17. Cadence Encounter Conformal Support

QII53011-13.1.0

This chapter describes equivalence checking with the Cadence Encounter Conformal Logic Equivalence Check (LEC) software. The Quartus® II software provides formal verification support for Altera® designs through interfaces with the Conformal LEC software.

Logic equivalence checking uses Boolean arithmetic techniques to compare the logical equivalence of two versions of the same design. You can use the Conformal LEC software to verify the functional equivalence of a post-synthesis Verilog Quartus Mapping (**.vqm**) netlist from the Synopsys Synplify Pro software, a post-fit Verilog Output File (**.vo**) from the Quartus II software, or both. You can also use the Conformal LEC software to verify the functional equivalence of the register transfer level (RTL) source code and post-fit **.vo** with the Quartus II software when using Quartus II integrated synthesis.

This chapter discusses the following topics:

Formal Verification Versus Simulation

Formal verification is not a replacement for vector-based simulation. Formal verification only complements the existing vector-based simulation techniques to speed up the verification cycle. Vector-based simulation techniques of gate-level designs can take a considerable amount of time.

You can use vector-based simulation techniques to perform the following functions:

- Verify design functionality
- Verify timing specifications
- Debug designs

Formal Verification: What You Must Know

There might be an impact on area and performance during recompilation of your design with the Quartus II software if you use the formal verification flow for the Conformal LEC software. The following factors might affect the area and performance of your design:

- Preserving hierarchy
- Implementing ROM by logic elements (LEs)
- Enabling retiming

Before you consider using the formal verification flow in your design methodology, refer to "Known Issues and Limitations" on page 17–16.

Formal Verification Design Flow

Altera supports formal verification with the Conformal LEC software for the following two synthesis tools:

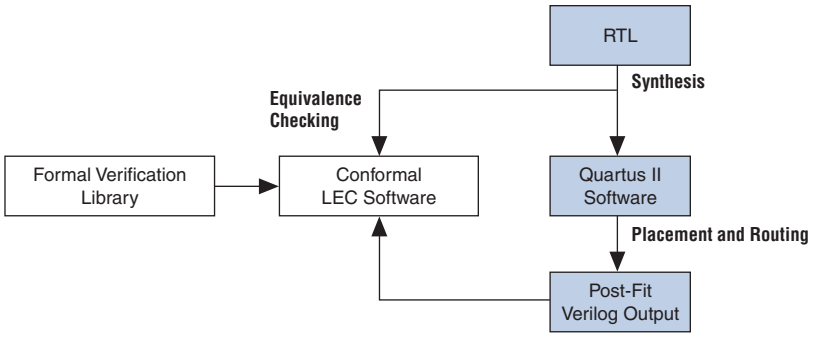
- "Quartus II Integrated Synthesis" on page 17–3
- "Synplify Pro" on page 17–3

The following sections describe the supported design flows for these synthesis tools.

Quartus II Integrated Synthesis

Figure 17–1 shows the design flow for formal verification with Quartus II integrated synthesis. This flow performs equivalence checking of the RTL source code and the post-fit netlist generated by the Quartus II software. The RTL source code can be in Verilog HDL or VHDL format. The Quartus II-generated post-fit netlist is in Verilog HDL format.

Figure 17–1. Formal Verification Using Quartus II Integrated Synthesis and the Conformal LEC Software



EDA Tool Support for Quartus II Integrated Synthesis

The formal verification flow using the Quartus II software and Conformal LEC software supports the following software versions and operating systems:

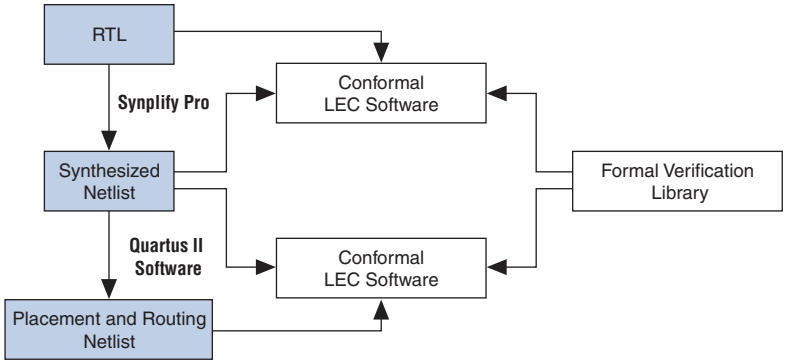
- The Quartus II software beginning with version 4.2
- The Conformal LEC software beginning with version 4.3.5A
- Linux operating system

Synplify Pro

Figure 17–2 shows the design flow for formal verification with Synplify Pro Synthesis performing equivalency checking for the post-synthesis netlist from Synplify Pro and the post-fit netlist generated by Quartus II software.

For more information about performing equivalence checking between RTL source code and post-synthesis netlists generated from the Synplify Pro software, refer to the Synplify Pro documentation.

Figure 17–2. Formal Verification Flow Using Synplify Pro and the Conformal LEC Software



RTL Coding Guidelines for Quartus II Integrated Synthesis

The Conformal LEC software compares the RTL source code against the Quartus II-generated post-fit netlist. The Conformal LEC software and Quartus II integrated synthesis parse and compile the RTL description differently. Quartus II integrated synthesis supports some RTL features that the Conformal LEC software does not support and vice versa. The style of the RTL source code is of particular concern because neither tool supports every construct, leading to potential formal verification mismatches. For example, different encoding mechanisms for state machine extraction can result in different structures. Therefore, Quartus II integrated synthesis and the Conformal LEC software must interpret the RTL source code in the same manner for successful verification.

The following section describes how you can identify and prevent problems that may occur in the formal verification flow.

For more information about RTL coding styles for Quartus II integrated synthesis, refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*.

Some of the coding guidelines apply to both Quartus II integrated synthesis and Synplify Pro flow, as indicated in each of the guidelines in the following sections.

Synthesis Directives and Attributes

You can use synthesis directives, also known as pragmas, to compare and verify the RTL source codes against the post-fit **.vo** netlist from the Quartus II software.

Quartus II integrated synthesis and the Conformal LEC software support the "synthesis" and "synopsys" trigger keywords. When Quartus II integrated synthesis does not recognize a keyword (such as "verplex"), the Quartus II software disables the keyword in the formal verification scripts produced for use with the Conformal LEC software. Therefore, you must use caution with unsupported pragmas because the unsupported pragmas can lead to verification mismatches.

Example 17–1 and Example 17–2 show that you can use Quartus II integrated synthesis to synthesize an RTL source code with the `read_comments_as_HDL` () synthesis directive.

Example 17–1. Verilog HDL Example of Read Comments as HDL

```
// synthesis read_comments_as_HDL on
// my_rom lpm_rom(.address (address),
// .data (data));
// synthesis read_comments_as_HDL off
```

Example 17–2. VHDL Example of Read Comments as HDL

```
-- synthesis read_comments_as_HDL on
-- my_rom : entity lpm_rom
-- port map (
-- address => address,
-- data => data, );
-- synthesis read_comments_as_HDL off
```

The Conformal LEC software does not support the `read_comments_as_HDL` synthesis directive, and the directive does not affect the Conformal LEC software.

Table 17–1 lists supported pragmas and trigger keywords for formal verification.

Table 17–1. Supported Pragmas and Trigger Keywords for Formal Verification

Pragmas	Trigger Keywords
full_case parallel_case pragma synthesis_off synthesis_on translate_off translate_on	synthesis synopsys

Do not use Verilog 2001-style pragma declarations. The Quartus II software and the Conformal LEC software support this style of pragma differently.

Fixed-Output Registers

Quartus II integrated synthesis and Synplify Pro eliminate registers that have fixed ouput. Quartus II integrated synthesis issues a warning message and adds an entry to the corresponding report panel in the formal verification folder of the **Analysis & Synthesis** section of the Compilation Report. If the Conformal LEC software does not find the same optimizations, the result can lead to unmapped points in the golden netlist. Example 17–3 shows logic causing register outputs to be fixed at a constant value.

Example 17–3. Verilog HDL Example Showing Fixed Register Outputs

```
module stuck_at_example{clk, a,b,c,d,out};
input a,b,c,d,clk;
output out;
reg e,f,g;
    always @(posedge clk) begin
        e <= a and g;// e is stuck at 0
        g <= c and e;// g is stuck at 0
        f <= e | b;
    end
assign out = f and d;
endmodule
```

In this module description, registers `e` and `g` are tied to logic `0`. In this example, the Quartus II software generates the following warning message:

```
Warning: Reduced register "g" with stuck data_in port to stuck value GND
Warning: Reduced register "e" with stuck data_in port to stuck value GND
```

Example 17–4 shows that Quartus II integrated synthesis adds a command to the formal verification scripts to inform the Conformal LEC software that a register is stuck at a constant value.

Example 17–4. Conformal LEC Script Showing Commands for Instance Equivalence

```
// report floating signals
// Instance-constraints commands for constant-value registers removed
// during compilation
// add instance constraints 0 e -golden
// add instance constraints 0 g -golden
```

Quartus II integrated synthesis comments the command in the formal verification script to force the Conformal LEC software to treat the register as stuck at a constant value and potentially hides a compilation error. You must verify that input to the `e` and `g` registers is constant in your design and uncomment the command to obtain accurate results.

Altera recommends recoding your design to eliminate registers that have fixed output.

ROM, LPM_DIVIDE, and Shift Register Inference

For formal verification, Quartus II integrated synthesis implements ROM and shift registers with LEs instead of with dedicated on-chip memory resources. Using LEs can be less area efficient than inferring a megafunction that you can implement in a RAM block. The Quartus II software generates a warning message to indicate that the software does not infer the megafunction. Quartus II integrated synthesis also reports a suggested ROM or shift register instantiation that enables you to either use the MegaWizard™ Plug-In Manager to create the appropriate megafunction explicitly, or to isolate the corresponding logic in a separate entity that you can set as a black box. By setting black box properties on a module or a particular entity, you are directing the formal verification tool not to look inside the module or entity for formal verification. If you set the black box properties on the corresponding megafunction before synthesis, you can verify the megafunction with the Conformal LEC software. For more information about setting black box properties on a particular module, refer to Table 17–2 on page 17–9.

If your design contains division functionality, the Quartus II software infers an LPM_DIVIDE megafunction. The Quartus II software treats the inferred LPM_DIVIDE megafunction as a black box for formal verification.

RAM Inference

When the Quartus II software infers the ALTSYNCRAM megafunction from the RTL source code, the Quartus II software generates the following warning message:

```
Created node "<mem_block_name>" as a RAM by generating altsyncram
megafunction to implement register logic with M512 or M4K memory block
or M-RAM. Expect to get an error or a mismatch for this block in the
formal verification tool.
```

The Quartus II software generates this warning message because the memory block (`altsyncram`) is a new instance in the post-fit netlist. The Quartus II software handles the ALTSYNCRAM megafunction as a black box by the formal verification tool. However, no such instance exists in the original RTL design, resulting in mismatch or error reporting in the formal verification tool.

Latch Inference

A combinational feedback loop implements a latch in Quartus II integrated synthesis. The Conformal LEC software infers a latch primitive in the Conformal LEC software library to implement a latch. This results in having a library on the golden side and a combinational loop with a cut point on the revised side, leading to verification mismatches. The Quartus II software issues a warning message whenever the Conformal LEC software infers a latch. The Quartus II software then adds an entry to the report panel in the Formal Verification folder of the Analysis & Synthesis report.

Altera recommends that you avoid latches in your design; however, if latches are necessary, Altera recommends using the LPM_LATCH megafunction.

For more information about latches, refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*.

Combinational Loops

If your design consists of an intended combinational loop, you must define an appropriate cut point for both the RTL and the post-fit **.vo** netlist. You can find a warning indicating that a combinational loop exists in your design in the **Formal Verification** subfolder of the Quartus II software Analysis & Synthesis report.

For more information about issues with combinational loops, refer to "Known Issues and Limitations" on page 17–16.

Finite State Machine Coding Styles

When the Conformal LEC software infers a state machine, the state machine uses sequential encoding as the default encoding in the absence of user encoding. The Quartus II software selects the encoding most suited for the inferred state machine if you set the **State Machine Processing** setting to the default value (**Auto**). To do this, follow these steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. In the **Category** list, select **Analysis & Synthesis Settings**. The **Analysis & Synthesis Settings** page appears.
3. Click **More Settings**. The **More Analysis & Synthesis Settings** dialog box appears.
4. Under **Existing Option Settings**, in the **Name** list, select **State Machine Processing**. In the **Setting** list, select **Auto**.
5. In the **More Analysis & Synthesis Settings** dialog box, click **OK**.
6. Click **OK**.

Use the coding style described in the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook* when writing finite state machines (FSMs). The coding style in the specified chapter allows Quartus II integrated synthesis and the Conformal LEC software to infer a similar state machine for the same RTL source code.

Black Boxes in the Conformal LEC Flow

The Quartus II software generates a flattened netlist; however, you must treat the following modules in your design as black boxes:

- LPMs and megafunctions without formal verification models
- Encrypted IP functions
- Entities not implemented in Verilog HDL or VHDL

To perform equivalence checking of a design between its version, which consists of the modules listed above and its implemented version, the Conformal LEC software must treat these modules as black boxes. To facilitate the formal verification flow, the Quartus II software reconstructs the hierarchy of the black boxes with a port interface that is identical to the module on the golden side of your design.

If your golden netlist (**.vqm** netlist from Synplify Pro or RTL) includes any design entity not having a corresponding formal verification model, the software treats that entity as a black box with its boundary interface preserved. Table 17–2 on page 17–9 lists three types of black boxes with corresponding required actions.

The Quartus II-generated **.vo** contains the black box hierarchy when you make an EDA Formal Verification Hierarchy assignment with the value `BLACKBOX`.

If you do not make this assignment for a module, the Quartus II software implements that module in logic cells. When this happens, the **.vo** netlist no longer contains the black box hierarchy and does not preserve the port interface, resulting in a mismatch in the Conformal LEC software.

Table 17–2. Black Boxes and Required Action

Type of Black Box	Required Action
Altera library of parameterized modules (LPMs) and megafunctions.	No action required. The Quartus II software automatically creates a black box list of components and preserves the hierarchy.
Any parameterized entity other than the parameterized entities listed in the *Guidelines for Creating a Design for Use with the Encounter Conformal and Quartus II Software* topic in Quartus II Help.	You must designate the wrapper that instantiates the parameterized entity as a black box.
Non-parameterized entities that you want to designate as a black box.	You can designate the entity itself as a black box.

You can also use the Quartus II GUI to set the black box property on the entities, which the formal verification tool does not compare.

To preserve the boundary interface of an entity using the GUI, make an EDA Formal Verification Hierarchy assignment to the entity with the value `BLACKBOX`.

Generating the Post-Fit Netlist Output File and the Conformal LEC Setup Files

The following steps describe how to set up the Quartus II software environment to generate the post-fit **.vo** netlist and the Conformal LEC script for use in formal verification. With the exception of step 2, the steps are identical for both of the synthesis tools:

To create a new Quartus II project or open an existing project, follow these steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.

2. In the **Category** list, click **EDA Tool Settings**.

   If you are using Quartus II integrated synthesis, follow these steps:

   a. In the **Category** list, under **EDA Tool Settings**, select **Design Entry/Synthesis**. Select **<None>** from the **Tool name** list.

   b. In the **Category** list, under **EDA Tool Settings**, select **Formal Verification**. Select **Conformal LEC** from the **Tool name** list.

   If you are using Synplify Pro, follow these steps:

   a. In the **Category** list, under **EDA Tool Settings**, select **Design Entry/Synthesis**. Select **Synplify Pro** from the **Tool name** list.

   b. In the **Category** list, under **EDA Tool Settings**, select **Formal Verification**. Select **Conformal LEC** from the **Tool name** list.

3. In the **Category** list, click **Incremental Compilation** under **Compilation Process Settings**. The **Incremental Compilation** page appears.

4. Type the following Tcl command in the Quartus II software Tcl console to turn on the incremental compilation feature:

```
set_global_assignment -name INCREMENTAL_COMPILATION FULL_INCREMENTAL_COMPILATION
```

   Altera recommends that you turn on the incremental compilation feature for formal verification, and that your design does not contain any partition that you created. The incremental compilation feature is on by default.

5. In the **Category** list, click **Physical Synthesis Optimizations** under **Compilation Process Settings**. The **Physical Synthesis Optimizations** page appears.

6. Turn off **Perform register retiming**.

   If you do not turn off **Perform register retiming**, an error occurs during compilation: “Physical Netlist Optimization Register retiming is not supported by Formal Verification tool Conformal LEC”.

7. Under **Optimize for fitting (physical synthesis for density)**, turn off **Perform physical synthesis for combinational logic** and **Perform logic to memory mapping** to prevent the software from mapping logic to RAMs.

   Retiming a design, either during the synthesis step or during the fitting step, usually results in moving and merging registers along the critical path and is not supported by the equivalence checking tools. Because equivalence checkers compare the logic cone terminating at registers, do not use retiming to move the registers during optimization in the Quartus II software.

For more information about physical synthesis, refer to the *Netlist Optimizations and Physical Synthesis* chapter in volume 2 of the *Quartus II Handbook*.

8. Perform a full compilation of your design. On the Processing menu, click **Start Compilation**, or click the **Start Compilation** icon on the toolbar.

Quartus II Software Generated Files, Formal Verification Scripts, and Directories

After successful compilation, the Quartus II software generates a list of files, formal verification scripts, and directories in the <*project_directory*>**/fv/conformal/** directory (Table 17–3).

Table 17–3. Quartus II Software Compiler-Generated Files and Directories

File or Directory	Name	Details
Script file	<*proj rev*>**.ctc**	The <*proj rev*>**.ctc** references <*proj rev*>**.clg** and <*proj rev*>**.clr** that read the library files and black box descriptions. The <*proj rev*>**.ctc** also references the <*proj rev*>**.cmc** containing information about the mapped points. Use the <*proj rev*>**.ctc** with the Conformal LEC software.
	<*proj rev*>**.cec**	The <*proj rev*>**.cec** contains information for instance equivalences.
	<*proj rev*>**.cep**	The <*proj rev*>**.cep** contains information for black box pin equivalences in your design.
	<*proj rev*>**.cmp**	The <*proj rev*>**.cmp** contains information for the black box pin mapping between the golden and revised sides. The Quartus II software calls the <*proj rev*>**.cmp** from the <*proj rev*>**.ctc** script file. By default, the line in which this file is called is commented out.
	<*proj rev*>**.cmc**	The <*proj rev*>**.cmc** contains information about the additional points that the Quartus II software maps in addition to the points that the tool selects.

	<*proj rev*>_**trivial.cmc**	This <*proj rev*>_**trivial.cmc** contains mapping information for all the key points in your design.
		Sometimes, the Conformal LEC software performs incorrect key point mapping, resulting in formal verification mismatches. To overcome the verification mismatches, the Quartus II software writes out the <*proj rev*>_**trivial.cmc** that contains mapping information for all the key points in your design. Reading this file during the formal verification setup can result in increased run time. Therefore, the Quartus II software writes out the top-level script file <*proj rev*>**.ctc** with the command to read the <*proj rev*>_**trivial.cmc** commented out. If the formal verification results are not acceptable, you can uncomment the command and read the <*proj rev*>_**trivial.cmc**. The command in the <*proj rev*>**.ctc** is:
		`//Trivial mappings with same name registers`
		`//read mapped points $PROJECT/fv/conformal/<proj rev>_trivial.cmc`

	<*proj rev*>**.clr**	The <*proj rev*>**.clr** contains information about the macros and libraries for the revised design.
	<*proj rev*>**.clg**	The <*proj rev*>**.clg** contains information about the macros and libraries for the golden design.
blackboxes directory	<*project directory*>**/fv/conformal/**<*project rev*>_**blackboxes**	This directory contains top-level module descriptions for all the user-defined black box entities and contains modules with definitions other than Verilog HDL or VHDL, for example, in your design directory <*project directory*>**/fv/conformal/**<*project rev*>_**blackboxes**
.vo netlist file	<*proj rev*>**.vo**	The Quartus II software-generated netlist for formal verification.

The script file contains the setup and constraints information to use with the formal verification tool. The <*entity*>**.v** in the **blackboxes** directory contains the module description of entities that you do not define in the formal verification library. The file also contains entities that you treat as black boxes. For example, if a reference to a black box for an instance of the ALTDPRAM megafunction in your design is present, the **blackboxes** directory does not contain a module description for the ALTDPRAM megafunction because you define the module description in the **altdpram.v** of the formal verification library. When a module does not have an RTL description, or the description exists only in the formal verification library and you do not want to compare the module with formal verification, a file containing only the top-level module description with port declaration is written out to the **blackboxes** directory and read into the Conformal LEC software. To learn more about black boxes, refer to "Black Boxes in the Conformal LEC Flow" on page 17–8.

Understanding the Formal Verification Scripts for the Conformal LEC Software

The Quartus II software generates scripts to use with the Conformal LEC software. This section describes the details of the Conformal LEC commands in the scripts to help you compare the revised netlist with the golden netlist. Usually, you do not have to add anymore Conformal LEC constraints to verify your netlists.

You can view a sample Quartus II software generated script in "Conformal Dofile/Script Example" on page 17–18.

Conformal LEC Commands in the Quartus II Software Generated Scripts

The value for the variable `QUARTUS` is the path to the Quartus II software installation directory:

```
setenv QUARTUS <Quartus Installation Directory>
```

The Quartus II software assigns the current working directory of your project to the `PROJECT` variable. Use this variable to change your project directory to the directory in which you install your design files when moving from a UNIX to a Windows environment, or vice versa:

```
setenv PROJECT <Quartus Project Directory>
```

The following command reads both the golden and the revised netlists, along with the appropriate library models:

```
read design <design files>
```

You must update your project location when you move the files from the Windows environment to the UNIX environment.

The post placement and routing netlist from the Quartus II software might contain net and instance names that are slightly different from net and instance names of the golden netlist. With the following command, the Quartus II software defines temporary substitute string patterns enabling the Conformal LEC software to map key points automatically when the names are different:

```
add renaming rule <rule>
```

The Conformal LEC software employs three name-based methods to map key points to compare the revised netlist with the golden netlist. Scripts set the correct method to get the best results.

```
set mapping method <mapping_rule>
```

The Quartus II software performs several optimizations, including optimizing the registers whose input is driven by a constant. Under these circumstances, for the formal verification software to compare the netlists properly, use the command `set flatten model` with the option `seq_constant`.

```
set flatten model <flattening_rule>
```

When you use the `report black box` command, verify that the software lists the following modules as black boxes, along with any of the modules that you treat as black boxes in the golden and revised netlists:

- LPMs and megafunctions without the formal verification models
- Encrypted IP functions
- Entities not implemented in Verilog HDL or VHDL

Use the following command to set the same implementation on multipliers for both the golden and revised netlists:

```
set multiplier implementation <implementation_name>
```

If combinational loops or instances of `LPM_LATCH` are present, the Quartus II software cuts the loop at the same point using the following command on both the golden and revised netlists:

```
add cut point
```

The Conformal LEC software does not always automatically map all the key points, or can incorrectly map some key points. To help the Conformal LEC software successfully complete the mapping process, the Quartus II software records optimizations performed on the netlist as a series of `add mapped points` in the Conformal LEC <*file_name*>**.cmc** script.

```
add mapped points <key_points>
```

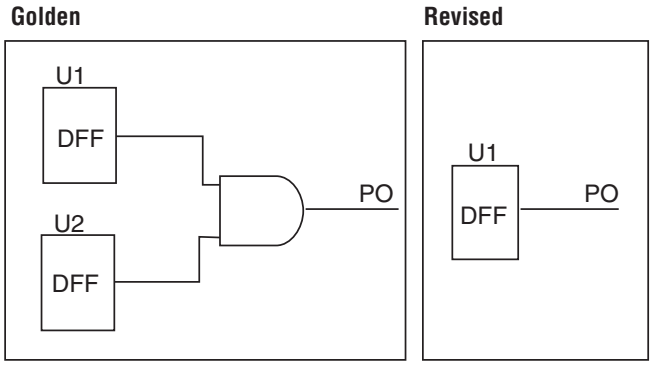
When the software moves the inverter before the register to after the register, use the following command:

```
add mapped points <key_points> -invert
```

The following command reads in the mapped point information from the specified file:

```
read mapped points <file_name>.cmc
```

Figure 17–3. Instance Equivalence



During optimization, the Quartus II software might merge two registers into one (Figure 17–3). The Quartus II software informs the formal verification tool that the U1 and U2 registers are equivalent to each other using the following command:

```
add instance equivalence <instance_pathname ..> [-golden]
```

When register duplication happens, use the following command:

```
add instance equivalence <instance_pathname ..> [-revised]
```

When the software moves the inverter beyond the register along with either register duplication or merging, use the following command:

```
add instance equivalences <instance_pathname>
[-invert <instance_pathname>]
```

Sometimes, the software drives the register output to a constant, either `logic 0` or `logic 1`. The Quartus II software sets the value of the register to a constraint using the `add instance constraint` command. For more information about this command, refer to "Fixed-Output Registers" on page 17–6.

```
add instance constraint <constraint_value>
```

Comparing Designs Using the Conformal LEC Software

This section describes using the Conformal LEC software to compare designs, and to prove logical equivalence between two versions of your design.

Running the Conformal LEC Software from the GUI

To run the Conformal LEC software from the GUI, follow these steps:

1. Open the Conformal LEC software.
2. On the File menu, click **Do Dofile**.
3. Select the <*path to project directory*>**/fv/conformal/**<*proj rev*>**.ctc**.

The Conformal LEC software GUI displays the comparison results. The Golden window displays the original RTL description or the post synthesis **.vqm** netlist from Synplify Pro, and the Revised window displays the information from the post-fit netlist generated by the Quartus II software. The message section at the bottom of the window reports the verification results and the number of unmapped and non-equivalent points found in your design.

To investigate the verification results, click the **Mapping Manager** icon in the toolbar, or on the Tools menu, click **Mapping Manager**. The Conformal LEC software reports the mapped, unmapped, and compared points in the **Mapped Points**, **Unmapped Points**, and **Compared Points** windows, respectively.

For more information about how to diagnose non-equivalent points, refer to the Conformal LEC software user documentation.

Running the Conformal LEC Software From a System Command Prompt

To run the Conformal LEC software without using the GUI, type the command shown in Example 17–5 at a system command prompt.

Example 17–5. Conformal LEC Command to Run Formal Verification

```
lec -dofile /<path to project directory>/fv/conformal/<proj rev>.ctc -nogui
```

To get a downloadable design example showing the formal verification flow with Quartus II software, refer to the Formal Verification Design Example page of the Altera website.

For more information about the latest debugging tips and solutions for formal verification flow between the Conformal LEC software and the Quartus II software, go to www.altera.com and perform an advanced search with keywords “formal verification”.

Known Issues and Limitations

The following known issues and limitations can occur when using the formal verification flow described in this chapter:

- AIn designs with combinational feedback loops, the Conformal LEC software can insert extra cut points in the revised netlist, causing unmapped points and ultimately verification mismatches.
- For Cyclone II designs, the Conformal LEC software might report non-equivalent flipflops and extra cut points for the revised (post-fit) design under the following conditions:
  - When your HDL source code instantiates the `lpm_ff` primitive with an asynchronous load signal `aload` (with or without any other asynchronous control signals) and;
  - When you use the asynchronous clear signal `aclr` and asynchronous set signal `aset` together.

  To avoid this problem, ensure that a wrapper module or entity is present around the `lpm_ff` instantiation, and black box the module or entity that instantiates the `lpm_ff` primitive.
- For Stratix III designs, the Conformal LEC software creates cut points for the combinational loops on the golden side and might fail equivalence checking due to improper mapping. The combinational loops are due to logic around the registers emulating multiple sets, resets, or both. The Quartus II software reports these cut points with warning messages during mapping. You can add Conformal LEC commands manually to add cut points, which can result in proper mapping and formal verification.
- To perform formal verification, the Quartus II software turns off certain synthesis optimization options (such as register retiming, optimization through black box hierarchy boundaries, and disabling the ROM and shift register inference), which can have an impact on the area resource and performance.

  
  In the Quartus II software version 9.0 and earlier, turning on gate-level register retiming as part of a formal verification flow might impact area and resource utilization.

- When you do not verify RAM and ROM instantiations, inferences, or both using formal verification.
- Incremental compilation for formal verification does not support user-created design partitions.
- Formal verification does not support clear box netlists due to unconnected ports on its WYSIWYG instances.
- Formal verification does not support VHDL megafunction variations due to undriven ports on the megafunctions.
- When a black box contains bidirectional ports, the Quartus II software does not reconstruct the hierarchy. Therefore, a flat netlist represents the black box, which results in formal verification mismatches.

- You must treat ROMs as black boxes in your design before compilation with Quartus II integrated synthesis, because the Quartus II software might perform some optimizations on the ROM, resulting in formal verification mismatches.
- The Conformal LEC software might report mismatches or cancel comparisons of some key points when the Quartus II software implements a DSP megafunction in LEs, due to implicit optimizations in the DSP and the complexity of the multiplier logic in terms of LEs.
- Unused logic optimized in and around a black box by the Quartus II software can result in a black-box interface different from the interface in the synthesized **.vqm** netlist.

Black Box Models

The black box models are interface definitions of entities, such as primitives, atoms, LPMs, and megafunctions. These models have a parameterized interface, and do not contain any definition of behavior. These models work with the Conformal LEC software, which uses these models along with your design to generate black boxes for instances of the entity with varying sets of parameters in your design.

For a complete list of supported black box models, refer to *Guidelines for Creating a Design for Use with the Encounter Conformal and Quartus II Software* in Quartus II Help.

Conformal Dofile/Script Example

Example 17–6 shows an example script, generated by the Quartus II software. The example script lists some of the setup commands in Conformal LEC software.

Example 17–6. Conformal LEC Script (Part 1 of 2)

```
// Copyright (C) 1991-2008 Altera Corporation
// Your use of Altera Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logi
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Altera Program License
// Subscription Agreement, Altera MegaCore Function License
// Agreement, or other applicable license agreement, including,
// without limitation, that your use is for the sole purpose of
// programming logic devices manufactured by Altera and sold by
// Altera or its authorized distributors. Please refer to the
// applicable agreement for further details.

// Script generated by the Quartus II software

reset
set system mode setup
set log file mfs_3prm_1a.fv.log -replace
set naming rule "%s" -register -golden
set naming rule "%s" -register -revised
// Naming rules for Verilog
set naming rule "%L.%s" "%L[%d].%s" "%s" -instance
set naming rule "%L.%s" "%L[%d].%s" "%s" -variable
// Naming rules for VHDL
// set naming rule "%L:%s" "%L:%d:%s" "%s" -instance
// set naming rule "%L:%s" "%L:%d:%s" "%s" -variable
// set undefined cell black_box -both
// These are the directives that are not supported by the QIS RTL to gates FV flow
set directive off verplex ambit
set directive off assertion_library black_box clock_hold compile_off compile_on
set directive off dc_script_begin dc_script_end divider enum infer_latch
set directive off mem_rowselect multi_port multiplier operand state_vector template
add notranslate module alt3pram -golden
add notranslate module alt3pram -revised
setenv QUARTUS /data/quark/build/ajaishan/quartus
setenv PROJECT /net/quark/build/ajaishan/quartus_regtest/eda/fv/conformal/synplify/
stratix/mfs_3prm_1a_v1_/mfs_3prm_1a/qu_allopt
```

Example 17–6. Conformal LEC Script (Part 2 of 2)

```
read design \
    $QUARTUS/eda/fv_lib/vhdl/dummy.vhd \
    -map lpm $QUARTUS/eda/fv_lib/vhdl/lpms \
    -map altera_mf $QUARTUS/eda/fv_lib/vhdl/mfs \
    -map stratix $QUARTUS/eda/fv_lib/vhdl/stratix \
    -vhdl -noelaborate -golden
read design \
    -file $PROJECT/fv/conformal/mfs_3prm_1a.clg \
    $PROJECT/p3rm_block.v \
    $PROJECT/mfs_3prm_1a.v \
    -verilog2k -merge none -golden
read design \
    $QUARTUS/eda/fv_lib/vhdl/dummy.vhd \
    -map lpm $QUARTUS/eda/fv_lib/vhdl/lpms \
    -map altera_mf $QUARTUS/eda/fv_lib/vhdl/mfs \
    -map stratix $QUARTUS/eda/fv_lib/vhdl/stratix \
    -vhdl -noelaborate -revised
read design \
    -file $PROJECT/fv/conformal/mfs_3prm_1a.clr \
    $PROJECT/fv/conformal/mfs_3prm_1a.vo \
    -verilog2k -merge none -revised
// add ignored inputs _unassoc_inputs_* -all -revised
add renaming rule r1 "~I\/" "\/" -revised
add renaming rule r2 "_I\/" "\/" -revised
set multiplier implementation rca -golden
set multiplier implementation rca -revised
set mapping method -name first
set mapping method -nounreach
set mapping method -noreport_unreach
set mapping method -nobbox_name_match
set flatten model -seq_constant
set flatten model -nodff_to_dlat_zero
set flatten model -nodff_to_dlat_feedback
set flatten model -nooutput_z
set root module mfs_3prm_1a -golden
set root module mfs_3prm_1a -revised
report messages
report black box
report design data
// report floating signals
dofile $PROJECT/fv/conformal/mfs_3prm_1a.cec
// dofile $PROJECT/fv/conformal/mfs_3prm_1a.cep
// Instance-constraints commands for constant-value registers removed
// during compilation
set system mode lec -nomap
read mapped points $PROJECT/fv/conformal/mfs_3prm_1a.cmc


// Trivial mappings with same name registers
// read mapped points $PROJECT/fv/conformal/mfs_3prm_1a_trivial.cmc
// dofile $PROJECT/fv/conformal/mfs_3prm_1a.cmp
map key points
remodel -seq_constant -repeat
add compare points -all
compare
usage
// exit -f
```

Conclusion

Formal verification software enables verification of your design during all stages, from RTL to placement and routing. Verifying designs requires more time as designs increase in size. Formal verification helps to reduce the time needed for your design verification cycle.

Document Revision History

Table 17–4 lists the revision history for this chapter.

Table 17–4. Document Revision History

Date	Version	Changes
November 2013	13.1.0	■ Removed HardCopy device information.
June 2012	12.0.0	■ Removed survey link.
November 2011	11.1.0	■ Updated "Black Boxes in the Conformal LEC Flow" on page 17–8 and "Known Issues and Limitations" on page 17–16. ■ Removed Figures.
December 2010	10.1.0	Changed to new document template. Removed Table 21-1.
July 2010	10.0.0	Updates for new GUI changes, and added link to Help.
November 2009	9.1.0	Updated "Black Boxes in the Encounter Conformal Flow" section.
March 2009	9.0.0	Updated Table 21-1.
November 2008	8.1.0	■ Changed to 8-1/2 x 11 page size. ■ Added support for Stratix IV devices. ■ Added support for Cadence Conformal LEC version 7.2 and Synplify Pro version 9.6.2.
May 2008	8.0.0	■ Added support for Cyclone III devices. ■ Updated "Black Boxes in the Encounter Conformal Flow" section. ■ Updated Table 18–1 and Table 18–5.

For previous versions of the *Quartus II Handbook*, refer to the Quartus II Handbook Archive.