Synplify and Quartus II LogicLock Design Flow

August 2001, ver. 1.0 Application Note 165

Introduction

To maximize the benefits from the Quartus® II design software's LogicLock™ incremental design capability, a new design can be partitioned into a hierarchy of Verilog Quartus mapped (**.vqm**) files. This type of hierarchical structure allows the designer to have greater control over placement and preserve f_{MAX} performance. A hierarchical structure contains one VQM file for the top-level design and one or more VQM files for lower-level modules. By maintaining separate VQM files, you can place individual modules into LogicLock regions to control placement in the programmable logic device (PLD) and maintain performance of the module even as other modules are added to the project.

This application note describes how to generate multiple VQM files in the Synplify® software that allow you to use the LogicLock design flow in the Quartus II design software. This application note assumes that you are familiar with the Synplify software.

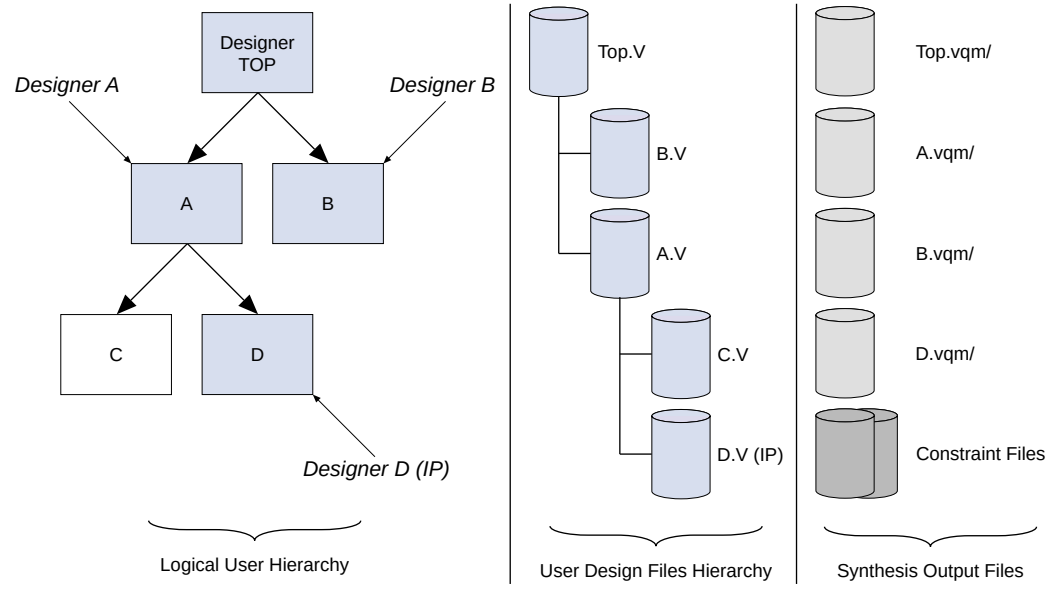
This application note applies to Synplify versions 6.2 and lower. To obtain the Synplify software, see the Synplify web site at **http://www.synplify.com**.

For more information on LogicLock regions and the LogicLock design flow, see *Application Note 161: Using the LogicLock Methodology in the Quartus II Design Software*.

Design Hierarchy

Figure 1 shows an example design hierarchy that separates several modules into separate VQM files. In Figure 1, modules TOP, A, B, C, and D are separate files that are coded in the same HDL language.

Figure 1. Example Design Hierarchy



In this example, there are four designers that are working on the project: designer A works on module A (including module C), designer B works on module B, and designer D works on module D. The fourth designer, designer TOP, is the system designer who performs the final place-and-route.

Generating Multiple VQM Files in the Synplify Software

To create multiple VQM files in the Synplify software, the general flow requires you to create a separate project for each module that you want to maintain as a separate VQM file and for the top-level design. The top-level design includes black-box instantiations of lower-level modules.

When synthesizing the lower-level modules and top-level design in the Synplify software, follow these guidelines.

For lower-level modules:

- Turn-on **Disable I/O Insertion** for the target technology.
- Read in HDL files for modules.
  - Modules may include black-box instantiations of lower-level modules that are also maintained as separate VQM files.
- Add constraints with SCOPE.
  - Enter the clock frequency with SCOPE to ensure that the design is correctly optimized.
  - In the **Attributes** tab, set `syn_netlist_hierarchy` to 0.

For top-level designs:

- Turn-off **Disable I/O Insertion** for the target technology.
- Read-in HDL files for top-level designs.
  - Black-box lower-level modules in the top-level design.
- Add constraints with SCOPE.
  - The constraints that affect lower-level modules should be the same as the constraints used when optimizing lower-level modules separately.
  - Enter the clock frequency with SCOPE to ensure that the design is correctly optimized.
  - In the **Attributes** tab, set `syn_netlist_hierarchy` to 0.

This section describes how to generate multiple VQM files in the Synplify software. By generating multiple VQM files, you can take advantage of the LogicLock incremental design flow in the Quartus II software.

Generating VQM Files for Lower-Level Modules

To generate VQM files for lower-level modules, the following steps must be completed:

- Set the target technology
- Read-in files
- Perform synthesis

Setting the Target Technology

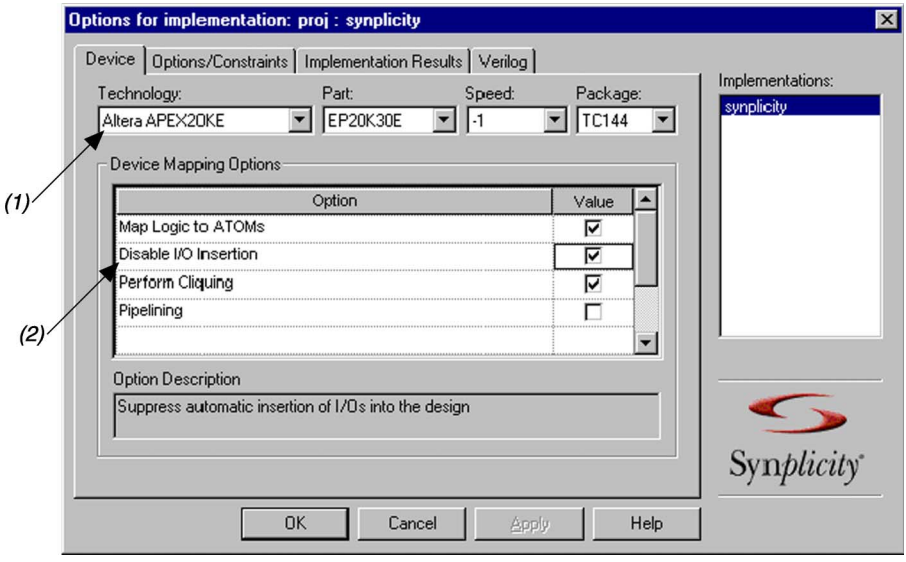
To set the target technology:

1. From the **Options for Implementation** dialog box (see Figure 2), set the target technology (Technology pull-down menu) to a family that supports the LogicLock design flow. At this time, the LogicLock design flow supports APEX™ II, APEX 20KE, APEX 20KC, and ARM®-based Excalibur™ devices.

2. Turn-on the **Disable I/O Insertion** option.

3. Click **Apply**.

Figure 2. Options for Implementation Dialog Box



Notes to Figure 2:

(1) Set device family.

(2) Turn on **Disable I/O Insertion**.

Reading-In Files

To read-in files:

1. From the **Synplify** window, click **Add**. The **Select Files to Add to Project** dialog box appears (see Figure 3).

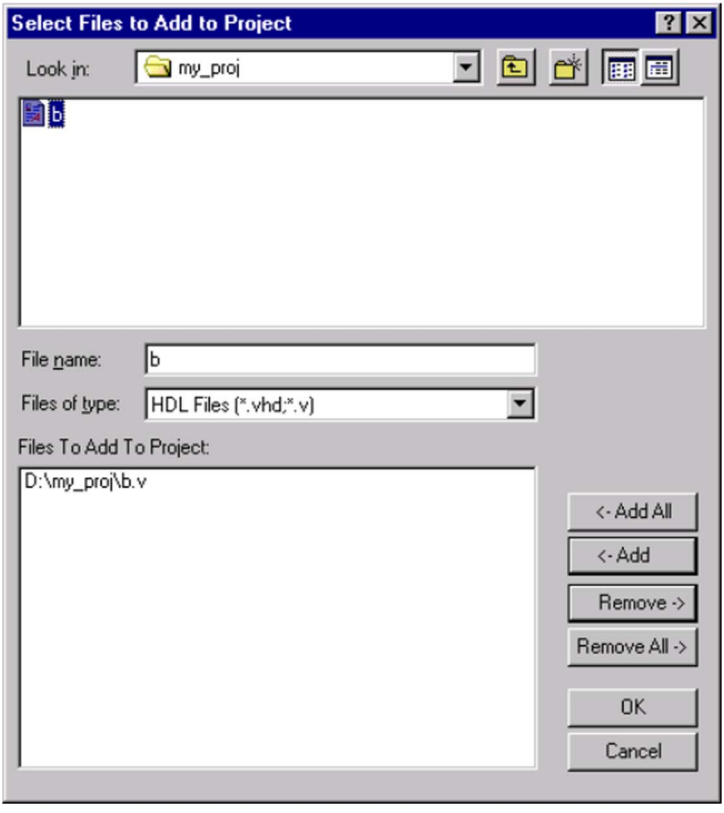
2. From the **Select Files to Add to Project** dialog box, click the appropriate HDL file(s).

3. Click **Add**.

4. Click **OK**.

The HDL files have now been added to the project.

Figure 3. Select Files to Add to Project Dialog Box



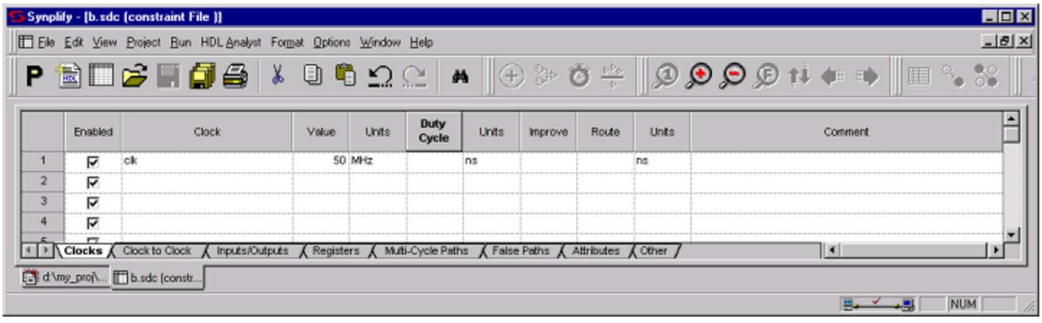
Performing Synthesis

To perform synthesis:

1. Use the **SCOPE** dialog box to enter any user constraints to the module (see Figure 4). To ensure that the design is optimized, enter the desired clock frequency.

2. In the Attributes tab (**SCOPE** dialog box), set `syn_netlist_hierarchy` to 0.

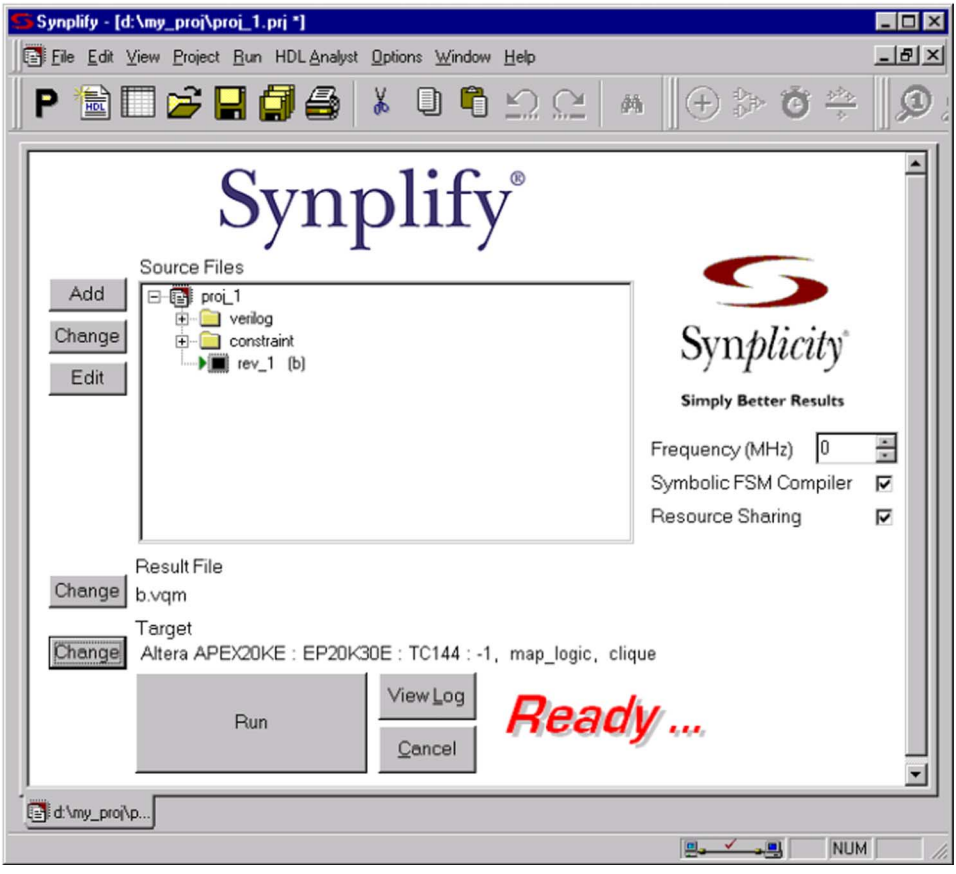
Figure 4. SCOPE Dialog Box



3. From the **Synplify** window, click **Run** (see Figure 5). The VQM netlist file is created.

You can now use the netlist for place-and-route.

Figure 5. Synplify Window

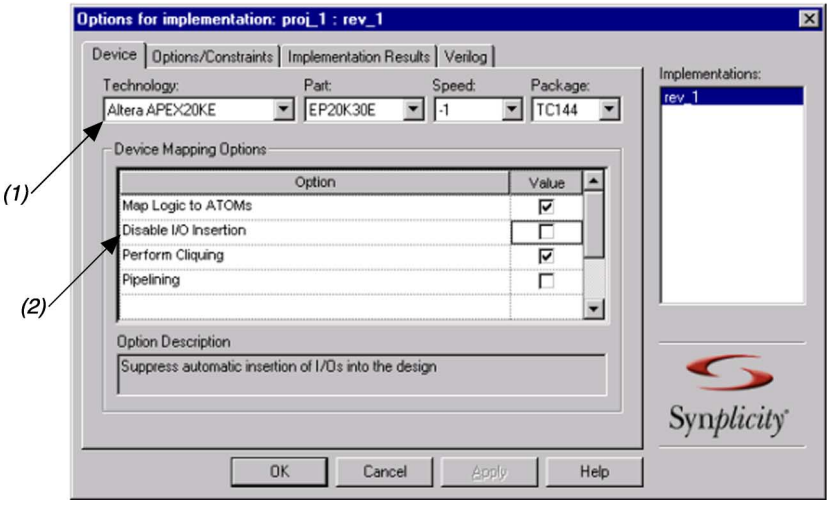


Generating VQM Files for the Top-Level Design

To generate a VQM file for the top-level design, follow all of the steps described in the "Generating VQM Files for Lower-Level Modules" on page 3 section except for one: turn-off **Disable I/O Insertion** (see Figure 6).

Although VHDL is not case-sensitive, VQM (a subset of Verilog) is case-sensitive. Entity names and their port declarations are forwarded to the VQM. Black-box names and port declarations are similarly forwarded to the VQM. To prevent case-sensitive mismatches between VQM, use the same capitalization for black-box and entity declarations in VHDL designs.

Figure 6. Options for Implementation Dialog Box



Notes to Figure 6:

(1) Set device family.

(2) Turn off **Disable I/O Insertion**.

Black-Boxing Modules Example

This section describes an example of black-boxing modules using the files described in Figure 1 on page 2. This example includes the following steps:

- Black-boxing the modules.
- Synthesizing the top-level file.
- Performing flow in batch mode.

VHDL and Verilog HDL examples are included in the black-boxing modules example.

Black-Boxing the Modules

To black-box the modules:

1. Generate a VQM file for module D (see "Generating VQM Files for Lower-Level Modules" on page 3). Use **D.V** as the source file.

2. Generate a VQM file for module A (see "Generating VQM Files for Lower-Level Modules" on page 3). Use **A.V** and **C.V** as the source files. Make sure to black-box module D, which was already optimized in the previous step.

Verilog Black-Boxing

The attribute `syn_black_box` instructs the Synplify software to treat the instance `U1` as a black-box. To black-box module D in Verilog, use the following **A.V** file example (see Figure 7):

Figure 7. A.V File Example for Verilog Black-Boxing

```
module A (data_in,clk,clrn,e,ld,data_out );

input [15:0] data_in;
clk, clrn, e, ld;
output [15:0] data_out;
reg [15:0] cnt_out;
reg [15:0] reg_a_out;

D U1( .data_in (data_in),.clk (clk), .clrn (clrn), .e(e), .ld (ld),
.data_out(cnt_out) );

C U2 ( .d(cnt_out), .clk (clk),.clrn (clrn), e(e), .q (reg_out));

endmodule

// Module Declarations of Sub-Blocks C and D follow here

// Insert Module Declaration for C

// Insert Module Declaration for D with the syn_black_box
// directive
module D (data_in, clk, clrn, e, ld, data_out) /* synthesis syn_black_box */;

input [15:0] data_in;
clk, clrn, e, ld;
output [15:0] data_out;

endmodule
```

VHDL Black-Boxing

The `syn_black_box` attribute can also be used in the VHDL flow. To black-box module D in VHDL, use the following **A.VHD** file example (see Figure 8):

Figure 8. A.VHD File Example for VHDL Black-Boxing (Part 1 of 2)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY synplify;
use synplify.attributes.all;

ENTITY A IS
PORT    ( data_in      : IN INTEGER RANGE 0 TO 15;
          clk          : IN STD_LOGIC;
          clrn         : IN STD_LOGIC;
          e            : IN STD_LOGIC;
          ld           : IN STD_LOGIC;
          data_out     : OUT INTEGER RANGE 0 TO 15
        );

END A;

ARCHITECTURE a_arch OF A IS

COMPONENT C PORT(
          data_in      : IN INTEGER RANGE 0 TO 15;
          clk          : IN STD_LOGIC;
          clrn         : IN STD_LOGIC;
          e            : IN STD_LOGIC;
          ld           : IN STD_LOGIC;
          data_out     : OUT INTEGER RANGE 0 TO 15
        );
END COMPONENT;

COMPONENT D PORT(
          d            : IN INTEGER RANGE 0 TO 15;
          clk          : IN STD_LOGIC;
          clrn         : IN STD_LOGIC;
          e            : IN STD_LOGIC;
          q            : OUT INTEGER RANGE 0 TO 15
        );
END COMPONENT;

attribute syn_black_box of D: component is true;

signal cnt_out    : INTEGER RANGE 0 TO 15;
signal reg_a_out  : INTEGER RANGE 0 TO 15;

        BEGIN
          CNT : C
          PORT MAP     (
                       data_in    => data_in,
                       clk        => clk,
                       clrn       => clrn,
                       e          => e,
```

Figure 8. A.VHD File Example for VHDL Black-Boxing (Part 2 of 2)

```
                        ld          => ld,
                        data_out    => cnt_out
                        );

          REG_A : D
          PORT MAP      (
                        d           => cnt_out,
                        clk         => clk,
                        clrn        => clrn,
                        e           => e,
                        q           => reg_a_out
                        );

          REG_B : D
          PORT MAP      (
                        d           => reg_a_out,
                        clk         => clk,
                        clrn        => clrn,
                        e           => e,
                        q           => data_out
                        );

END a_arch;
```

Creating the Top-Level File

Using the file examples created in the previous section, create a top-level file that instantiates the modules and black-box them in the HDL file (see "Generating VQM Files for the Top-Level Design" on page 7).

Verilog Top-Level File

For a Verilog top-level file, use the following Verilog framework (see Figure 9):

Figure 9. Verilog Framework (Part 1 of 2)

```
module top (data_in,clk,clrn,e,ld,data_out );

// port declarations

A U1 ( .data_in (data_in),.clk (clk), .clrn (clrn), .e(e), .ld (ld),
.data_out(cnt_out) );

B U2 ( .d(cnt_out), .clk (clk),.clrn (clrn), e(e), .q (reg_out));

endmodule
```

Figure 9. Verilog Framework (Part 2 of 2)

```
// Module Declarations of Sub-Blocks A and B follow here

// Insert Module Declaration for A with syn_black_box directive
module A (data_in, clk, clrn, e, ld, data_out) /* synthesis syn_black_box */;

// Insert Module Declaration for B with syn_black_box directive
module (data_in, clk, clrn, e, ld, data_out) /* synthesis syn_black_box */
```

VHDL Top-Level File

For a VHDL top-level file, use the following VHDL framework (see Figure 10).

Figure 10. VHDL Framework (Part 1 of 2)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY synplify;
use synplify.attributes.all;

ENTITY top IS

        PORT
        (
          -- Insert Port Declarations Here
        );

END top;

ARCHITECTURE a OF top IS
        COMPONENT A PORT (
          -- Insert Port Declarations Here
        );
        END COMPONENT;

        COMPONENT B
        PORT(
          -- Insert Port Declarations Here
        );

        END COMPONENT;

attribute syn_black_box of A: component is true;
attribute syn_black_box of B: component is true;
```

Figure 10. VHDL Framework (Part 2 of 2)

```
BEGIN
        CNT : A
        PORT MAP (
          -- Insert Port Mapping Here
        );

        REG_A : B
        PORT MAP(
          -- Insert Port Mapping Here
        );

END a;
```

Synthesizing the Top-Level File

Synthesize the top-level file (see "Generating VQM Files for the Top-Level Design" on page 7).

Performing Flow in Batch Mode

To perform this flow in batch mode, you can modify the examples shown in Figures 11 through 13.

Setting the Target Technology

To set the target technology, use the following settings (see Figure 11):

Figure 11. Batch Mode Target Technology Settings

```
set_option -technology APEX20KC
set_option -part EP20K400C
set_option -grade -7
set_option -package BC652
```

Reading-In and Synthesizing (Modules)

To read-in and synthesize modules, use the following insertion (see Figure 12):

Figure 12. Module Insertion for Batch Mode

```
add_file  D:/my_proj/B.VHD
project -run synthesis
```

Repeat this for the other modules to generate **A.VQM** and **D.VQM**.

Reading-In and Synthesizing (Top-Level File)

To read-in and synthesize the top-level file, use the following insertion (see Figure 13):

Figure 13. Top-Level File Insertion for Batch Mode

```
add_file  D:/my_proj/TOP.VHD
project -run synthesis
```

After you have completed the steps outlined in this section, you will have four VQM files: **Top.vqm**, **A.vqm**, **B.vqm**, **D.vqm**. These files can now be used in the Quartus II software and the LogicLock incremental design methodology.

Conclusion

The LogicLock incremental design flow uses multiple VQM files to help designers preserve performance of modules and have control over placement. Generating multiple VQM files is simple, and can be done by following the guidelines in this document.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 544-7104
Literature Services:
lit_req@altera.com