solution for implementing DSP applications, higher DSP throughput, and more raw data processing power than DSP processors. You can reconfigure FPGAs in hardware, therefore they offer complete hardware customization while implementing various DSP applications. You can customize the architecture, bus structure, memory, hardware accelerator blocks, and the number of MAC blocks in an FPGA system.

FPGA Architecture Features

You can configure FPGAs to operate in different modes corresponding to a required functionality. This hardware flexibility allows you to use a suitable hardware description language (HDL) such as VHDL or Verilog HDL to implement any hardware design. 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 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 and subtraction, and summation, which are common arithmetic operations in DSP functions. Altera FPGAs offer much more multiplier bandwidth than DSP processors, which only offer a limited number of multipliers.

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.

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 some cases.

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 multimaster 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. DSP processors have predefined hardware accelerator blocks, but FPGAs can implement hardware accelerators for each application, allowing the best achievable performance from hardware acceleration. You can implement hardware accelerator blocks with parameterizable IP functions or from scratch using HDL.



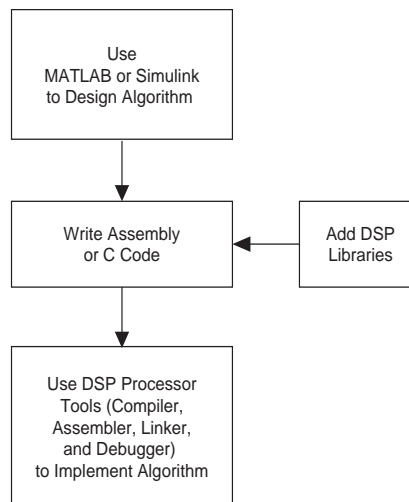
Altera offers many IP cores for DSP design, for more information about these IP cores, refer to [Chapter 2, Introducing DSP Builder](#).

You can parameterize Altera DSP IP cores for the most efficient hardware implementation and to provide maximum flexibility. 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 1-1 shows the typical software design flow that DSP programmers follow.

Figure 1-1. Software-Based DSP Design Flow



You can use algorithm development tools such as MATLAB 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 with 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 tools such as DSP Builder, Qsys, and the Nios II embedded design suite (EDS) enable you to follow a software-based design flow while targeting FPGAs.

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. Altera's DSP Builder tool provides an interface from Simulink directly to the FPGA hardware (Figure 1-2). Additionally, you can incorporate the designs created by DSP Builder into a Qsys system for a complete DSP system implementation

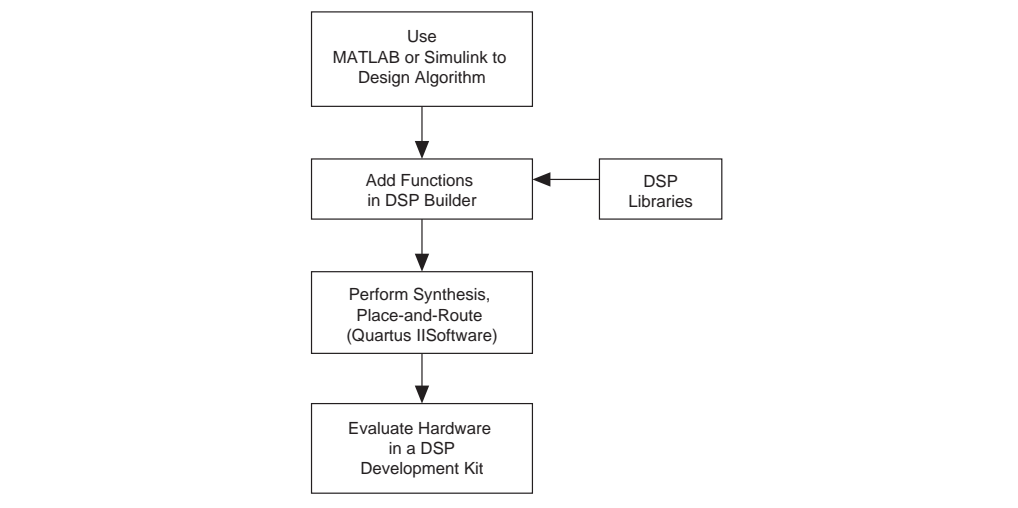
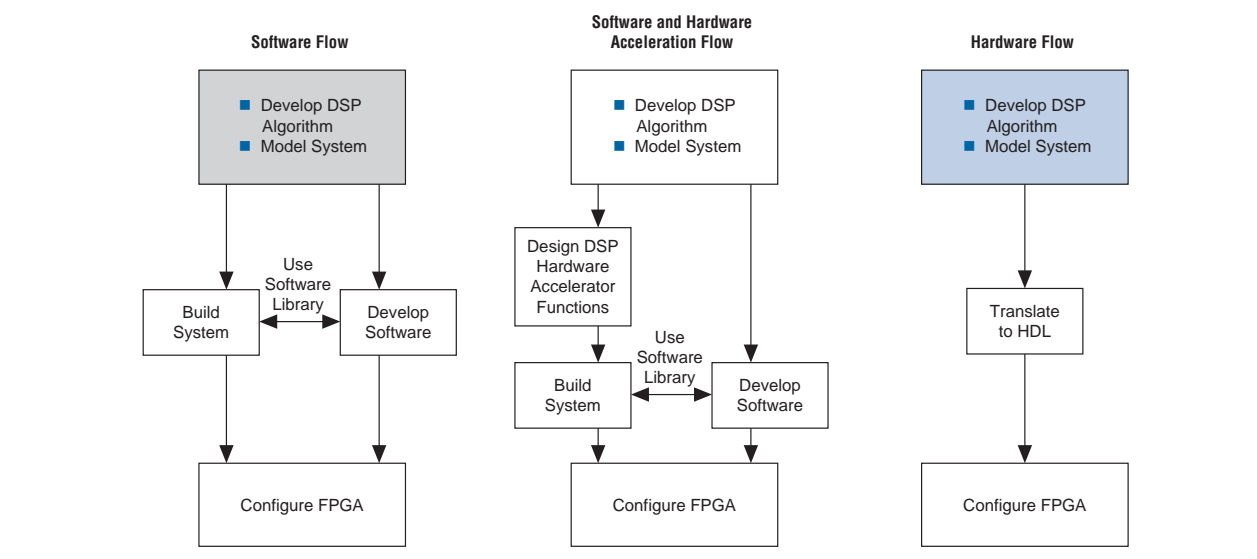
Figure 1–2. DSP Builder General Design Flow for Altera FPGAs

Figure 1–3 shows 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 the Nios II EDS development tools for compiling, debugging, assembling, and linking software designs. You can then download these software designs to an FPGA using either on-chip RAM or an external memory device.

Software with Hardware Acceleration Flow

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 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.

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

Altera also provides system integration tools such as Qsys for system-level partitioning and interconnection. You can use Qsys to build entire hardware systems by combining the embedded processor, such as a Nios II embedded processor, with other system peripherals and IP cores.

Hardware Design Flow

You can use an HDL-based hardware design flow to develop a pure hardware implementation of a DSP system. 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 IP cores can simplify this development process. You can also follow the DSP Builder design flow ([Figure 1-2](#)) and implement hardware-only DSP systems in FPGAs without learning HDL.

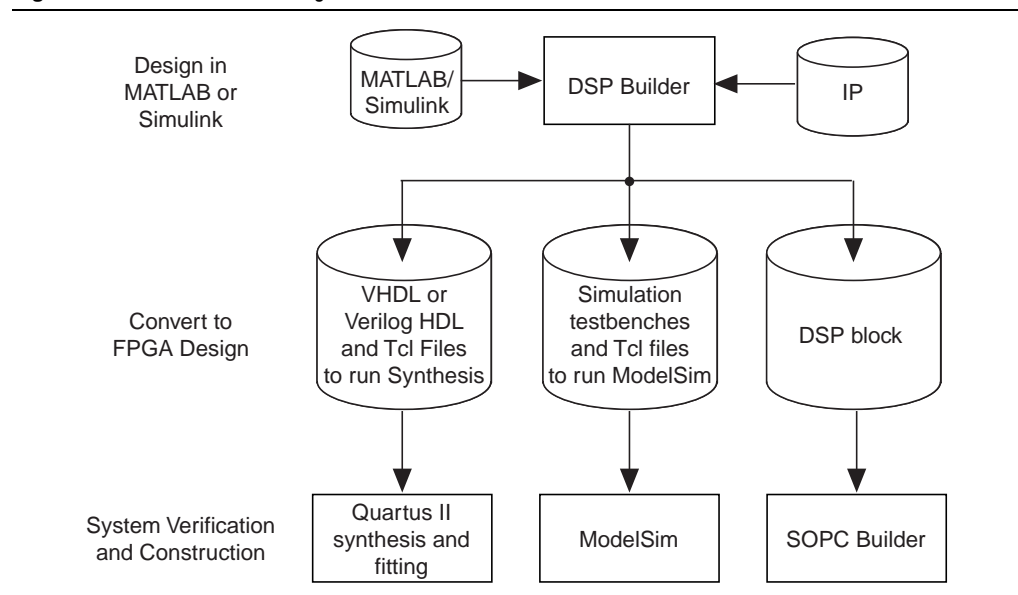


For information about the DSP Builder design flow, refer to [Chapter 2, Introducing DSP Builder](#).

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 2-1). DSP Builder supports all Altera DSP IP cores.

Figure 2-1. DSP Builder Design Flow



Standard and Advanced Blocksets

When you install DSP Builder two separate blocksets (standard and advanced) are added to the Simulink library browser, which you can use separately or together.

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

The standard blockset has the following features:

- Cycle-accurate behavioral models
- Multiple clock domain management
- Control rich with backpressure support
- Access to specific hardware device features
- Hardware-in-the-loop (HIL) support enables FPGA hardware cosimulation
- Support for importing VHDL or Verilog HDL design entities

- Tabular and graphical state machine support
- Rapid prototyping using Altera DSP development boards
- SignalTap® II logic analyzer debugging support
- Direct instantiation of DSP IP cores

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

The advanced blockset has the following features:

- 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
- Simulink fixed-point types
- Single system clock for the main datapath logic
- Feed-forward datapath 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-loop (HIL) 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, and embedded memory.

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

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



For more information about the standard blockset, refer to *Volume 2: DSP Builder Standard Blockset* in the *DSP Builder Handbook*.

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 untimed netlist into low-level, pipelined hardware for the target FPGA device and clock rate. DSP Builder implements the hardware as VHDL with scripts that integrate with the Quartus II software and the ModelSim simulator.

The combination of these features allows you to create a design without intimate device knowledge, which can run on a variety of FPGA families with different hardware architectures.

After specifying the 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, DSP Builder also optimizes folding (time-division multiplexed (TDM) designs) 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, you can use a parameter file within the Simulink design. DSP Builder then adds the required TDM control logic and generates the updated RTL.

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

- 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. Altera provides several implementation examples including up and down converters.
- 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 DSP Builder decides 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 *Volume 3: DSP Builder Advanced Blockset* in the *DSP Builder Handbook*.

Tool Integration

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

Simulink

DSP Builder is interoperable with other Simulink blocksets. In particular, use the basic Simulink blockset to create interactive testbenches.

The DSP Builder standard blockset uses signed integer, unsigned integer or signed fractional signal types and 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 *Volume 2: DSP Builder Standard Blockset* in the *DSP Builder Handbook*.

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

- For information about Simulink fixed point types, the signal processing blockset and the communications blockset, refer to the MATLAB Help.

A VHDL model generates for subsystems with the advanced blockset when you run a Simulink simulation.

There are many examples of using Simulink blocks in the tutorial and design examples. In particular, use Simulink scopes in the advanced blockset example to identify signals to add to the ModelSim **Wave** window for display as a digital or analog signal.

- For information about the tutorials and design examples, refer to *Design Examples* in *DSP Builder Standard Blockset Libraires* section in volume 2 of the *DSP Builder Handbook* and the *DSP Builder Advanced Blockset Libraires* section in volume 3 of the *DSP Builder Handbook*.

ModelSim

You can run the ModelSim simulator from within a DSP Builder standard or advanced blockset design, if the ModelSim executable (**vsim.exe**) is in your path.

Use the TestBench block to integrate between the DSP Builder standard blockset and the ModelSim simulator. When you click **Compile against HDL**, a VHDL testbench generates and a **tb_<model name>.tcl** script generates that you can use to load the testbench into the ModelSim simulator. 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 Standard Blockset Libraires* section in volume 2 of the *DSP Builder Handbook*.

You can use any of the following generated scripts to integrate between the DSP Builder advanced blockset and the ModelSim simulator:

- **Control.do.** This script, named after the control block (normally `Control`) compiles the entire design with RTL equivalents of the testbench structures, adds all relevant signals to the Wave window, and runs 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 adds the signals to the Wave window, before the simulation is started. Use these easy-to-follow scripts when building custom flows. To automatically load the design in the ModelSim simulator, click on the `Run ModelSim` block in the top-level model.



Only a subset of the Simulink blocks are translated into RTL that you can use for simulation in the ModelSim simulator. For a list of compatible blocks, refer to the *Run ModelSim* block description in the *DSP Builder Advanced Blockset Libraires* section in volume 3 of the *DSP Builder Handbook*.

- `<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 Libraires* section in volume 3 of the *DSP Builder Handbook*.

Quartus II Software

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 allows you to build high-speed, high-performance DSP datapaths. In most production designs there is an RTL layer surrounding this datapath to perform interfacing to processors, high speed I/O, memories, and so on.

To complete the design, use the DSP Builder standard blockset, Qsys, or RTL to assign board level components. 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.

Qsys

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 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 Qsys.

-  For an example of the standard blockset integration with Qsys, refer to the *Using the Interface Library* chapter in the *DSP Builder Standard Blockset User Guide* section in volume 2 of the *DSP Builder Handbook*.

DSP Builder creates a memory-mapped interface and **class.ptf** file for each advanced blockset design. This file can expose the processor bus for connection in Qsys. A DSP Builder advanced blockset subsystem is available from the **System Contents** tab in Qsys after you add the path to the **class.ptf** file to the Qsys IP search path.

-  For an example of the advanced blockset integration with Qsys, refer to the *DSP Builder Advanced Blockset User Guide* section in volume 3 of the *DSP Builder Handbook*.

This chapter describes how to install DSP Builder.

System Requirements

As DSP Builder integrates with the The MathWorks MATLAB and Simulink tools and with the Altera Quartus® II software, ensure both tools are available on your workstation before you install DSP Builder.



For system requirements and installation instructions, refer to *Altera Software Installation and Licensing*.



You should use the same version of the Quartus II software and DSP Builder.

Table 3–1 lists the tool dependencies for DSP Builder.

Table 3–1. DSP Builder Tool Dependencies

Tool	Software Version		
	11.0	10.1	10.0
DSP Builder			
The MathWorks (MATLAB and Simulink) (1), (2), (3)	R2009b R2010a R2010b R2011a	R2009a R2009b R2010a	R2008a R2008b R2009a

Notes to Table 3–1:

- (1) DSP Builder does not work with MATLAB in read-only mode. If your PC issues error messages while creating board components during the DSP Builder installation, reinstall MATLAB with the READ ONLY option unchecked.
- (2) The DSP Builder advanced blockset uses Simulink fixed-point types for all operations and requires licensed versions of Simulink Fixed Point. Altera also recommends the Simulink Signal Processing Blockset and Communications blocksets, which the design examples use.
- (3) DSP Builder advanced blockset supports 64-bit MATLAB; DSP Builder standard blockset supports 64-bit MATLAB on Linux OS only.

Obtaining and Installing DSP Builder

You install DSP Builder from the Altera Complete Design Suite DVD. In the Altera software installer, ensure you turn on DSP Builder on the **Select components** screen (Figure 3–1).

Figure 3-1. Select Components—DSP Builder

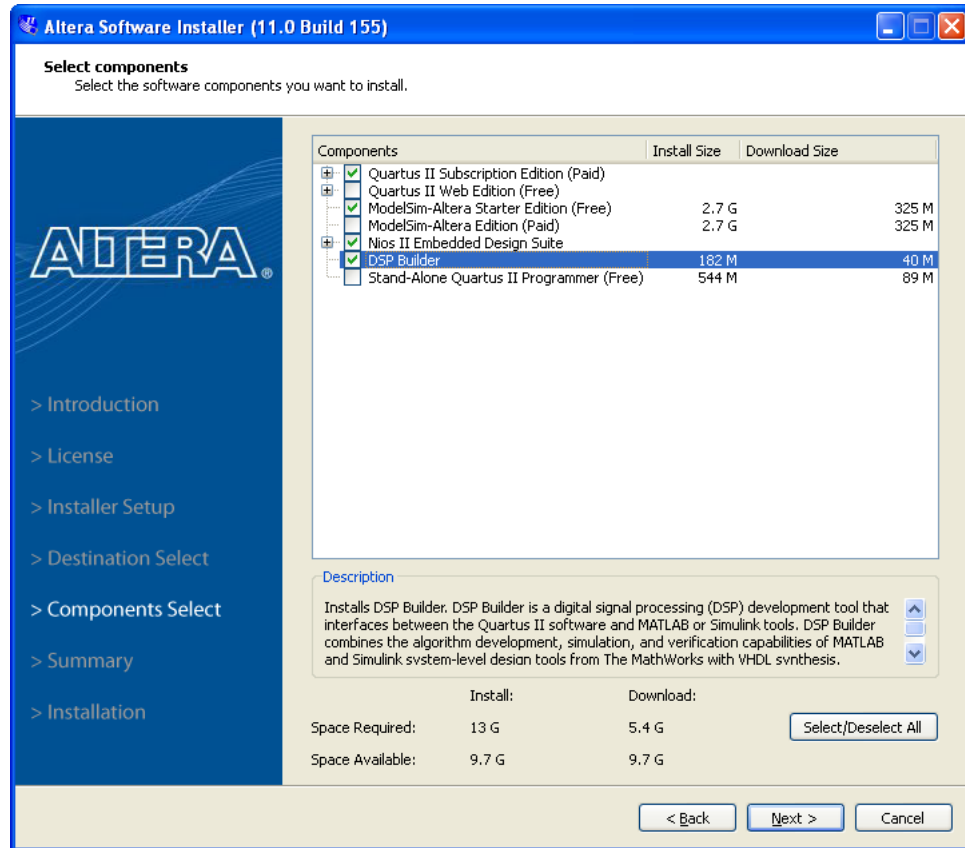
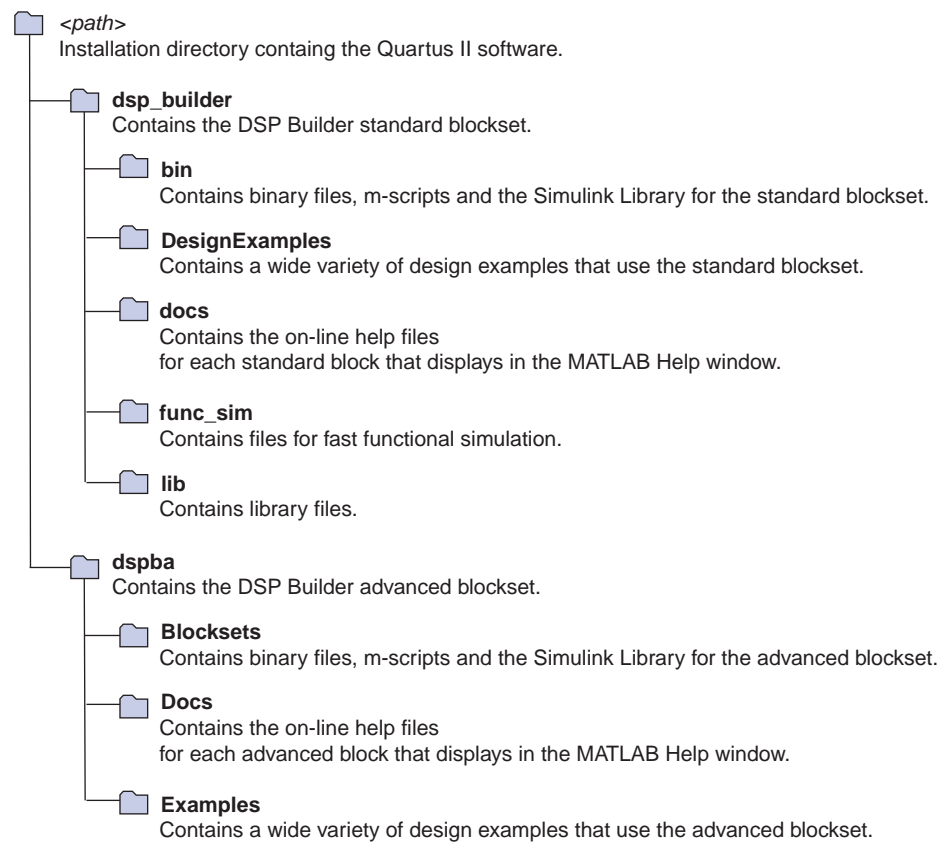


Figure 3-2 shows the DSP Builder directory structure, where *<path>* is the installation directory that contains the Quartus II software. The default installation directory is `c:\altera\<version>\quartus` on Windows or `/opt/altera<version>/quartus` on Linux.

Figure 3-2. DSP Builder Directory Structure




After installing DSP Builder, the Altera DSP Builder standard blockset and the Altera DSP Builder advanced blockset libraries, which you specified, are available in the Simulink library browser in the MATLAB software.

To start DSP Builder, follow one of these steps:

1. On Windows, start MATLAB, click the Simulink icon.
2. On Linux, use the following command, which automatically finds MATLAB.


```
run dsp_builder.sh
```

 You can use the following options after the `dsp_builder.sh` command:

- `-m <path to MATLAB>` to specify another MATLAB path
- `-glnx86` to run 32-bit DSP Builder

Using Previous Versions of DSP Builder

The DSP Builder installer searches for an existing installation of DSP Builder before it installs a new version. If it finds an existing installation, the installer prompts you to update the existing installation or install a new copy of the product. If you choose to update an existing product, you can modify, repair, or remove the old version.

 Do not use the current installer to modify or repair a previous version of DSP Builder.

Previous Versions

A previous version of DSP Builder cannot coexist with a current version in the same version of MATLAB. However, you can register each version of DSP Builder with different versions of MATLAB. Refer to [“Using Multiple Versions of MATLAB” on page 3-4](#).



Use DSP Builder with a matching version of the Quartus II software.

DSP Builder Start Up Dependencies

DSP Builder uses the Quartus II libraries to share functionality that exists in the Quartus II software, which places explicit dependencies on the Quartus II versions.

DSP Builder is Simulink dependent. After installing DSP Builder, you need to register it inside MATLAB to enable the DSP Builder features. You can then create DSP designs using DSP Builder blocks and run Simulink simulations without any requirements on the Quartus II software.

However, when you want to generate VHDL for the DSP design and to fit the design into an FPGA, DSP Builder requires the Quartus II synthesis, and Fitter tools.

You can only start the DSP Builder `Signal Compiler` tool with a matching version of the Quartus II software—it explicitly requires the correct version libraries and dynamic link library (.dll) files in the Quartus II software installation. The **Signal Compiler** dialog box does not display without a matching version of the Quartus II software.

MATLAB Procedures

[Table 3-1 on page 3-1](#) shows the which versions of MATLAB you can use with DSP Builder .

Directory Path Names in MATLAB Scripts

Commands run from within an m-script cannot resolve a uniform naming convention (UNC) path to a remote file system. An error message appears when you attempt to run a MATLAB script that uses a UNC path.

This error affects the HDL import, functional simulation, hardware-in-the-loop (HIL), and the `State Machine Table` block.

Avoid this issue by mapping the network UNC path to a local drive.

Using Multiple Versions of MATLAB

You specify the MATLAB version that you want to use with DSP Builder during DSP Builder installation. If you have more than one MATLAB version (for example, release R2007b and R2008a) you can register DSP Builder with another version using the following procedure:

Standard Blockset

1. Open a command prompt and change directory to the standard blockset installation:

```
cd <DSP Builder Installation Path>dsp_builder
```

2. Run the following command to register the DSP Builder blocksets with the required MATLAB version:

```
setupMatlabClassPath install <MATLAB Installation Path> <DSP Builder Installation Path>\dsp_builder
```



You must use quotation marks around the DSP Builder or MATLAB installation path, if the paths include spaces.

Advanced Blockset

1. Open MATLAB and change directory to the advanced blockset installation:

```
cd <DSP Builder Installation Path>dspba
```

2. Run the following command in the MATLAB command window to register the DSP Builder advanced blockset with the current MATLAB version:

```
alt_adv_dspb_install
```

Licensing DSP Builder

Before using DSP Builder, you must request a license file from the Altera website at www.altera.com/licensing and install it on your computer. When you request a license file, Altera e-mails you a **license.dat** file that enables HDL file and Tcl script generation.

If you do not have a DSP Builder license file, you can create models with DSP Builder blocks but you cannot generate HDL files or Tcl scripts.



Before you set up licensing for DSP Builder, you must already have the Quartus II software installed on your computer with licensing set up.

To install your license, you can either append the license to your existing **license.dat** file or you can specify an additional license file location.

Appending the License to Your license.dat File

To install your license, perform the following steps:

1. Close the following software if it is running on your computer:
 - The Quartus II software
 - The LeonardoSpectrum software
 - The Synplify software
 - The MATLAB and Simulink tools
 - The ModelSim simulator
 - The Precision RTL synthesis software
2. Open the DSP Builder license file in a text editor. The file contains one **FEATURE** line, spanning two lines.
3. Open your Quartus II **license.dat** file in a text editor.

4. Copy the `FEATURE` line from the DSP Builder license file and paste it into the Quartus II license file.



Do not delete any `FEATURE` lines from the Quartus II license file.

5. Save the Quartus II license file.



When using editors such as Microsoft Word or Notepad, ensure that the file does not have any extra extensions appended to it after you save (for example, `license.dat.txt` or `license.dat.doc`). Verify the file name at the system command prompt.

Specifying the License File Location

DSP Builder always looks for a license in the `LM_LICENSE_FILE` environment variable, you can modify this variable to specify an additional location for the DSP Builder license file. To ensure the Quartus II software uses the `LM_LICENSE_FILE` variable follow these steps:

1. On the Tools menu click License Setup. The License Setup dialog box appears.
2. Turn on Use `LM_LICENSE_FILE`.

Updating Models From the Previous Version

To update your designs from the previous version, open them with the latest version and save the model files.

This procedure does not automatically update any MegaCore function blocks in your design but the previous versions are compatible if the previous version of the MegaCore IP library is still installed on your system.



If you have a pre-v7.1 design, update to v7.2 before you update the v7.2 design to v8.x or v9.0 or later.

Limitations of the Update Model Utility

The blocks in your model are updated to use the corresponding block in the new blockset. However some blocks may be obsolete or require manual intervention to complete the conversion process.

Table 4–1 lists some of the issues that may require attention.

Table 4–1. Model Update Issues (Part 1 of 2)

Issue	Action
PLL output clocks cannot be named	In previous versions of DSP Builder, it was possible to have a PLL block and multiple <code>ClockAltr</code> blocks which represented PLL outputs. The PLL output clocks took the names of the clock blocks. This feature is no longer supported and cannot be automatically fixed. The PLL output clocks are now named <code><PLL name>_clk<output index></code> . All source blocks and rate change blocks referencing clock pins must be manually edited to reference these PLL clock output pins.
PLL period multiply and divide values must be integers	In pre-7.1 versions, the multiply and divide values may have non-integer values and may be specified using MATLAB variables. You must now specify the clock period ratio directly as an integer period multiplier and an integer period divider. MATLAB variables cannot be used.
The PLL output clock period is incorrect after update	Occasionally, the PLL parameters are updated incorrectly. Open the PLL parameter dialog and enter the correct values for the period multipliers and dividers.
When upgrading a design with a PLL, extra clock blocks are created for each distinct sample time	The extra Clock and Clock_Derived blocks should be removed, and any blocks referencing them manually corrected to reference the PLL-driven clocks. Note that the numbering of these clock pins does not in general match the numbering of the PLL clocks.
The PLL input clock frequency information is lost during the update process	Typically, you may want to create a new Clock block replicating this information, as the base clock pin generated by the update script is unlikely to be the correct driving clock domain. For example, if the PLL specified an input clock frequency of 50 MHz, add a Clock block and configure it to a clock period of 20ns and sample time 20e-9.
Clock blocks do not support rate change divider	In pre-7.1 versions, the ClockAltr blocks supported a rate change option (addition clock divider), which can generate a slower clock signal. This feature is no longer supported. If you want to generate different frequency clocks internally, you must add a PLL block driven from the required input clock.

Table 4-1. Model Update Issues (Part 2 of 2)

Issue	Action
Error assigning clock for Dual-Clock FIFO block	Under some circumstances - noted by the message "No clock specified for {write/read} port, ..." you may have to manually select clocks after upgrading designs containing the Dual-Clock FIFO block.
Error assigning PLL clock for Multi-Rate DFF block	When upgrading a multirate DFF block connected to a PLL clock, an error is issued of the form: "Cannot update clock in block foo/Multi-Rate DFF. Original clock source: PLL CLOCK0." The blocks must be manually corrected to reference the PLL clock.
Unnecessary clock specification for source blocks	In general, source blocks do not need to specify a clock domain, if it can be inferred from the blocks they are driving. However, the update path always specifies a clock if it is not the base clock. Your multi-clock design may be easier to maintain if, after upgrading, you manually turn off Specify Clock for source blocks – especially constant, VCC and GND blocks – wherever possible.
Errors issued if a constant, GND or VCC block is driving a block with a different sample time	These errors can usually be fixed by turning off Specify Clock on the constant block. If the block is fed into several clock domains, you also need to add a Tsamp block before each one.
The BP block does not support sample time mode	A warning is issued if your design includes a bus probe (BP) block which was set to display the sample time because this option is no longer supported.
Phase selection has been standardized across all blocks	Results in behavioral change when upgrading blocks that use phase selection.
The Multi Channel Display and Extract blocks are not supported	These blocks are no longer supported and should be removed before running the update script. You can use the Avalon-ST Packet Format Converter block directly in place of these blocks. To prevent HDL being generated, insert Output blocks followed by Non-synthesizable Input blocks on the inputs to the Avalon-ST packet format converter block.
HIL designs must be recompiled	For designs with Hardware in the Loop, you must recreate the Quartus II project and recompile the HIL revision after upgrading.
Changes to rounding method used for the MATLAB arrays initialize the LUT and RAM blocks	The rounding method used when the data values specified by an initializing MATLAB array are not exactly expressed if the data type changes. For example, if you specify the data type as Unsigned Integer and the value as 1.9 in a pre-v7.1 design, this value was rounded up to 2; it is now rounded down to 1. You should check the outputs from LUT or RAM blocks if an error is issued stating that the values cannot be exactly represented in the selected data format and choose revised initialization values that can be represented exactly if the outputs are not as expected.
Black box subsystem are not updated	Altbus blocks used as black box inputs or outputs must be manually changed to HDL Input and HDL Output blocks and a HDL Entity block added to specify the HDL file and clock/reset ports.
An error is issued for any block name which has a '/' character	Rename any block containing a '/' character in its name.
AltBus blocks within subsystems which function as input pins not updated correctly	Move the input pins to the top level or replace them by Input blocks. (It is better to replace these AltBus blocks before upgrading to ensure that the clock signals are set correctly.)
Device Programmer block is not supported	Remove the Device Programmer block before running the update program. Use Signal Compiler or a HIL block to program the DSP development board.
The External RAM block is not updated	This block is outside the DSP Builder system and is not automatically updated. You must manually replace any pre-7.1 External RAM blocks in your designs with the latest version.

Using a Simulink Library Forwarding Table

You can use a Simulink library forwarding table to update models with changes in the names or locations of the library blocks that they reference. Using this feature, all of the links in the updated version of the old file can be quickly and easily changed to point to a different library.



For more information, refer to *Making Backward-Compatible Changes to Libraries* in the MATLAB Simulink help.

Updating the IP Cores in your Design

After you have updated designs containing IP cores, regenerate the IP blocks to ensure they are up-to-date. Either individually double-click on each block, (changing the parameters, if required) and click the **Generate** option, or enter the following command at the MATLAB prompt:

```
alt_dspbuilder_refresh_megacore
```

This command automatically regenerates all the IP cores in the updated design, using the existing parameterizations.

Revision History

The following table shows the revision history for this document.

Date	Version	Changes Made
July 2011	11.0	Updated MATLAB version support.
April 2011	11.0	<ul style="list-style-type: none"> ■ Updated MATLAB version support ■ Added support for 64-bit MATLAB ■ Updated installation instructions
December 2010	10.1	Updated MATLAB version support.
July 2010	10.0	First release.

How to Contact Altera

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

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







Note:

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

Typographic Conventions

The following table shows the typographic conventions that this document uses.

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. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, dialog box options, software utility names, and other GUI labels. 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. For example, resetn. Indicates command line commands and anything that must be typed exactly as it appears. For example, c:\qdesigns\tutorial\chiptrip.gdf. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword SUBDESIGN), and logic function names (for example, TRI).
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.
 CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
 WARNING	A warning calls attention to a condition or possible situation that can cause you injury.
	The angled arrow instructs you to press Enter.
	The feet direct you to more information about a particular topic.