Black-Boxing in DSP Builder

October 2005, ver. 1.0 Application Note 402

Introduction

When using DSP Builder to design a system, you may need to integrate some modules or subsystems that were created using non-DSP Builder blocksets into the system. Using black boxes allows you to bridge the boundaries or partitions between the DSP Builder subsystems and the non-DSP Builder subsystems.

Non-DSP Builder subsystems may be control logic or complex state machines that are more suited for HDL coding than DSP Builder blocksets. In other cases, there may be existing HDL code (either Verilog HDL or VHDL) that needs to be integrated into the overall DSP Builder design. Rather than recoding these blocks using DSP Builder blocksets, it is more efficient to import them directly into the system. The DSP Builder design flow supports integrating HDL code or pre-compiled Quartus® II projects with DSP Builder blocks. DSP Builder also generates the simulation model automatically that allows the co-simulation of these HDL subsystems within the Simulink environment.

When evaluating a design at the system level, you may want to integrate generic Simulink blocksets into your DSP Builder design. The black-box interface can also be used to encapsulate non-DSP Builder blocksets such as generic Simulink blocksets.

Black-Boxing

Black-boxing allows SignalCompiler in DSP Builder to recognize subsystems that should not be altered during the conversion process from Simulink to HDL. SignalCompiler achieves this by using the black-box interface. The benefits of using black boxes in the design flow include:

- Providing an integrated design environment allowing co-simulation of existing HDL modules, pre-compiled Quartus II projects, and generic Simulink blocksets
- Improving efficiency by allowing you to reuse existing HDL modules
- Speeding up the verification process

There are two types of black-box interfaces in DSP Builder: *implicit* and *explicit*.

Implicit Black-Box Interfaces

The implicit black-box interface can be inferred by the HDL Import block. When you instantiate the HDL Import block, it automatically indicates the subsystem underneath is a black box. SignalCompiler recognizes the HDL Import block as a black box and bypasses it during the HDL translation.

This block is described in more detail in "HDL Import Block".

Explicit Black-Box Interfaces

You instantiate the explicit black-box interface to integrate non-DSP Builder blocksets into your DSP Builder design. Non-DSP Builder blocksets include generic Simulink blocksets and HDL designs imported using the SubSystemBuilder block. Using these blocks prevents SignalCompiler from translating the sub-element of the design into HDL code.

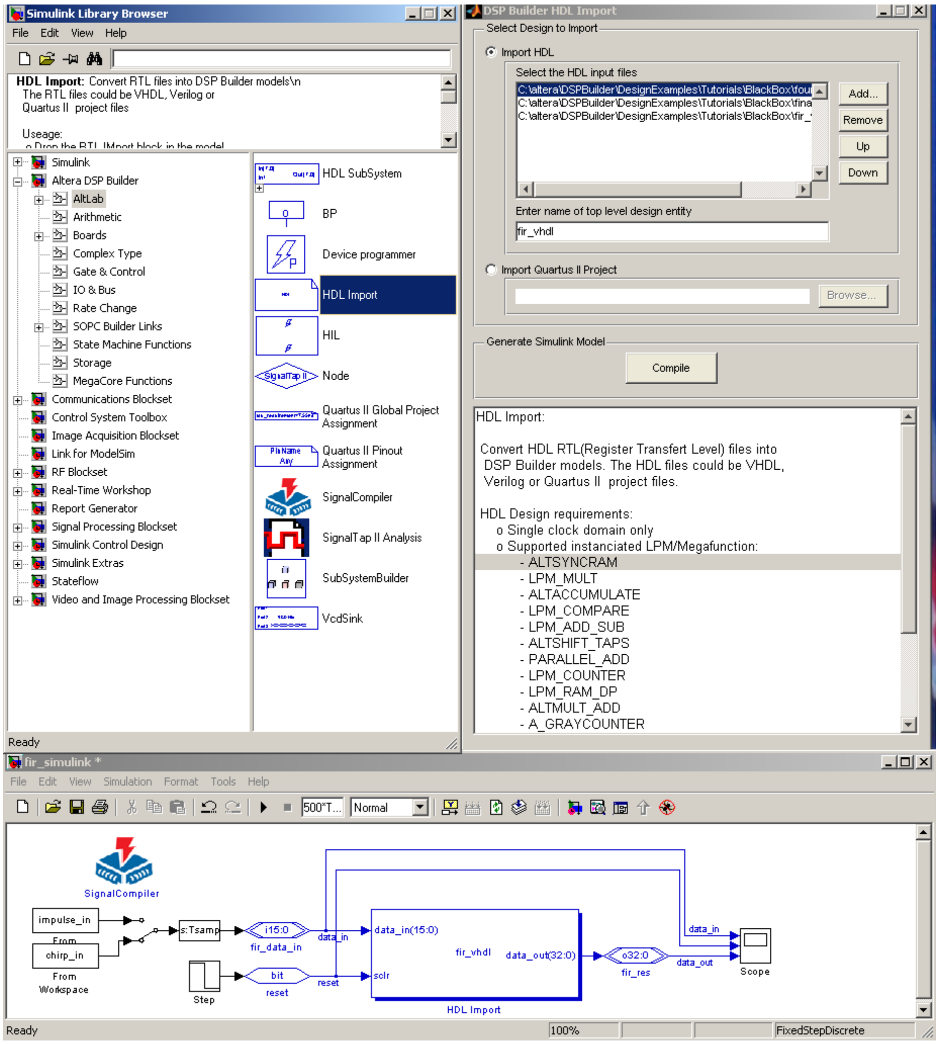
For more information on using the SubSystemBuilder block, refer to "SubSystemBuilder Block" on page 6.

The explicit black-box interface consists of the AltBus block in the Black Box Input Output mode. Using the AltBus block in this mode is described in more detail in "Generic Simulink Blocksets" on page 11.

HDL Import Block

The HDL Import block allows you to import either HDL code (VHDL or Verilog HDL) or an entire Quartus II project into your DSP Builder design. The HDL Import block also generates the simulation model for the imported subsystems. This feature supports hierarchical designs with multiple entities, spanning across multiple files. The HDL Import block automatically black-boxes the subsystem, allowing SignalCompiler to bypass the module during the HDL translation of the rest of the DSP Builder design. Figure 1 shows an example of using the HDL Import block.

Figure 1. HDL Import Example



HDL Import: Convert RTL files into DSP Builder models\n
The RTL files could be VHDL, Verilog or
Quartus II project files

Useage:
o Drop the RTL IMport block in the model

C:\altera\DSPBuilder\DesignExamples\Tutorials\BlackBox\fou
C:\altera\DSPBuilder\DesignExamples\Tutorials\BlackBox\fina
C:\altera\DSPBuilder\DesignExamples\Tutorials\BlackBox\fir_

fir_vhdl

HDL Import:

Convert HDL RTL(Register Transfert Level) files into
DSP Builder models. The HDL files could be VHDL,
Verilog or Quartus II project files.

HDL Design requirements:
o Single clock domain only
o Supported instanciated LPM/Megafunction:
- ALTSYNCRAM
- LPM_MULT
- ALTACCUMULATE
- LPM_COMPARE
- LPM_ADD_SUB
- ALTSHIFT_TAPS
- PARALLEL_ADD
- LPM_COUNTER
- LPM_RAM_DP
- ALTMULT_ADD
- A_GRAYCOUNTER

For more information on using the HDL Import block, refer to the *HDL Import Block* section in the *AltLab Library* chapter in the *DSP Builder Reference Manual*.

For a complete tutorial on using black boxes in DSP Builder, refer to the *Using Black Boxes for Non-DSP Builder Subsystems* chapter in the *DSP Builder User Guide*.

HDL Requirements for the HDL Import Block

The HDL Import feature only supports single clock domains. The HDL Import block automatically maps the input clock signal to the main system clock signal used in the rest of the DSP Builder design. If there are multiple clocks present in the HDL design, one clock is used as the implicit clock, while the additional clocks appear as ports on the Simulink block that is generated.

The HDL design can only consist of elements that use logic elements (LEs), DSP blocks, or memory bits. Other structures such as phase-locked loops (PLLs), low-voltage differential signaling (LVDS), and WYSIWYG structures are not supported.

The following megafunctions and libraries of parameterized modules (LPMs) are supported by the HDL Import block:

- a_graycounter
- altaccumulate
- altmult_add
- altshift_taps
- altsyncram
- lpm_abs
- lpm_add_sub
- lpm_compare
- lpm_counter
- lpm_mult
- lpm_mux
- lpm_ram_dp
- parallel_add

The megafunctions and LPMs may be either explicitly instantiated in the imported files or inferred by the Quartus II software. If your HDL file contains other megafunctions and LPMs not listed above, refer to "SubSystemBuilder Block" on page 6.

Simulating the HDL Import Subsystem

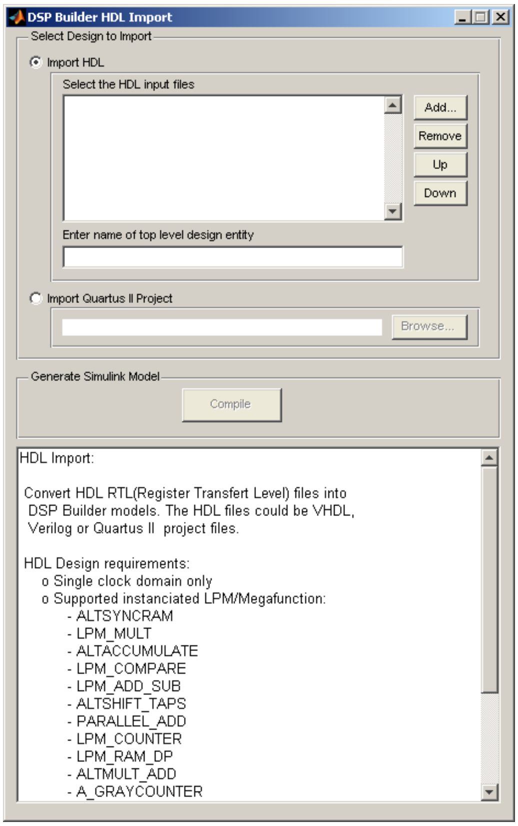
From the HDL Import block, you can generate the simulation model for the imported HDL module. This action enables co-simulation of the HDL modules with the rest of the DSP Builder blocksets in the DSP Builder design without having to invoke a separate RTL simulation tool. The seamless integration speeds up the verification process.

The system can also import multiple HDL modules. Each HDL module has its own individual simulation model generated by the HDL Import block.

Using the HDL Import Block

The HDL Import block has an easy-to-use interface to import either HDL code or a Quartus II project into the DSP Builder design (Figure 2). You can generate the Simulink simulation model from the same HDL Import interface to co-simulate the HDL modules with the rest of the DSP Builder design. The black-box interface is inferred by the use of the HDL Import block.

Figure 2. HDL Import Interface



```
HDL Import:

 Convert HDL RTL(Register Transfert Level) files into
  DSP Builder models. The HDL files could be VHDL,
  Verilog or Quartus II  project files.

 HDL Design requirements:
     o Single clock domain only
     o Supported instanciated LPM/Megafunction:
          - ALTSYNCRAM
          - LPM_MULT
          - ALTACCUMULATE
          - LPM_COMPARE
          - LPM_ADD_SUB
          - ALTSHIFT_TAPS
          - PARALLEL_ADD
          - LPM_COUNTER
          - LPM_RAM_DP
          - ALTMULT_ADD
          - A_GRAYCOUNTER
```

To import an HDL file using the HDL Import block, perform the following steps:

1. In MATLAB, open the **Simulink Library Browser**.

2. Open the **AltLab** library in the **Altera DSP Builder** blockset.

3. Drop the **HDL Import** block into your DSP Builder design.
4. Double-click on the **HDL Import** block.
5. In the **DSP Builder HDL Import** dialog box, select whether to import HDL files or a Quartus II project.
6. To import HDL files, select the files using the **Add** button. Files can be removed with the **Remove** button. The compilation order can be changed with the **Up** and **Down** buttons. Also, specify the name of the top level design entity.
7. To import a Quartus II project, select the project using the **Browse** button.
8. Click **Compile** to generate the simulation model. This action also generates a Simulink block configured with the ports matching the top-level entity from the imported HDL code or Quartus II project.
9. Simulate the HDL module with the rest of your DSP Builder design in Simulink.

For a complete tutorial on using black boxes in DSP Builder, refer to the *Using Black Boxes for Non-DSP Builder Subsystems* chapter in the *DSP Builder User Guide*.

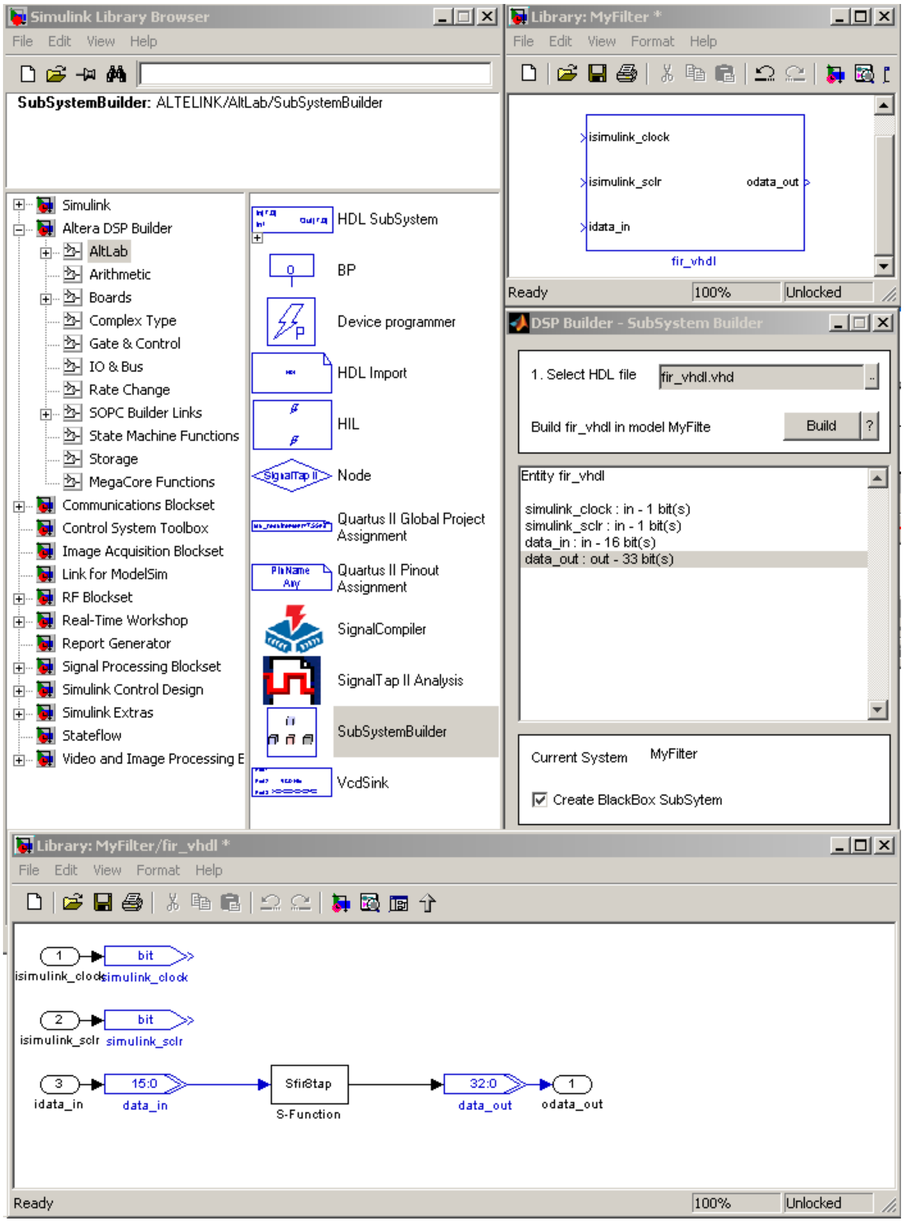
SubSystem-Builder Block

The SubSystemBuilder block, like the HDL Import block, allows you to import an HDL file into your DSP Builder design. The SubSystemBuilder block reads the port description for the top-level HDL design and creates an HDL SubSystem block with the corresponding input and output signals. If your HDL design contains an LPM or megafunction not supported by the HDL Import block, you can use the SubSystemBuilder block.

Unlike the HDL Import block described in the previous section, the SubSystemBuilder block does not automatically generate a Simulink simulation model for the imported HDL design. The SubSystemBuilder block allows you to create your own custom Simulink simulation model from non-DSP Builder blocks for faster simulation speed.

The SubSystemBuilder block can automatically black-box the subsystem, allowing SignalCompiler to bypass the module during the HDL translation of the rest of the DSP Builder design. Figure 3 shows an example of using the SubSystemBuilder block.

Figure 3. SubSystemBuilder Example



1. Select HDL file: fir_vhdl.vhd

Build fir_vhdl in model MyFilte

Entity fir_vhdl

simulink_clock : in - 1 bit(s)
simulink_sclr : in - 1 bit(s)
data_in : in - 16 bit(s)
data_out : out - 33 bit(s)

Current System: MyFilter

☑ Create BlackBox SubSytem

For more information on using the SubSystemBuilder block, refer to the *SubSystemBuilder Block* section in the *AltLab Library* chapter in the *DSP Builder Reference Manual*.

For a complete tutorial on the SubSystemBuilder block, refer to the *Using Black Boxes for Non-DSP Builder Subsystems* chapter in the *DSP Builder User Guide*.

SubSystemBuilder Requirements

The SubSystemBuilder block automatically maps every input port named `simulink_clock` or `simulink_sclr` in the VHDL entity section or Verilog HDL port declaration section to the main Simulink system clock and synchronous clear, respectively.

You must declare synchronous clear and clock as inputs in the entity, even if your design uses an asynchronous reset and a clock. These inputs are routed to a global clock bin and a global synchronous clear pin. If the entity does not need a clock or a global synchronous clear, you should still declare these signals as inputs and leave them unconnected within the HDL design.

The VHDL entity should be formatted according to the following guidelines:

- The VHDL file contains a single entity
- Port direction: `in` or `out`
- Port type: STD_LOGIC or STD_LOGIC_VECTOR
- Bus size:
  - `a(7 DOWNTO 0)` is supported (bit 0 is the LSB and must be the value 0)
  - `a(8 DOWNTO 1)` is not supported
  - `a(0 TO 7)` is not supported
- One port declaration per line:
  - `a:STD_LOGIC;` is supported
  - `a,b,c:STD_LOGIC;` is not supported

The Verilog HDL module should be formatted according to the following guidelines:

- The Verilog HDL file contains a single module
- Port direction: `input` or `output`
- Bus size:
  - `input [7:0] a;` is supported (bit 0 is the LSB and must be the value 0)
  - `input [8:1] a;` is not supported
  - `input [0:7] a;` is not supported

- One port declaration per line
  - `input [7:0] a;` is supported
  - `input [7:0] a,b,c;` is not supported

Simulating With SubSystemBuilder Blocks

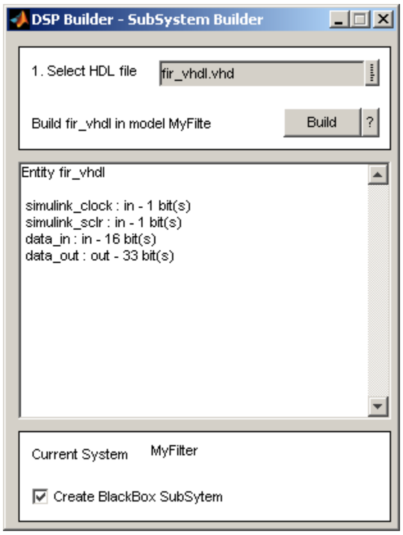
In addition to porting the HDL design to a Simulink subsystem, you need to create the Simulink simulation model for the block. The simulation model describes the functionality of the particular HDL subsystem. The following list shows the options available to create Simulink simulation models:

- Simulink generic library
- Simulink blocksets (for example: DSP blockset, Communications blockset)
- DSP Builder blockset
- MATLAB functions
- S-functions

Using the SubSystemBuilder Block

The SubSystemBuilder block has an easy-to-use interface to import an HDL file (Verilog HDL or VHDL) into the DSP Builder design (Figure 4). In the SubSystem Builder dialog box, you select the top-level HDL file and the port information is displayed automatically. Because the HDL code already exists, you need to indicate the SubSystem is a black box to bypass the conversion process. By turning on the **Create BlackBox SubSystem** option in the SubSystem Builder dialog box, you automatically create the black-box interface in the generated HDL SubSystem.

Figure 4. SubSystem Builder Interface


DSP Builder - SubSystem Builder
1. Select HDL file
fir_vhdl.vhd
Build fir_vhdl in model MyFilte
Build
?
Current System
MyFilter
☑ Create BlackBox SubSytem

```
Entity fir_vhdl

simulink_clock : in - 1 bit(s)
simulink_sclr : in - 1 bit(s)
data_in : in - 16 bit(s)
data_out : out - 33 bit(s)
```

To import an HDL file using the SubSystemBuilder block, perform the following steps:

1. In MATLAB, open the **Simulink Library Browser**.
2. Open the **AltLab** library in the **Altera DSP Builder** blockset.
3. Drop the **SubSystemBuilder** block into your DSP Builder design.
4. Double-click on the **SubSystemBuilder** block.
5. In the **SubSystem Builder window**, select the top-level HDL file (either Verilog HDL or VHDL) to import into your DSP Builder design. The SubSystemBuilder block reads in the ports of the HDL file and displays them.
6. Turn on **Create BlackBox SubSystem**.
7. Click **Build**. This action creates the HDL SubSystem block with the corresponding ports based on the imported HDL file.
8. Create the Simulink simulation model for the imported block using one of the techniques listed in "Simulating With SubSystemBuilder Blocks" on page 9.

9. Simulate the black box with the rest of your system in Simulink.

For a complete tutorial on the SubSystemBuilder block, refer to the *Using Black Boxes for Non-DSP Builder Subsystems* chapter in the *DSP Builder User Guide*.

Generic Simulink Blocksets

In the Simulink environment, you have access to different blocksets offered by The MathWorks. These blocksets include the Communication blockset, Signal Processing blockset, and Video and Image Processing blockset. When designing your system, you can mix these Simulink blocksets with DSP Builder blocksets.

Co-Simulating With Simulink Blocksets

You can mix and match DSP Builder blocksets with non-DSP Builder blocksets in your Simulink design for simulation purposes. The ability to co-simulate with non-DSP Builder blocks gives you access to a wider selection of libraries. Also, some blocks are useful only in the simulation environment and are not required to be translated into hardware. For example, you can easily model a transmission channel in your communications system by integrating the channel blocks from the Communications blockset into your DSP Builder design.

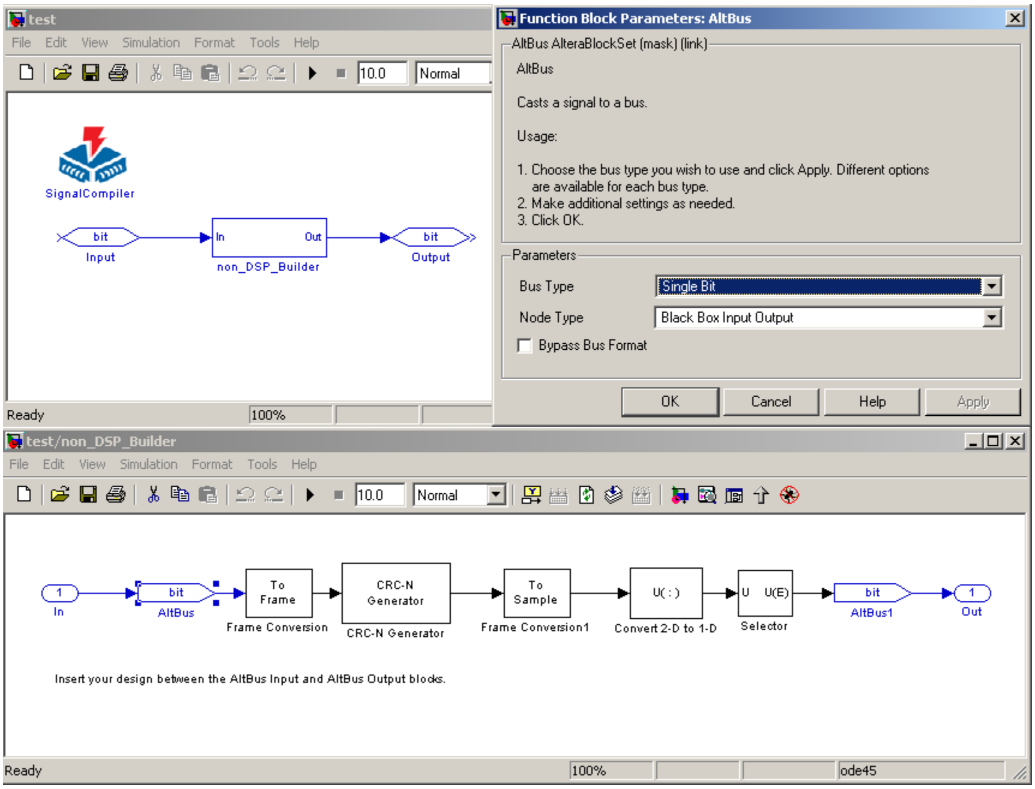
Using the HDL SubSystem Block for Black Boxing

Before converting designs that contain non-DSP Builder blocksets such as the Simulink blocksets, the latter must be encapsulated as a separate subsystem using the **HDL SubSystem** block from the DSP Builder AltLab library. This block enables you to create the black-box interface that indicates the boundaries of the subsystem to SignalCompiler.

The non-DSP Builder subsystems are identified as black boxes by setting the AltBus ports' node type to **Black Box Input Output** as shown in Figure 5.

SignalCompiler only converts the DSP Builder components to HDL. If needed, you must create the HDL code for the non-DSP Builder subsystem manually. The port names in the HDL code must match the port names in the **HDL SubSystem** block.

Figure 5. Black-Boxing Using AltBus Block Set to Black-Box Input Output Mode



AltBus AlteraBlockSet (mask) (link)

AltBus

Casts a signal to a bus.

Usage:

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

Parameters

Bus Type: Single Bit

Node Type: Black Box Input Output

☐ Bypass Bus Format

Insert your design between the AltBus Input and AltBus Output blocks.

Using the HDL SubSystem Block for Non-DSP Builder Blocks

The HDL SubSystem block allows you to create a subsystem for the non-DSP Builder blocks. You must replace the ports with the **AltBus** blocks in the **Black Box Input Output** mode to indicate the boundaries for the black-box subsystem. This allows you to co-simulate the non-DSP Builder blocks with the rest of the DSP Builder design.

To use the HDL SubSystem block, perform the following steps:

1. Open the **Simulink Library Browser**.
2. Open the **AltLab** library in the **Altera DSP Builder** blockset.
3. Drop the **HDL SubSystem** block into your DSP Builder design.

4. Rename the **HDL SubSystem** block to suit your non-DSP Builder subsystem. Double-click on the **HDL SubSystem** block. This action opens another Simulink model window.

5. Delete the Input and Output blocks in the model. Replace them with the **AltBus** block located in the **IO & Bus** library in the **Altera DSP Builder** blockset. Add or remove the **AltBus** blocks to suit the number of ports required in your non-DSP Builder subsystem.

6. Double-click on the **AltBus** block.

7. In the **Function Block Parameters: AltBus** dialog box, set the appropriate **Bus Type** and bit width information if applicable. Set the **Node Type** to **Black Box Input Output**. Click **OK**. Figure 5 shows an example of the **AltBus** block.

8. Repeat step 7 for each **AltBus** block in the subsystem.

9. Insert the non-DSP Builder blocks between the **AltBus** ports. These ports define the boundaries for the non-DSP Builder subsystem. This action prevents SignalCompiler from translating the sub-element of the design into HDL.

For more information on using the AltBus block to create a black box, refer to the *AltBus Black Box Input Output Mode* section in the *IO & Bus Library* chapter in the *DSP Builder Reference Manual*.

Conclusion

The black-boxing feature gives you access to a more integrated environment, in which you can combine existing HDL modules and other generic Simulink blocksets into your DSP Builder design. This integration is an important feature that allows the DSP Builder design environment to create larger and more complex system level designs.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
literature@altera.com