DSP Builder Handbook

Volume 1: Introduction to DSP Builder

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

HB_DSPB_INTRO-5.0

Document last updated for Altera Complete Design Suite version: 14.0
Document publication date: June 2014

Feedback Subscribe

Contents

1. About DSP Design

This chapter introduces DSP Builder for implementing digital signal processing (DSP) designs on Altera® FPGAs.

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. Because you can reconfigure FPGAs, 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. You can 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 addition, subtraction, and multiplication, 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 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.

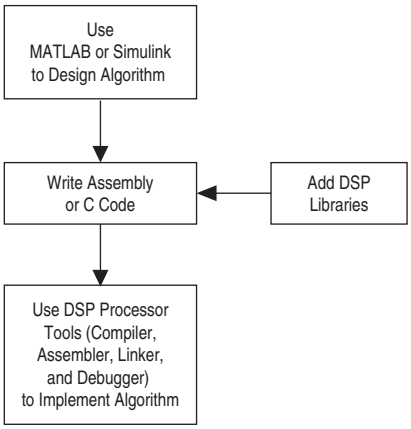
Altera offers many IP cores for DSP design, for more information about these IP cores, refer to Chapter 2, About DSP Builder.

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, system engineers use a hardware flow based on a HDL language, such as Verilog HDL or VHDL, to implement DSP systems in FPGAs. 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. DSP Builder 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

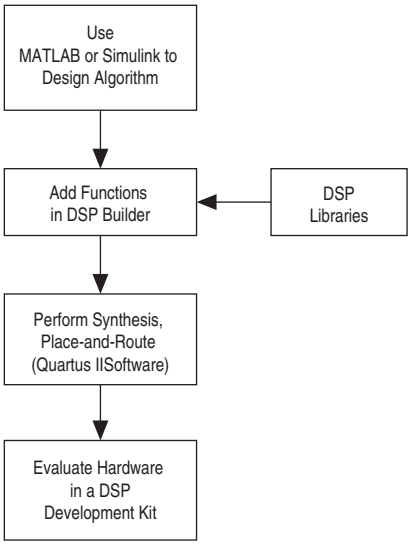
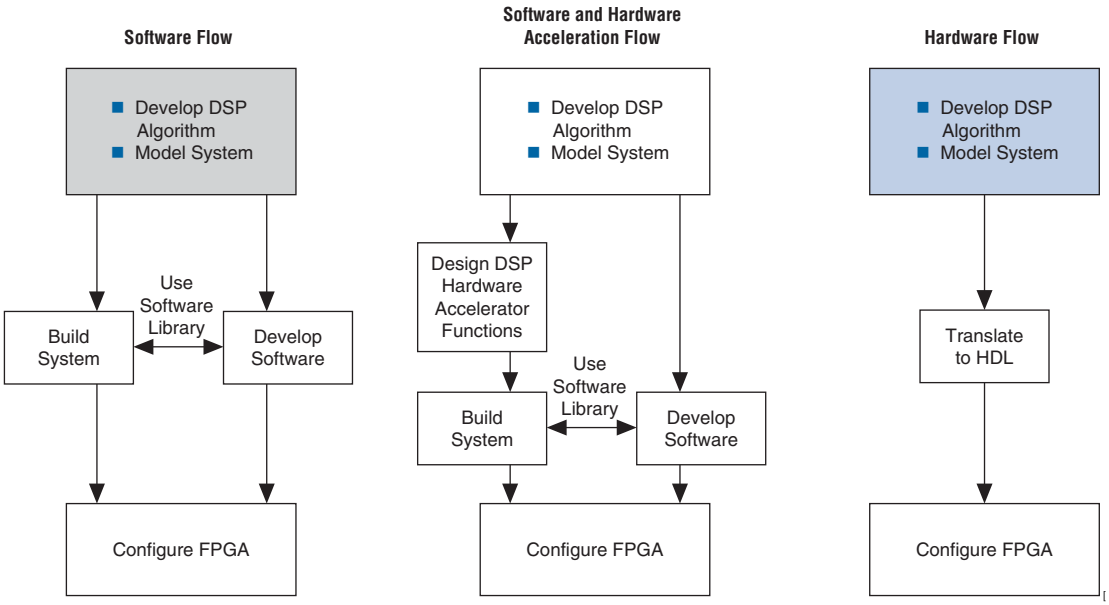


Figure 1–3 shows the various design-flow options available for FPGAs.

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 use either on-chip RAM or an external memory device to download these software designs to an FPGA.

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 critical. 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. You can also follow the DSP Builder design flow (Figure 1–2) and implement hardware-only DSP systems in FPGAs without learning HDL.

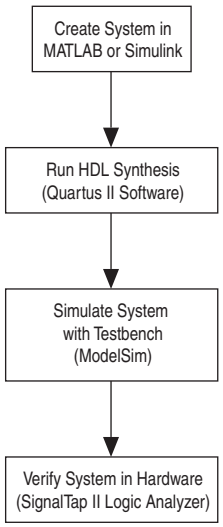
For information about the DSP Builder design flow, refer to Chapter 2, About DSP Builder.

2. About 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 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 to verify system level specifications and perform simulation. Figure 2–1 shows the DSP Builder system-level design flow.

Figure 2–1. DSP Builder System-Level Design Flow



Advanced and Standard Blocksets

The DSP Builder installer installs two separate blocksets (advanced and standard), which you can use separately or together from the Simulink library browser.

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

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

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

Tool Integration

DSP Builder works with Simulink, the ModelSim software, and the Quartus II software (including Qsys).

Simulink

DSP Builder is interoperable with other Simulink blocksets. In particular, you can use the basic Simulink blockset to create interactive testbenches. The `testbench` block allows you to generate a VHDL model, so that you can compare Simulink simulation results with the ModelSim simulator.

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

ModelSim

You can run the ModelSim simulator from within DSP Builder, if the ModelSim executable is in your path. You can use a script to integrate between the DSP Builder advanced blockset and the ModelSim simulator. The script runs the automatic testbench flow for a block. It reads 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.

Quartus II Software

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 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

DSP Builder creates a memory-mapped interface and **hw.tcl** 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 **hw.tcl** file to the Qsys IP search path.

3. Installing DSP Builder

This chapter describes how to install DSP Builder.

System Requirements

DSP Builder integrates with the The MathWorks MATLAB and Simulink tools and with the Altera Quartus® II software.

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

Ensure at least one version of The MathWorks MATLAB and Simulink tool is available on your workstation before you install DSP Builder. Table 3–1 lists the tool dependencies for DSP Builder.

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

Table 3–1. DSP Builder Tool Dependencies

Tool	Version		
DSP Builder	14.0	13.1	13.0
The MathWorks (MATLAB and Simulink) (1)	R2012a R2012b R2013a R2013b R2014a	R2012a R2012b R2013a R2013b	R2010a R2010b R2011a R2011b R2012a R2012b

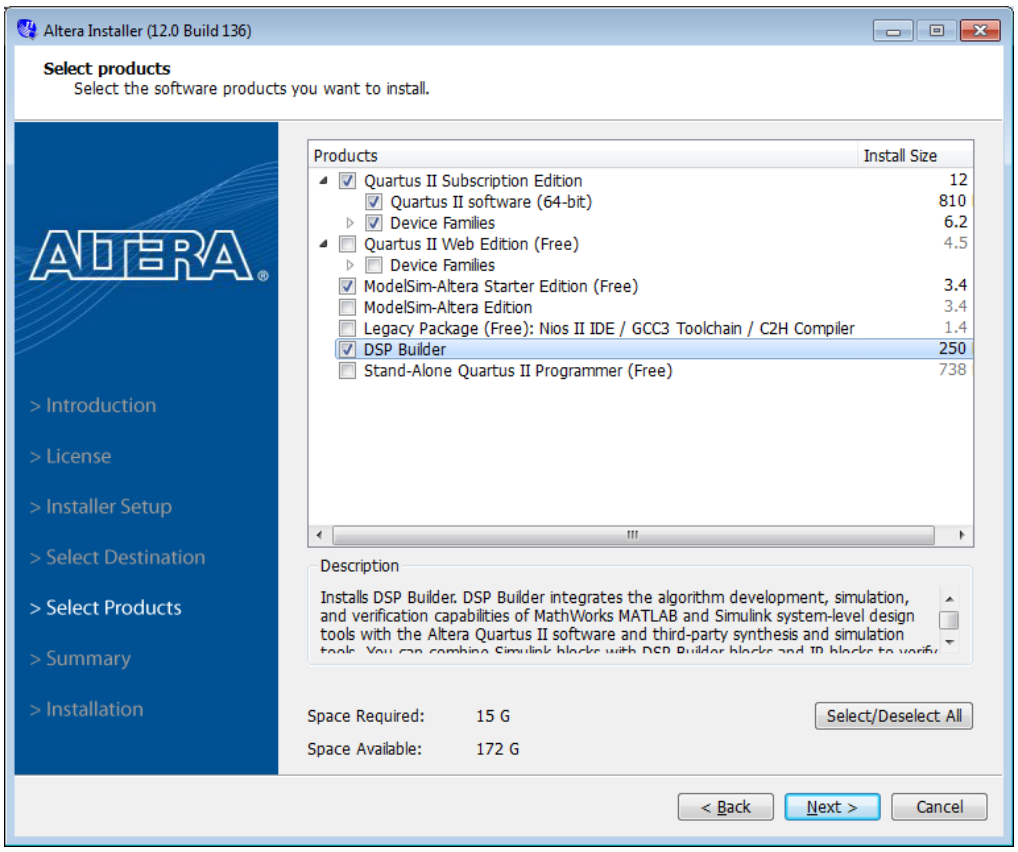
Notes to Table 3–1:

(1) The DSP Builder advanced blockset uses Simulink fixed-point types for all operations and requires licensed versions of Simulink Fixed Point. Altera also recommends DSP System Toolbox and the Communications System Toolbox, which some design examples use.

Obtaining and Installing DSP Builder

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

Figure 3–1. Select Components—DSP Builder

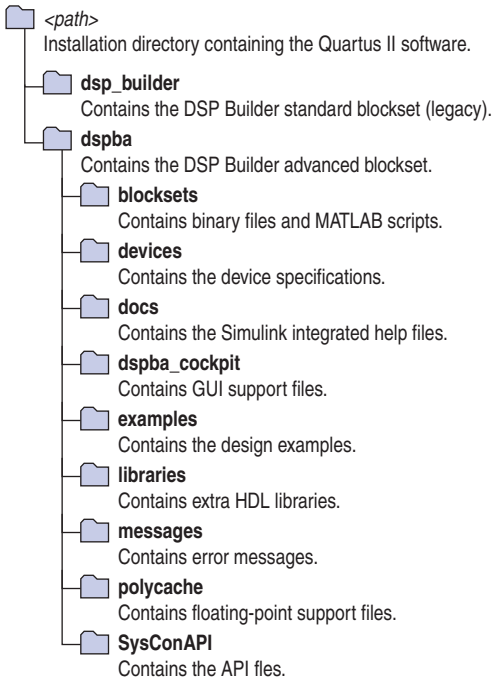


After installing DSP Builder, the Altera DSP Builder standard blockset and the Altera DSP Builder advanced blockset libraries are available in the Simulink library browser in the MATLAB software.

Directory Structure

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



Starting DSP Builder

To start DSP Builder, follow one of these steps:

- On Windows OS, click on **Start**, point to **All Programs**, click **Altera <*version*>**, click **DSP Builder**, and click **Start in MATLAB version** *XX*.

  If you have multiple versions of MATLAB installed, you can start DSP Builder in your desired version from this menu.

- On Linux OS, use the following command, which automatically finds MATLAB.

  ```
  <path to the Quartus II software>/dsp_builder/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

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.

You can use DSP Builder blocks to create DSP designs and you can 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.

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.

For more information about licensing DSP Builder, refer to the *Altera Software Installation and Licensing Manual*.

The Quartus II software recommends you specify a path to an LM_LICENSE_FILE variable, but it also allows you to use an explicit path to a license file. However, DSP Builder allows you to specify a path to only an LM_LICENSE_FILE variable.

Upgrading from Earlier Versions

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.

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this document.

Date	Version	Changes Made
June 2014	14.0	Updated MATLAB version support for DSP Builder v14.0.
November 2013	13.1	Updated MATLAB version support for DSP Builder v13.1.
May 2013	13.0	Updated MATLAB version support for DSP Builder v13.0.
November 2012	12.1	Updated MATLAB version support.
June 2012	12.0	■ Updated MATLAB version support ■ Deleted *Upgrading from v7.1* chapter ■ Updated installation instructions ■ Updated instructions for starting DSP Builder
November 2011	11.1	Updated MATLAB version support.
April 2011	11.0	■ Updated MATLAB version support ■ Added support for 64-bit MATLAB ■ Updated installation instructions

How to Contact Altera

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

Contact (1)	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Nontechnical support (general)	Email	nacomp@altera.com
(software licensing)	Email	authorization@altera.com

Note to Table:

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

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, *n* + 1. Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>***.pof** file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections in a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.
	The question mark directs you to a software help system with related information.
	The feet direct you to another document or website with related information.
	The multimedia icon directs you to a related multimedia presentation.
	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 envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.
	The feedback icon allows you to submit feedback to Altera about the document. Methods for collecting feedback vary as appropriate for each document.