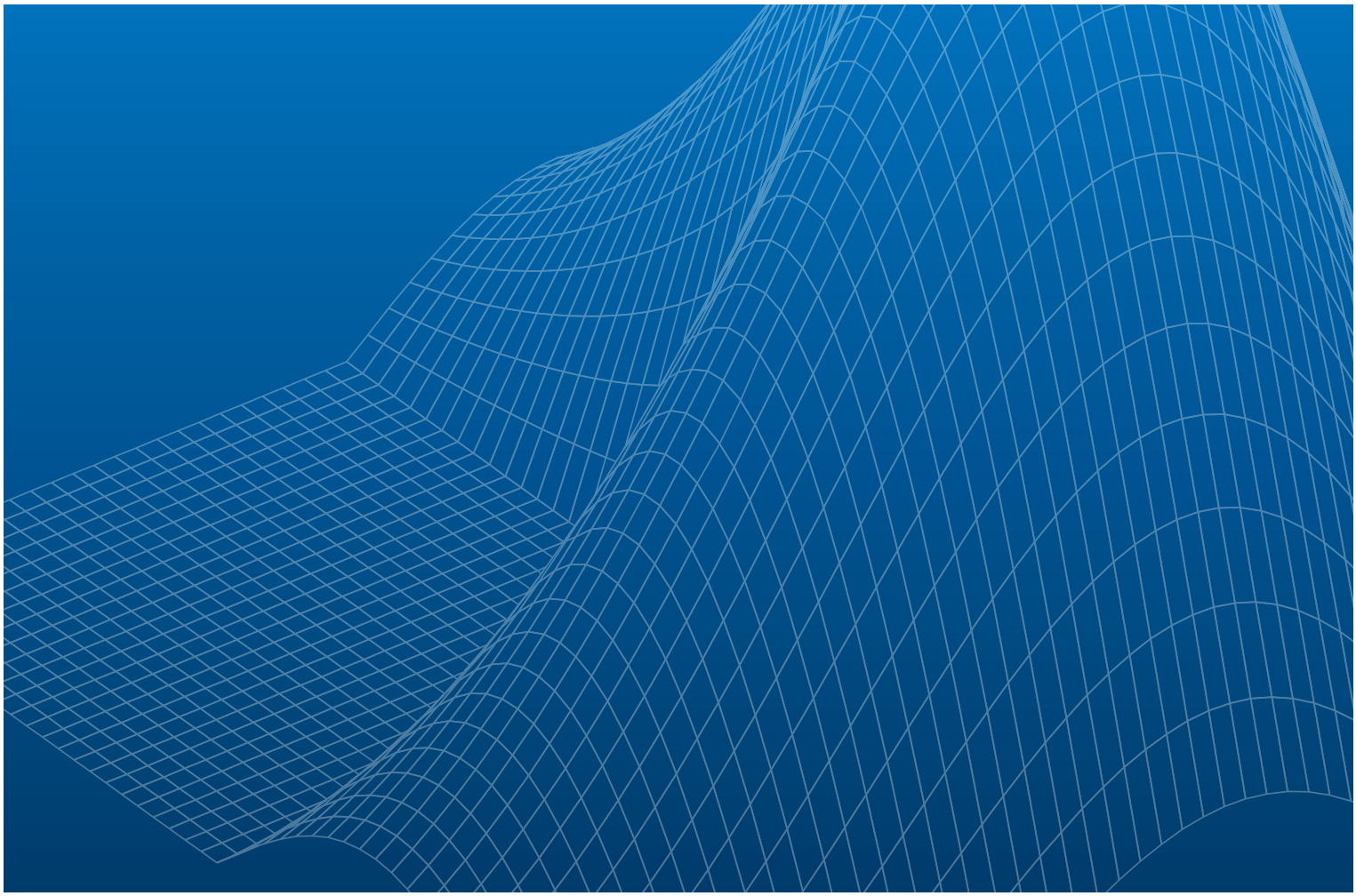

Model-Based Design Using Simulink, HDL Coder, and DSP Builder for Intel FPGAs

By Kiran Kintali, Yongfeng Gu, and Eric Cigan

WHITE PAPER

MathWorks®

Summary

This document describes how HDL Coder™ from MathWorks can be used with DSP Builder for Intel® FPGAs in an integrated FPGA workflow. We use an example to show how designers can integrate models built with DSP Builder Advanced Blockset into a Simulink® model, and how HDL Coder can generate HDL code for the complete design. This capability allows designers to reuse existing DSP Builder models when using HDL Coder to create new designs, or to incorporate target-optimized Intel FPGA IP blocks created for use with HDL Coder within Simulink models.

Introduction

MATLAB® and Simulink for Model-Based Design provide signal, image, and video processing engineers with a development platform that spans design, modeling, simulation, code generation, and implementation. Engineers who use Model-Based Design to target FPGAs or ASICs can design and simulate systems with MATLAB, Simulink, and Stateflow® and then generate bit-true, cycle-accurate, synthesizable Verilog® and VHDL® code using HDL Coder.

Alternatively, engineers who specifically target Intel FPGAs can use DSP Builder for Intel FPGAs, a plug-in to Simulink, to generate synthesizable hardware description language (HDL) code mapped to pre-optimized Intel FPGA implementations. DSP Builder includes the Advanced Blockset, a high-level synthesis technology that optimizes the high-level, untimed netlists into low-level, pipelined hardware for the target Intel FPGA device and desired clock rate.

Some projects benefit from implementing a workflow that combines the native Simulink workflow, device-independent code, and code readability offered by HDL Coder, with Intel-specific features and optimizations offered by DSP Builder Advanced Blockset.[1]

This paper describes a new workflow for designs that are created with blocks from both Simulink and DSP Builder Advanced Blockset.[2] Prior experience with MATLAB, Simulink, and DSP Builder will help you make the most of the examples in this paper.

Required Software

The models described in this paper are from the example included with HDL Coder, *Using Altera DSP Builder Advanced Blockset with HDL Coder*. Simulation and code generation from the model have been tested with the following versions of the software:

- MATLAB (R2013b or later)[3]
- Simulink
- HDL Coder (requires MATLAB Coder™ and Fixed-Point Designer™)
- Intel FPGA DSP Builder for Intel FPGAs (version 13.0sp1 or later)

[1] For a summary of the features of HDL Coder and Intel DSP Builder Advanced Blockset, see appendix.

[2] This integrated workflow is limited to models built with the DSP Builder Advanced Blockset. Models built using the Standard Blockset of DSP Builder for Intel FPGAs are not supported in this workflow. All subsequent references to DSP Builder (DSPB) subsystems describe models built from the DSP Builder Advanced Blockset.

[3] For the latest information, consult HDL Coder documentation, *Create an Altera DSP Builder Subsystem*.

To simulate, synthesize, and implement HDL code generated from the model, the following software is also required:

- Intel Quartus® Prime (version 13.0sp1 or later)

Setting Up the MATLAB Environment for HDL Code Generation and DSP Builder Advanced Blockset Integration

Before you can begin working on your model, you need to ensure that the MATLAB environment is aware of DSP Builder and that DSP Builder is configured to work with MATLAB.

To start DSP Builder, follow one of these steps (see *DSP Builder for Intel FPGAs Handbook – Volume 1: Introduction to DSP Builder* for more details):

- From Microsoft® Windows®, click **Start**, point to **All Programs**, click **Intel FPGA<version>**, click **DSP Builder**, and click **Start in MATLAB<version>**.
  - 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

Your MATLAB environment also needs to be made aware of the Quartus II installation. This is accomplished through the `hdlsetuptoolpath` command in MATLAB. The command below shows the typical path for a Windows PC Quartus II installation.

```
>> hdlsetuptoolpath('ToolName', 'Intel FPGA Quartus II', 'ToolPath',
'C:\intelfpga\13.0sp1\quartus\bin64\quartus.exe');
```

Note that `hdlsetuptoolpath` changes the system path and system environment variables for the current MATLAB session only. To execute the `hdlsetuptoolpath` command automatically when MATLAB starts, add it to your `startup.m` script.

Example: Design with Blocks from Simulink and DSP Builder Advanced Blockset

The example model (`hdlcoder_sldspba.slx`) performs FIR filtering. The top level of the design contains two subsystems, one implemented with blocks from the DSP Builder Advanced Blockset, and the other with native Simulink blocks (Figure 1). Because they are designed for synthesis on Intel FPGAs, Intel FPGA blocks will yield an optimized implementation of this FIR filter on an Intel FPGA. The Simulink subsystem contains a Stateflow block and a MATLAB function block, which provide flexibility in modeling complex control algorithms. Users can also explore various optimizations on the Simulink subsystem through HDL Coder.

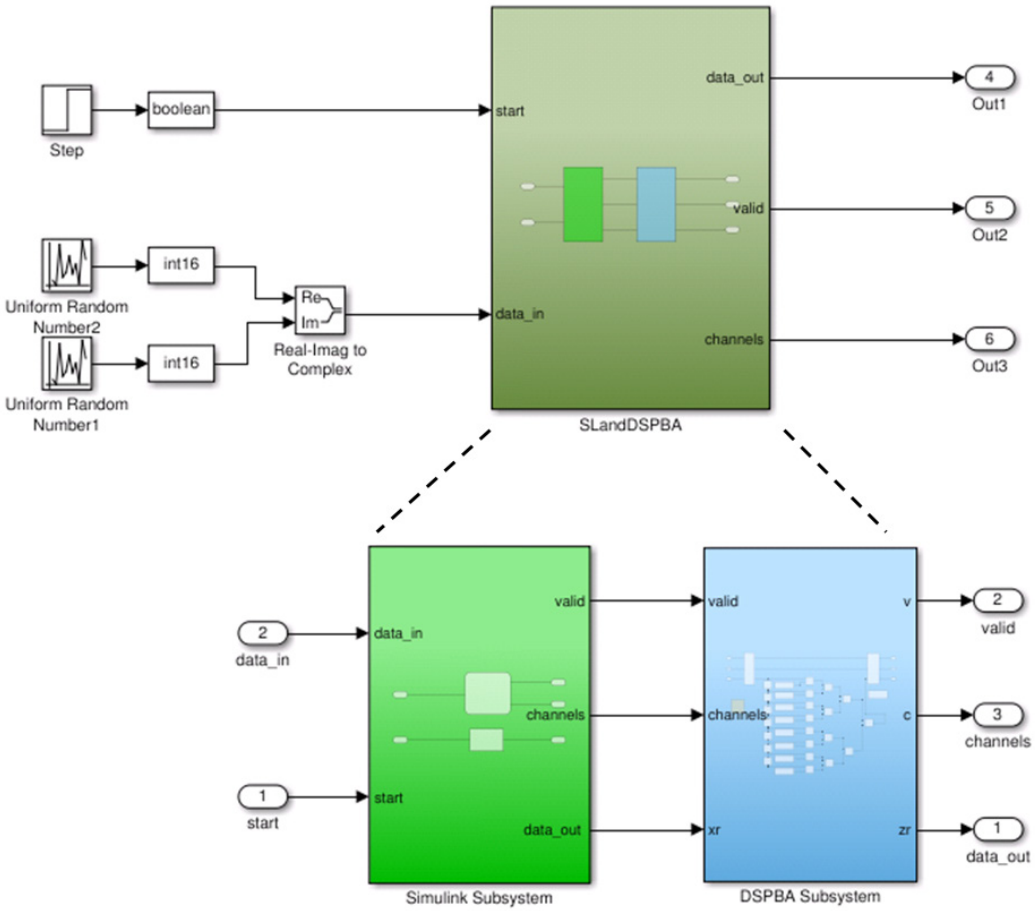

Figure 1. Simulink model of an FIR system (`hdlcoder_sldspba.slx`).

Preparing a Model for HDL Code Generation

In a typical development workflow, engineers model and simulate a design in Simulink, completing multiple iterations to identify and eliminate design problems in preparation for implementation. Before using HDL Coder and DSP Builder Advanced Blockset to generate code, you must prepare the model by:

- Creating a DSP Builder (DSPB) subsystem
- Setting code generation options for DSPB subsystems
- Setting code generation options for HDL Coder

Creating a DSPB Subsystem

To create a DSPB subsystem:

1. Put all the DSP Builder blocks in one subsystem. (Note: In general, HDL Coder supports the use of multiple DSPB subsystems provided that all DSP Builder blocks must be contained in DSPB subsystems; this example, however, demonstrates use of a single DSPB subsystem.)
2. Ensure the subsystem's **Architecture** parameter is set to **Module** (the default value). See the Code Generation Options for HDL Coder section (page 8) for more on this setting.
3. Place a Device block at the top level of the subsystem. You can have subsystem hierarchy in a DSPB subsystem, but there must be a Device block at the top level of the hierarchy.
4. Place a Control block and a Signals block at the top level of the model.

You can use the following MATLAB command to open the DSPB subsystem in the example model (Figure 2).

```
>> open_system('hdlcoder_sldspba/SLandDSPBA/DSPBA Subsystem');
```

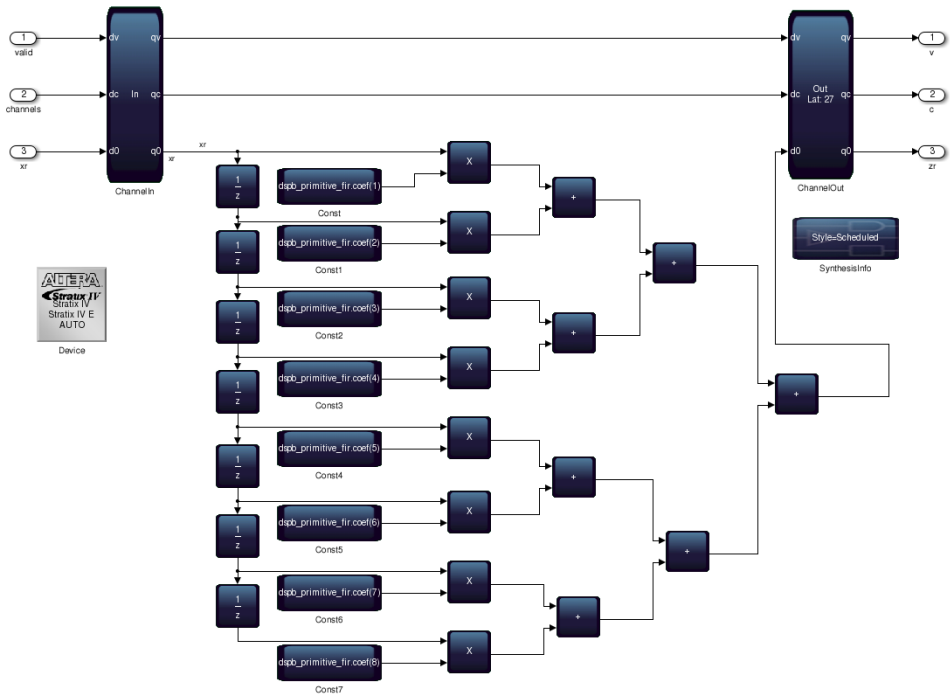

Figure 2. The image processing system's DSPB subsystem.

Setting Code Generation Options for DSP Builder

HDL Coder supports DSP Builder code generation with the following settings only:

- Device block (Figure 3):
  - The same Device must be chosen by all Device blocks and by HDL Coder if applicable.
- Control block (Figure 4):
  - **Generate Hardware** must be checked.
  - **Create Automatic Testbenches** must be unchecked.
- Signals block (Figure 5):
  - **Reset Active** must be same the setting in HDL Coder.

You do not need to configure these settings yourself. HDL Coder will modify and restore these settings on the DSP Builder blocks during code generation.

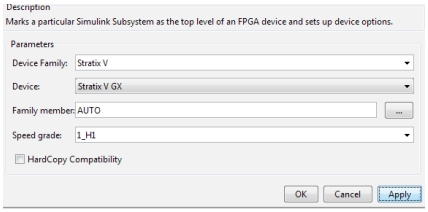

Figure 3. Options for the DSPB Device block.

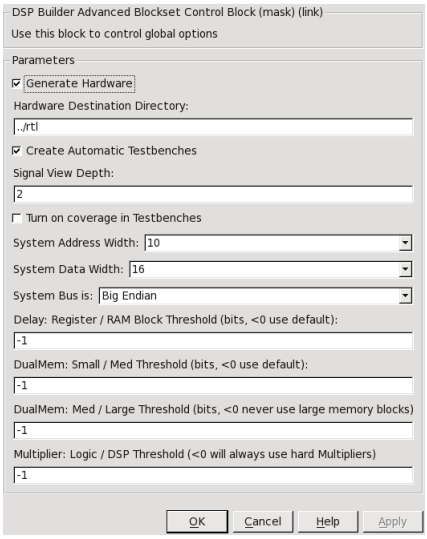

Figure 4. Options for the DSPB Control block.

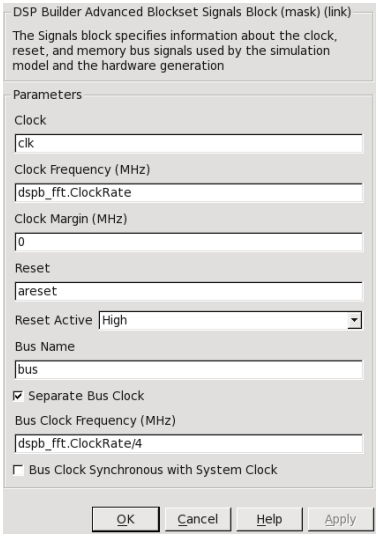

Figure 5. Options for the DSPB Signals block.

Setting Code Generation Options for HDL Coder

In addition to the settings described above for HDL code generation, HDL Coder requires the **Architecture** parameter of DSPB subsystems to be set to **Module** (Figure 6). If code generation is performed with HDL Workflow Advisor, the device settings for Workflow Advisor and DSPB Device blocks must be identical, as shown in Figures 3 and 7.

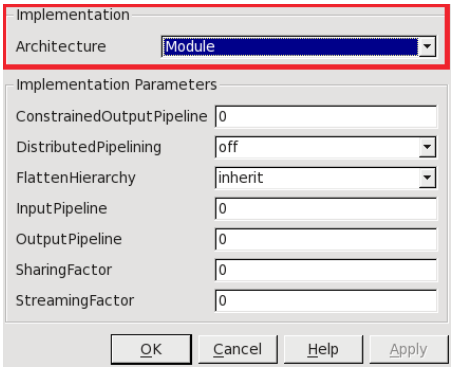

Figure 6. DSPB subsystem implementation parameters in HDL Coder.

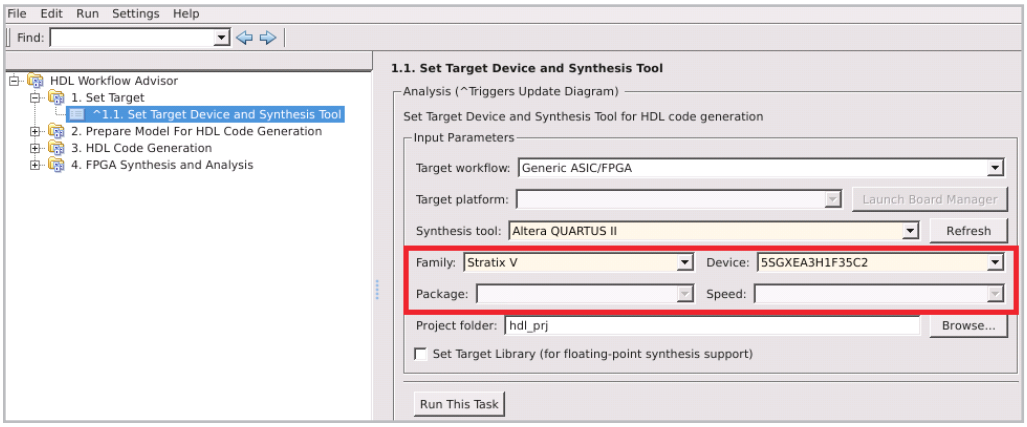

Figure 7. Tool and device options in HDL Workflow Advisor.

Generating HDL

You can generate HDL code from the configured model with the command line interface or with the GUI, as with any other model. For this example model, the command to generate code is:

```
>> makehdl('hdlcoder_sldspba/SLandDSPBA')
```

Refer to the *HDL Code Generation from a Simulink Model* tutorial provided with HDL Coder for details on how to generate code using the GUI.

Generating HDL Test Bench and Simulation Scripts

For this example model, the command to generate test bench and simulation scripts is:

```
>> makehdltb('hdlcoder_slsysgen/SLandSysGen');
```

Handling Simulation Mismatch When Valid Signal Not Asserted

The DSPB subsystem simulation may not match its generated code's behavior when the valid signal is not asserted under certain circumstances, such as when the design employs the folding optimization and/or floating-point support. You may observe such mismatches by turning on the **Folding** option in both `hdlcoder_sldspba/SLandDSPBA/DSPBA Subsystem/ChannelIn` and `hdlcoder_sldspba/SLandDSPBA/DSPBA Subsystem/ChannelOut`. The mismatch affects the downstream Simulink design and causes a test bench simulation failure.

To see the mismatch, you can turn on the folding setting for the ChannelIn and ChannelOut blocks:

```
>> set_param('hdlcoder_sldspba/SLandDSPBA/DSPBA Subsystem/ChannelIn',
'FoldingEnabled', 1);

>> set_param('hdlcoder_sldspba/SLandDSPBA/DSPBA Subsystem/ChannelOut',
'FoldingEnabled', 1);
```

Then, generate the HDL code and test bench again:

```
>> makehdl('hdlcoder_sldspba/SLandDSPBA');

>> makehdltb('hdlcoder_sldspba/SLandDSPBA');
```

After simulating the generated code and test bench, you can see that the outputs from HDL Coder match the reference data only when the valid signal is asserted as shown in Figure 8.

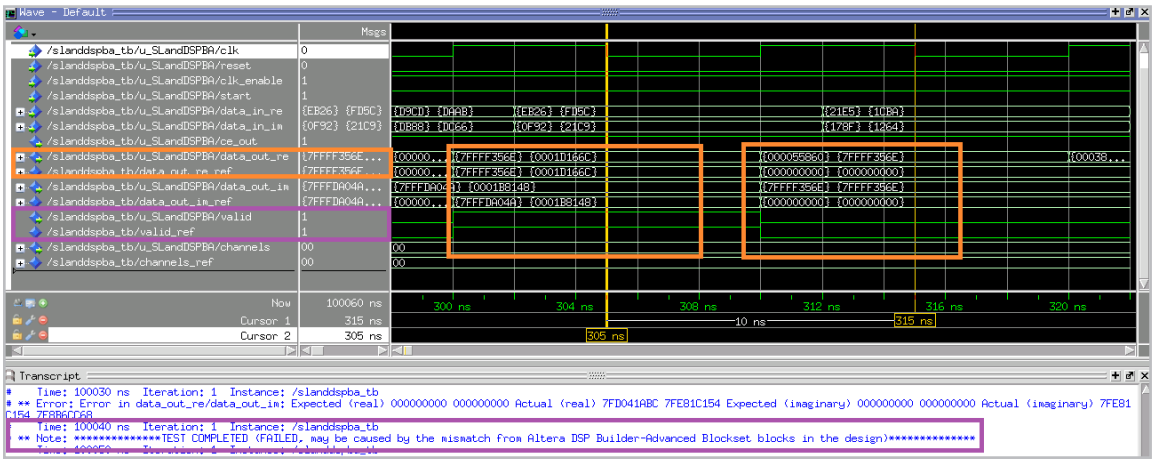

Figure 8. Simulation mismatch when valid signal not asserted.

As the message from the test bench indicates, the mismatch is expected.

To avoid this simulation mismatch, insert an enabled subsystem at the DSPB subsystem output signals, before they reach the Simulink part of your design or the output ports of the overall design. Figure 9 shows how to connect signals to the enabled subsystem.

```
>> open _ system('hdlcoder _ sldspba/SLandDSPBA2');
```

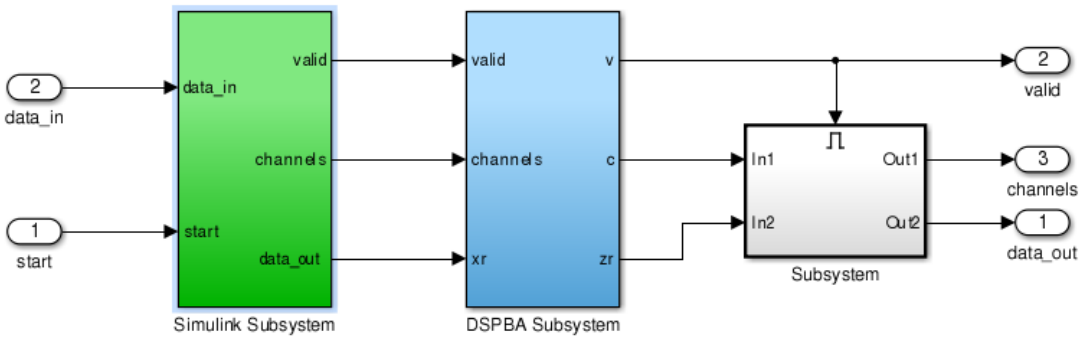

Figure 9. Use of enabled subsystem with DSPB subsystem to avoid mismatch when valid not asserted.

Further Reading

For more HDL Coder videos and examples, visit mathworks.com/products/hdl-coder/examples.html.

Appendix: Feature Comparison of HDL Coder and DSP Builder for Intel FPGAs Advanced Blockset

The table below summarizes the complementary features and benefits of HDL Coder and DSP Builder Advanced Blockset. Used independently, each approach provides an effective FPGA design flow.

Feature	HDL Coder	DSP Builder Advanced Blockset	Benefit
Software and hardware code generation	X		Partition algorithms between processors and hardware
Access to Simulink block library	X		Rapidly assemble system models using existing blocks
Support for native Simulink blocks	X		Easily migrate system model to hardware
Floating- to fixed-point conversion	X		Shorten design cycles
Design exploration	X	X	Rapidly explore hardware solution space
Automatic test bench generation	X	X	Verify hardware against system models
Transaction-Level Model (TLM) component generation	X		Use system-level modeling methodologies
Hardware cosimulation	X	X	Verify hardware implementations on Intel FPGA development boards
MATLAB to HDL code generation and hardware cosimulation	X		Follow a hardware design workflow without using Simulink
Readable, traceable HDL code	X		Streamline standards compliance and reporting
Access Intel FPGA IP in Simulink	X (when used with DSP Builder)	X	Generate implementations optimized for Intel FPGA targets
Analog data acquisition		X	Verify algorithms with real-world analog data
Hardware deployment	X	X	Deploy designs in hardware without FPGA design experience

92161v00 02/14