# Quartus II Handbook, Volume 2

## Design Implementation & Optimization

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com

**qii5v2-2.1**

# Contents

# Chapter 4. Quartus II Project Management

# Section II. Device & Board Utilities

# Chapter 5. I/O Assignment Planning & Analysis

## Section III. Area Optimization & Timing Closure

# Chapter Revision Dates

The chapters in this book, *Quartus II Handbook, Volume 2*, were revised on the following dates. Where chapters or groups of chapters are available separately, part numbers are listed.

| | | |
|---|---|---|
| Chapter 1. | Assignment Editor | |
| | Revised: | *June 2004* |
| | Part number: | *qii52001-2.0* |
| Chapter 2. | Command-Line Scripting | |
| | Revised: | *June 2004* |
| | Part number: | *qii52002-2.0* |
| Chapter 3. | Tcl Scripting | |
| | Revised: | *August 2004* |
| | Part number: | *qii52003-2.0* |
| Chapter 4. | Quartus II Project Management | |
| | Revised: | *June 2004* |
| | Part number: | *qii52012-1.0* |
| Chapter 5. | I/O Assignment Planning & Analysis | |
| | Revised: | *June 2004* |
| | Part number: | *qii52004-2.0* |
| Chapter 6. | Design Optimization for Altera Devices | |
| | Revised: | *June 2004* |
| | Part number: | *qii52005-2.0* |
| Chapter 7. | Timing Closure Floorplan | |
| | Revised: | *June 2004* |
| | Part number: | *qii52006-2.0* |
| Chapter 8. | Netlist Optimizations and Physical Synthesis | |
| | Revised: | *June 2004* |
| | Part number: | *qii52007-2.0* |
| Chapter 9. | Design Space Explorer | |
| | Revised: | *June 2004* |
| | Part number: | *qii52008-2.0* |

|  |  |  |
|---|---|---|
| Chapter 10. | LogicLock Design Methodology |  |
|  | Revised: | *August 2004* |
|  | Part number: | *qii52009-2.1* |
| Chapter 11. | Timing Closure in HardCopy Devices |  |
|  | Revised: | *June 2004* |
|  | Part number: | *qii52010-2.0* |
| Chapter 12. | Synplicity Amplify Physical Synthesis Support |  |
|  | Revised: | *February 2004* |
|  | Part number: | *qii52011-1.0* |

# About this Handbook

This handbook provides comprehensive information about the Altera® Quartus®II design software, version 4.1.

## How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide web site at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

| Information Type | USA & Canada | All Other Locations |
|---|---|---|
| Technical support | www.altera.com/mysupport/ | altera.com/mysupport/ |
| | (800) 800-EPLD (3753)<br>(7:00 a.m. to 5:00 p.m. Pacific Time) | (408) 544-7000 *(1)*<br>(7:00 a.m. to 5:00 p.m. Pacific Time) |
| Product literature | www.altera.com | www.altera.com |
| Altera literature services | lit_req@altera.com *(1)* | lit_req@altera.com *(1)* |
| Non-technical customer service | (800) 767-3753 | (408) 544-7000<br>(7:30 a.m. to 5:30 p.m. Pacific Time) |
| FTP site | ftp.altera.com | ftp.altera.com |

***Note to table:***
(1) You can also contact your local Altera sales office or sales representative.

## Typographic Conventions

This document uses the typographic conventions shown below.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box. |
| **bold type** | External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: **f**$_{\mathbf{MAX}}$, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.* |

| Visual Cue | Meaning |
|---|---|
| *Italic type* | Internal timing parameters and variables are shown in italic type. Examples: $t_{PIA}$, $n + 1$.<br><br>Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>*.**pof** file. |
| Initial Capital Letters | Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu. |
| "Subheading Title" | References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: "Typographic Conventions." |
| `Courier type` | Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input`. Active-low signals are denoted by suffix `n`, e.g., `resetn`.<br><br>Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier. |
| 1., 2., 3., and<br>a., b., c., etc. | Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| ■ ● • | Bullets are used in a list of items when the sequence of the items is not important. |
| ✓ | The checkmark indicates a procedure that consists of one step only. |
| ☞ | The hand points to information that requires special attention. |
| ⚠ CAUTION | The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process. |
| ⚠ | The warning indicates information that should be read prior to starting or continuing the procedure or processes |
| ↵ | The angled arrow indicates you should press the Enter key. |
| 👣 | The feet direct you to more information on a particular topic. |

# Section I. Scripting & Constraint Entry

As a result of the increasing complexity of today's FPGA designs and the demand for higher performance, designers must make a large number of complex timing and logic constraints to meet their performance requirements. Once you have created a project and your design, you can use the the Quartus® II software Assignment Editor and Floorplan Editor to specify your initial design constraints, such as pin assignments, device options, logic options, and timing constraints.

This section describes how to take advantage of these components of the Quartus II software, how to take advantage of Quartus II modular executables, and how to develop and run tool command language (Tcl) scripts to perform a wide range of functions.

This section includes the following chapters:

## Revision History

The table below shows the revision history for Chapters 1 to 4.

| Chapter(s) | Date / Version | Changes Made |
|---|---|---|
| 1 | June 2004 v2.0 | ● Updates to tables, figures.<br>● New functionality in the Quartus software version 4.1. |
|  | Feb. 2004 v1.0 | Initial release. |
| 2 | June 2004 v2.0 | ● Updates to tables, figures.<br>● New functionality in the Quartus software version 4.1. |
|  | Feb. 2004 v1.0 | Initial release. |
| 3 | Aug. 2004 v2.1 | ● Minor typographical corrections<br>● Enhancements to example scripts. |
|  | June 2004 v2.0 | ● Updates to tables, figures.<br>● New functionality in the Quartus software version 4.1. |
|  | Feb. 2004 v1.0 | Initial release. |
| 4 | June 2004 v1.0 | Initial release. |

# 1. Assignment Editor

qii52001-2.0

## Introduction

As a result of the increasing complexity of today's FPGA designs and the demand for higher performance, designers must make a larger number of complex timing and logic constraints to meet their performance requirements. This complexity is compounded by the increasing density and associated pin counts of current FPGAs. To successfully implement a complex design in the latest generation of FPGAs, designers must also make a large number of pin assignments that include the pin locations and I/O standards.

To facilitate the process of entering these assignments, Altera® has developed an intuitive, spreadsheet interface called the Assignment Editor. The Assignment Editor is designed to make the process of creating, changing, and managing a large number of assignments as easy as possible.

This chapter discusses the following topics:

- Using the Assignment Editor
- Effects of settings made outside the Assignment Editor user interface
- Category, node filter, information, edit bars and spreadsheet
- Integration with other Quartus® II features
- Enhanced spreadsheet interface
- Dynamic Syntax checker
- Node Filter bar
- Using Time Groups
- Customizable columns
- Tcl interface
- Exporting Assignments
- Importing Assignments

## Using the Assignment Editor

You can use the Assignment Editor throughout the design cycle. Before board layout begins, you can make pin assignments with the Assignment Editor. Throughout the design cycle, you can use the Assignment Editor to help achieve your design performance requirements by making timing assignments. You can also use the Assignment Editor to view, filter, and sort assignments based on node names or assignment type.

The Assignment Editor is a resizable and minimizable window. This scalability makes it easy to view or edit your assignments right next to your design files. You can launch the Assignment Editor from the Assignments menu or by clicking on the Assignment Editor icon in the toolbar.

## Effects of Settings Made Outside the Assignment Editor User Interface

Although the Assignment Editor is the most common method of entering and modifying assignments, there are other methods you can use to make and edit assignments. For this reason, the Assignment Editor updates itself if you add, remove or change an assignment outside the Assignment Editor.

The Assignment Editor is refreshed each time you click anywhere in the window. If you make an assignment in the Quartus II software, such as in the Tcl console or in the Floorplan Editor, the Assignment Editor reloads the new assignments from memory. If you modify the Quartus II Settings File (**.qsf**) outside the Quartus II software and you select the Assignment Editor window, the Assignment Editor reloads the QSF.

☞ If the QSF is being edited while the project is open, Altera recommends that you perform a Save Project (File menu) to ensure that you are editing the latest QSF file.

In either case, the **Messages** window displays the following message:

```
Info: Assignments reloaded -- assignments updated
outside Assignment Editor
```

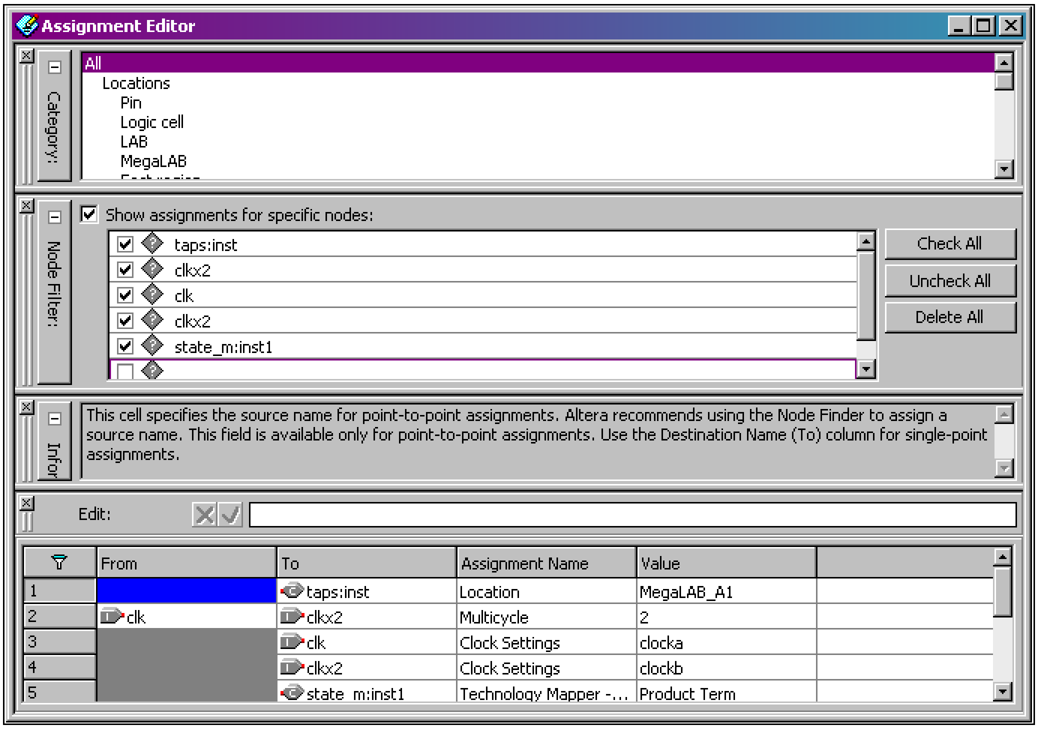
The assignments you make in the Assignment Editor, Floorplan Editor, or with Tcl are stored in memory. Use one of the following commands to write these assignments into the QSF:

- **Close Project** (File menu)
- **Save Project** (File menu)
- **Start Compilation** (Processing menu)

## Category, Node Filter, Information, Edit Bars & Spreadsheet

The Assignment Editor window is divided into four bars and a spreadsheet: see Figure 1–1. You can hide all four bars in the View menu if desired, and you can collapse the **Category**, **Node Filter**, and **Information** bars. Table 1–1 provides a brief description of each bar.

*Figure 1–1. The Assignment Editor Window*


Assignment Editor

Category:

All
Locations
Pin
Logic cell
LAB
MegaLAB

Node Filter:

☑ Show assignments for specific nodes:

☑ taps:inst
☑ clkx2
☑ clk
☑ clkx2
☑ state_m:inst1
☐

Check All
Uncheck All
Delete All

Infor

This cell specifies the source name for point-to-point assignments. Altera recommends using the Node Finder to assign a source name. This field is available only for point-to-point assignments. Use the Destination Name (To) column for single-point assignments.

Edit:

| | From | To | Assignment Name | Value | |
|---|---|---|---|---|---|
| 1 | | taps:inst | Location | MegaLAB_A1 | |
| 2 | clk | clkx2 | Multicycle | 2 | |
| 3 | | clk | Clock Settings | clocka | |
| 4 | | clkx2 | Clock Settings | clockb | |
| 5 | | state_m:inst1 | Technology Mapper -... | Product Term | |

*Table 1–1. Assignment Editor Bar Descriptions*

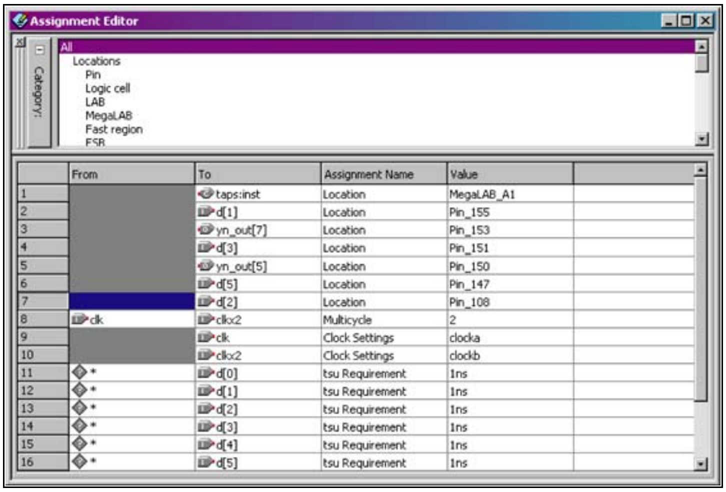
| Bar Name | Description |
|---|---|
| Category | Filters the type of available assignments |
| Node Filter | Filters a selection of design nodes to be viewed or assigned |
| Information | Displays a description of the cell currently selected |
| Edit | Allows you to edit the text in the currently selected cell(s) |

## Category Bar

The **Category** bar lists all assignment categories available for the chosen device. You can use the **Category** bar to select a particular assignment type and to filter out all other assignments. Selecting an assignment category from the Category list changes the spreadsheet to show only applicable options and values. To search for a particular type of assignment, use the **Category** bar to filter out all other assignments.

To view all $t_{SU}$ assignments in your project, select **tsu** in the category list. If you select **All** in the **Category** bar, the Assignment Editor displays all assignments. See Figures 1–2 and 1–3 below.

*Figure 1–2. All Selected in the Category List*

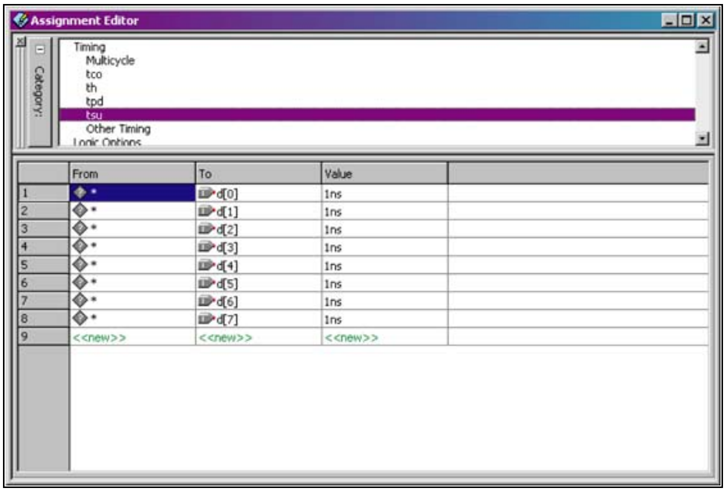


Assignment Editor

Category:
All
Locations
Pin
Logic cell
LAB
MegaLAB
Fast region
ESB

| | From | To | Assignment Name | Value |
|---|---|---|---|---|
| 1 | | taps:inst | Location | MegaLAB_A1 |
| 2 | | d[1] | Location | Pin_155 |
| 3 | | yn_out[7] | Location | Pin_153 |
| 4 | | d[3] | Location | Pin_151 |
| 5 | | yn_out[5] | Location | Pin_150 |
| 6 | | d[5] | Location | Pin_147 |
| 7 | | d[2] | Location | Pin_108 |
| 8 | clk | clkx2 | Multicycle | 2 |
| 9 | | clk | Clock Settings | clocka |
| 10 | | clkx2 | Clock Settings | clockb |
| 11 | * | d[0] | tsu Requirement | 1ns |
| 12 | * | d[1] | tsu Requirement | 1ns |
| 13 | * | d[2] | tsu Requirement | 1ns |
| 14 | * | d[3] | tsu Requirement | 1ns |
| 15 | * | d[4] | tsu Requirement | 1ns |
| 16 | * | d[5] | tsu Requirement | 1ns |

*Figure 1–3. $t_{SU}$ Selected in Category List*



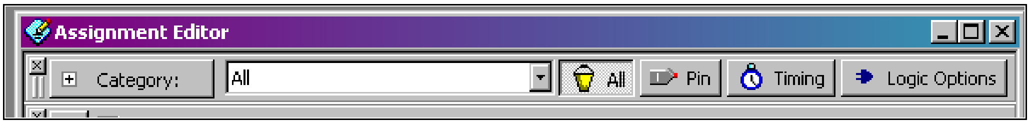
Assignment Editor

Category:
Timing
Multicycle
tco
th
tpd
tsu
Other Timing
Logic Options

| | From | To | Value |
|---|---|---|---|
| 1 | * | d[0] | 1ns |
| 2 | * | d[1] | 1ns |
| 3 | * | d[2] | 1ns |
| 4 | * | d[3] | 1ns |
| 5 | * | d[4] | 1ns |
| 6 | * | d[5] | 1ns |
| 7 | * | d[6] | 1ns |
| 8 | * | d[7] | 1ns |
| 9 | <<new>> | <<new>> | <<new>> |

When you collapse the **Category** bar, four shortcut buttons appear so you can select between various preset categories (see Figure 1–4).

*Figure 1–4. Category Bar*

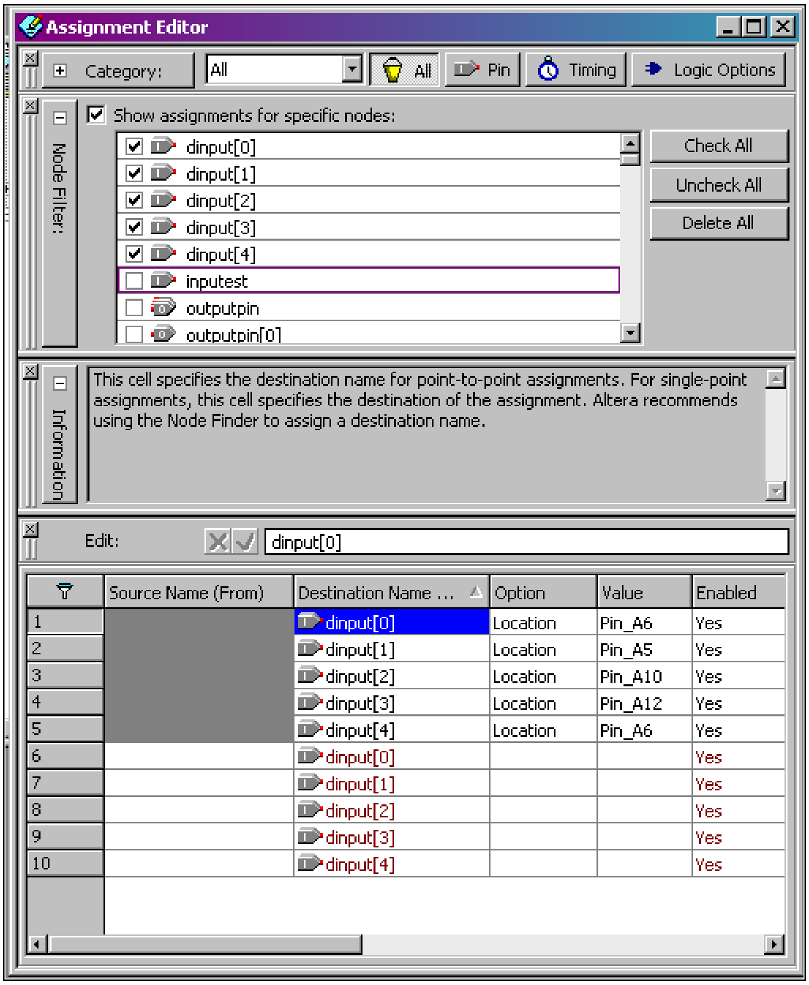


## Node Filter Bar

When **Show assignments for specific nodes** is turned on, the spreadsheet shows only assignments for nodes matching the selected node name filters in the **Node Filter** bar. You can selectively enable individual node name filters listed in the **Node Filter** bar. You can create a new node name filter by selecting a node name with the **Node Finder** or typing a new node name filter. The Assignment Editor automatically inserts a spreadsheet row and prepopulates the **To** field with the node name filter. You can easily add an assignment to the matching nodes by entering it in the new row. Rows with incomplete assignments appear in dark red. When you choose **Save** (File menu), all incomplete rows are removed and a message issued.

In Figure 1–5, when selecting all the bits of the `dinput` bus, all unrelated assignments are filtered out.

*Figure 1–5. Using the Node Filter in the Assignment Editor*



Show assignments for specific nodes:

- [✓] dinput[0]
- [✓] dinput[1]
- [✓] dinput[2]
- [✓] dinput[3]
- [✓] dinput[4]
- [ ] inputest
- [ ] outputpin
- [ ] outputpin[0]

This cell specifies the destination name for point-to-point assignments. For single-point assignments, this cell specifies the destination of the assignment. Altera recommends using the Node Finder to assign a destination name.

Edit: dinput[0]

| | Source Name (From) | Destination Name ... | Option | Value | Enabled |
|---|---|---|---|---|---|
| 1 | | dinput[0] | Location | Pin_A6 | Yes |
| 2 | | dinput[1] | Location | Pin_A5 | Yes |
| 3 | | dinput[2] | Location | Pin_A10 | Yes |
| 4 | | dinput[3] | Location | Pin_A12 | Yes |
| 5 | | dinput[4] | Location | Pin_A6 | Yes |
| 6 | | dinput[0] | | | Yes |
| 7 | | dinput[1] | | | Yes |
| 8 | | dinput[2] | | | Yes |
| 9 | | dinput[3] | | | Yes |
| 10 | | dinput[4] | | | Yes |

## Information Bar

The **Information** bar provides a brief description of the currently selected cell. This is a useful way for you to learn how to enter node names and assignments into the spreadsheet. For example, if the selected cell is a particular logic option, the **Information** bar shows a description of that option.

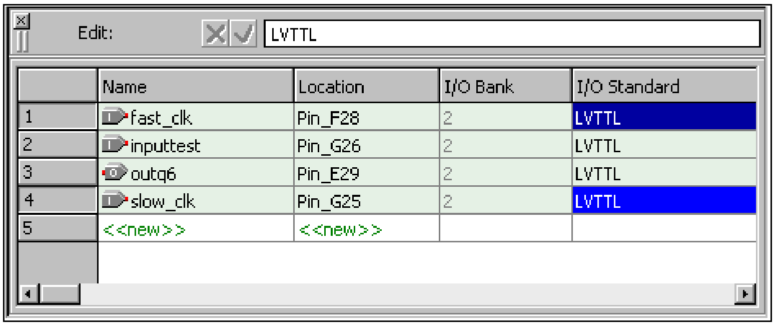
For more information on logic options, see Quartus II Help.

## Edit Bar

The **Edit** bar is an efficient way to enter a value into one or more spreadsheet cells.

To change the contents of multiple cells at the same time, select the cells in the spreadsheet (see Figure 1–6), then type the new value into the Edit box in the **Edit** bar (see Figure 1–7) and click the checkmark icon (Accept).

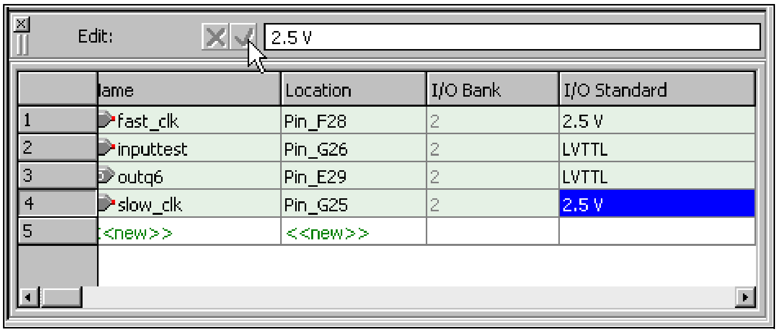
*Figure 1–6. Edit Bar Selection*



Edit: LVTTL

| | Name | Location | I/O Bank | I/O Standard |
|---|---|---|---|---|
| 1 | fast_clk | Pin_F28 | 2 | LVTTL |
| 2 | inputtest | Pin_G26 | 2 | LVTTL |
| 3 | outq6 | Pin_E29 | 2 | LVTTL |
| 4 | slow_clk | Pin_G25 | 2 | LVTTL |
| 5 | <<new>> | <<new>> | | |

*Figure 1–7. Edit Bar Change*



Edit: 2.5 V

| | lame | Location | I/O Bank | I/O Standard |
|---|---|---|---|---|
| 1 | fast_clk | Pin_F28 | 2 | 2.5 V |
| 2 | inputtest | Pin_G26 | 2 | LVTTL |
| 3 | outq6 | Pin_E29 | 2 | LVTTL |
| 4 | slow_clk | Pin_G25 | 2 | 2.5 V |
| 5 | <new>> | <<new>> | | |

## Assignment Editor Features

You can open the Assignment Editor from many locations, including the Text Editor, the Node Finder, the Timing Closure Floorplan, the Compilation Report, and the Messages window. For example, you can highlight a node name in your design file and open the Assignment Editor to cause your node name to appear in the Assignment Editor.

You can also open other windows from the Assignment Editor. From a node listed in the Assignment Editor spreadsheet, you can go to the location of the node in any of the following windows: Timing Closure Floorplan, Last Compilation Floorplan, Chip Editor, Block Editor, and Text Editor.

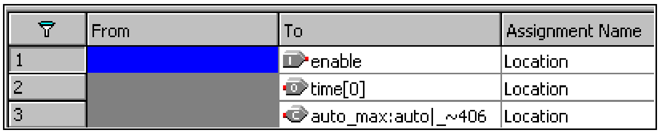
### Using the Enhanced Spreadsheet Interface

One of the key features of the Assignment Editor is the spreadsheet interface. With the spreadsheet interface, you can sort columns, use pull-down entry boxes, and copy and paste multiple cells in the Assignment Editor. As you enter an assignment, the font color of the row changes to indicate the status of the assignment.

☞ See the "Dynamic Syntax Checking" on page 1–9 for more information.

There are many ways to select or enter nodes into the spreadsheet, including: the Node Finder, the **Node Filter** bar, the **Edit** bar, or by directly typing the node name into the cell in the spreadsheet. A node type icon appears beside each node name and node name filter to identify its type. The node type icon identifies the entry as an input, output, or bidirectional pin, a register, or combinational logic. See Figure 1–8. The node type icon appears as an asterisk for node names and node name filters that use a wildcard character (* or ?).

*Figure 1–8. Node Type Icon Displayed Beside Each Node Name in the Spreadsheet*



| | From | To | Assignment Name |
|---|---|---|---|
| 1 | | enable | Location |
| 2 | | time[0] | Location |
| 3 | | auto_max:auto\|_~406 | Location |

The Assignment Editor supports wildcards in the following types of assignments:

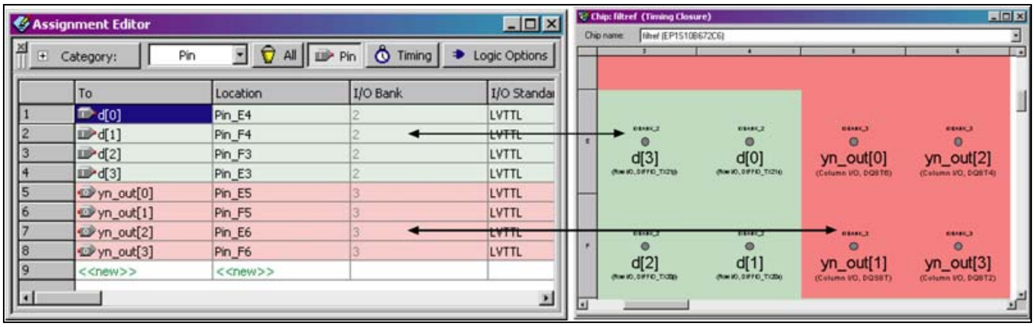
- All timing assignments
- Point-to-point global signal assignments (applicable to Stratix and Stratix II families)

- Point-to-point or pad-to-core delay chain assignments
- LogicLock region assignments

The spreadsheet also supports customizable columns, (see "Customizable Columns" on page 1–12), allowing you to show, hide, and arrange the columns.

When making pin location assignments, the background color of the cells coordinates with the color of the I/O bank also shown in the Floorplan Editor (see Figure 1–9).

*Figure 1–9. Spreadsheet-Like Interface*



| | To | Location | I/O Bank | I/O Standa |
|---|---|---|---|---|
| 1 | d[0] | Pin_E4 | 2 | LVTTL |
| 2 | d[1] | Pin_F4 | 2 | LVTTL |
| 3 | d[2] | Pin_F3 | 2 | LVTTL |
| 4 | d[3] | Pin_E3 | 2 | LVTTL |
| 5 | yn_out[0] | Pin_E5 | 3 | LVTTL |
| 6 | yn_out[1] | Pin_F5 | 3 | LVTTL |
| 7 | yn_out[2] | Pin_E6 | 3 | LVTTL |
| 8 | yn_out[3] | Pin_F6 | 3 | LVTTL |
| 9 | <<new>> | <<new>> | | |

Auto-fill pin names are supported in the spreadsheet if you have performed analysis and synthesis. Auto-fill pin locations are also supported in the spreadsheet if **Pin** is selected in the **Category** bar.

## Dynamic Syntax Checking

As you enter your assignments, the Assignment Editor performs simple legality and syntax checks. This checking is not as thorough as the checks performed during compilation, but it catches general incorrect settings. For example, the Assignment Editor does not allow assignment of a pin to a no-connect pin. In this case, the assignment is not accepted and you must enter a different pin location.

The color of the text in each row indicates if the assignment is incomplete, incorrect, or disabled (see Table 1–2 on page 1–10). You can customize the colors in the **Options** dialog box (Tools menu).

For more information, see the Quartus II Help.

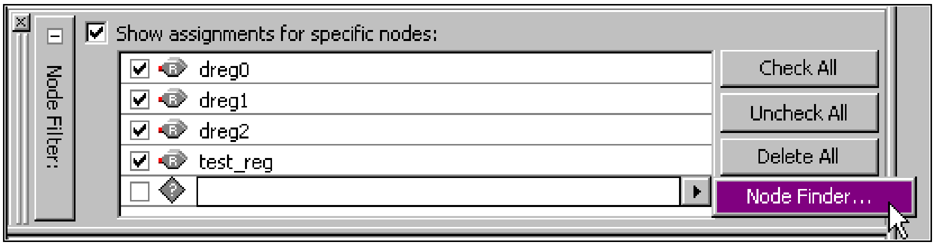
*Table 1–2. Description of the Text Color in the Spreadsheet*

| Text Color | Description |
|---|---|
| Green | A new assignment can be created |
| Yellow | The assignment contains warnings, such as an unknown node name |
| Dark Red | The assignment is incomplete |
| Bright Red | The assignment has an error, such as an illegal value |
| Light Gray | The assignment is disabled |

## Node Filter Bar

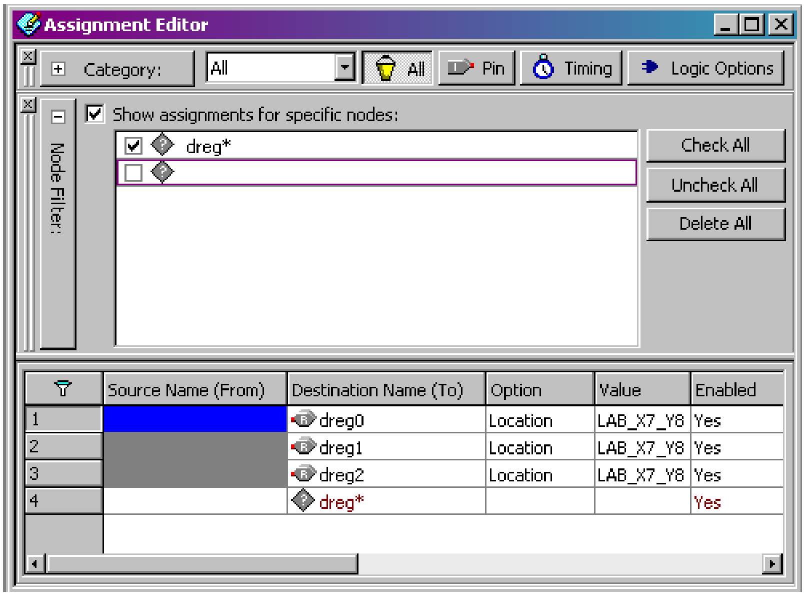
The **Node Filter** bar provides flexibility in how you view and make your settings. The **Node Filter** bar contains a list of node filters. To create a new entry, use the Node Finder or manually type the node name. Double-click an empty row in the **Node Filter** list and then click on the arrow to open the **Node Finder** (see Figure 1–10).

*Figure 1–10. Node Finder Option*



In the **Node Filter** bar, you can turn each filter on or off. To turn off the **Node Filter** bar, turn off **Show assignments for specific nodes**. The wildcards (* and ?) can be used to filter for a selection of all the design nodes with one entry in the Node Filter. For example, you can enter `dreg*` into the Node Filter list to view all assignments for dreg0, dreg1, and dreg2 (see Figure 1–11).

*Figure 1–11. Using the Node Filter Bar with Wildcards*



| | Source Name (From) | Destination Name (To) | Option | Value | Enabled |
|---|---|---|---|---|---|
| 1 | | dreg0 | Location | LAB_X7_Y8 | Yes |
| 2 | | dreg1 | Location | LAB_X7_Y8 | Yes |
| 3 | | dreg2 | Location | LAB_X7_Y8 | Yes |
| 4 | | dreg* | | | Yes |

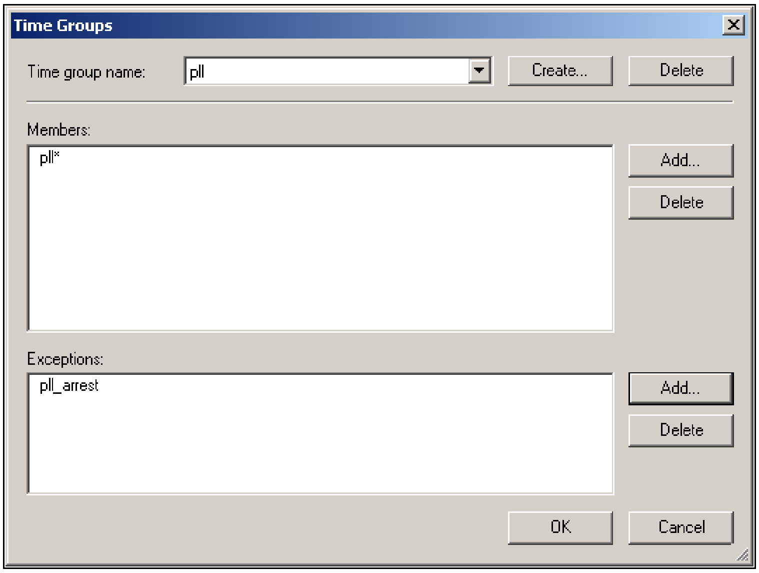
## Using Time Groups

A time group is a collection of design nodes grouped together and represented as a single unit for the purpose of making timing assignments to the collection. Using time groups with the Assignment Editor provides the flexibility required for complex timing assignments to a large number of nodes.

To create a time group, open the **Time Groups** dialog box by selecting **Time Groups** (Assignments menu). You can add and exclude members of each time group with wild cards in the Node Finder (See Figure 1–12 on page 1–12).

There are cases when wild cards are not flexible enough to select a large number of nodes that have node names that are quite similar. With time groups you can combine wild cards, which select a large number of nodes, and use exceptions to remove nodes that you did not intend to select.

*Figure 1–12. Time Groups Dialog Box*



Time group name: pll

Members: pll*

Exceptions: pll_arrest

## Customizable Columns

To provide more control over the display of information in the spreadsheet, the Assignment Editor supports customizable columns.

You can move columns, sort them in ascending or descending order, show or hide individual columns, as well as align (left, center, or right) the content in the column for improved readability.

When the Quartus II software starts for the first time, you see a pre-selected set of columns. However, you can show or hide any of the available columns by choosing the **Customize Columns** command (View menu). When you restart the Quartus II software, the column settings are maintained.

For example, the **Comments** and **Enabled** columns are hidden when the Quartus II software is first started.

You can use the **Comments** column to document the purpose of a pin or to explain why you applied a timing or logic constraint. You can use the **Enabled** column to disable any assignment without deleting it. This feature is useful when performing multiple compilations with different timing constraints or logic optimizations.

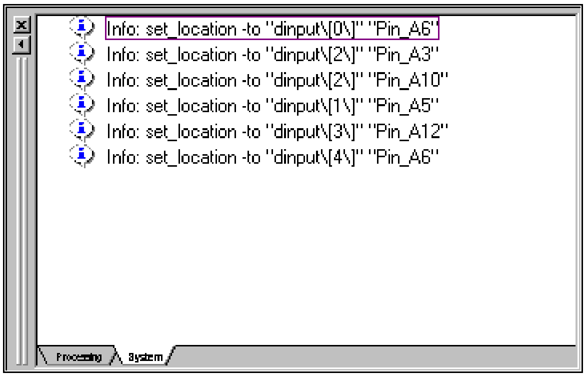
### Tcl Interface

Whether you use the Assignment Editor or another feature to create your design's assignments, you can export them all to a Tcl file. You can then use the Tcl file to re-apply all the settings or to archive your assignments. Choose **Export** (File menu) to export your assignments to a Tcl script.

You can also choose the **Generate Tcl File for Project** (Project menu) to generate a Tcl script file for your project.

In addition, as you use the Assignment Editor to enter assignments, the equivalent Tcl commands are shown in the system message window. You can use these Tcl commands to create customized Tcl scripts (see Figure 1–13). To copy a Tcl command from the Messages window, right-click the message and choose **Copy** (right button pop-up menu).

*Figure 1–13. Equivalent Tcl Commands Displayed in the Messages Window*



```
Info: set_location -to "dinput\[0\]" "Pin_A6"
Info: set_location -to "dinput\[2\]" "Pin_A3"
Info: set_location -to "dinput\[2\]" "Pin_A10"
Info: set_location -to "dinput\[1\]" "Pin_A5"
Info: set_location -to "dinput\[3\]" "Pin_A12"
Info: set_location -to "dinput\[4\]" "Pin_A6"
```

For more information on Tcl scripting with the Quartus II software, see the *Tcl Scripting* chapter in Volume 2 of the *Quartus II Handbook.*

## Exporting and Importing Assignments

With the **Export Assignments** and **Import Assignments** dialog boxes, you can export your Quartus II assignments to a Quartus II Settings file (**.qsf**), and import assignments from a **.qsf**, a Quartus II Entity Settings file (**.esf**), a MAX+PLUS II Assignment and Configuration file (**.acf**), or a Comma Separated Value file (**.csv**).

In addition to the **Export Assignments** and **Import Assignments** dialog boxes, the **Export** command (File menu) allows you to export your assignments to a **.csv** or Tcl script file (.tcl).

The **Export** command (File menu) exports the contents of the active window in the Quartus II software to another file format, when applicable.

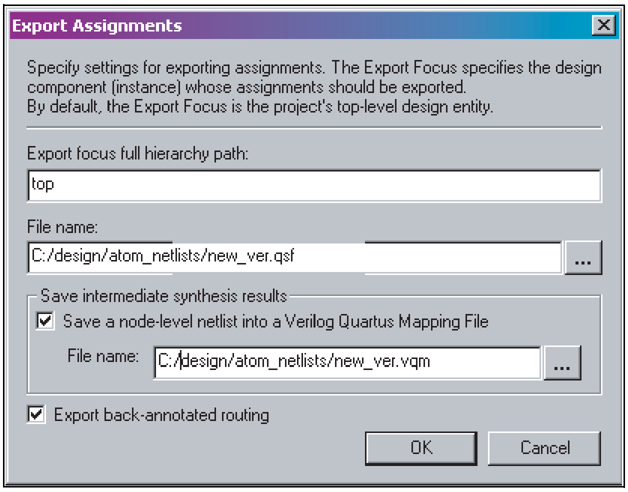
You can use these file formats for many different aspects of your project. For example, you can use a **.csv** file for documentation purposes or to transfer pin-related information to board layout tools. The Tcl file makes it easy to apply assignments in a scripted design flow. And the LogicLock design flow uses the **.qsf** file to transfer your LogicLock region settings.

## Exporting Assignments

The **Export Assignments** dialog box is used to export your Quartus II software assignments into a **.qsf** file, generate a node-level netlist file, and export back-annotated routing information as a Routing Constraints File (**.rcf**), as shown in Figure 1–14. Choose **Export Assignments** (Assignments menu) to open the **Export Assignments** dialog box. The LogicLock design flow uses this dialog box to export LogicLock regions.

For more information on using the **Export Assignments** dialog box to export LogicLock regions, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook*.

***Figure 1–14. Export Assignments Dialog Box***



Specify settings for exporting assignments. The Export Focus specifies the design component (instance) whose assignments should be exported.
By default, the Export Focus is the project's top-level design entity.

Export focus full hierarchy path: top

File name: C:/design/atom_netlists/new_ver.qsf

Save intermediate synthesis results

[✓] Save a node-level netlist into a Verilog Quartus Mapping File

File name: C:/design/atom_netlists/new_ver.vqm

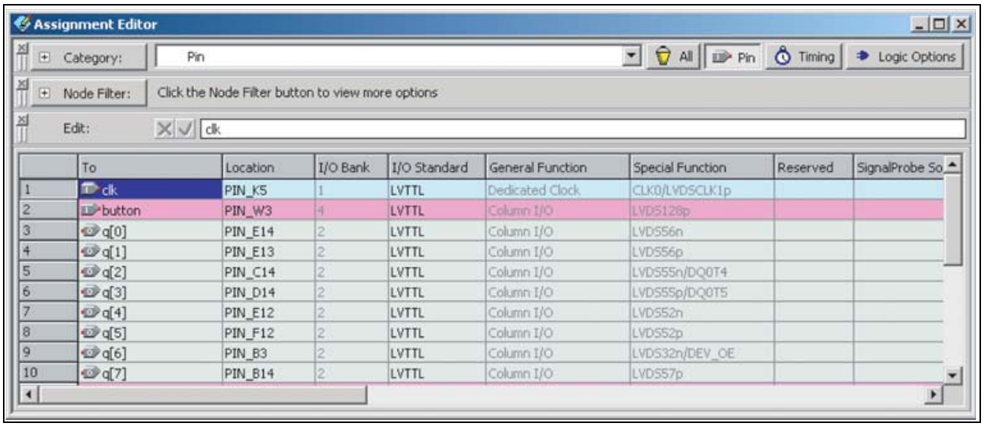
[✓] Export back-annotated routing

You can use the **Export** command (File menu) to export all assignments to a Tcl file or export a set of assignments to a **.csv** file. When you export assignments to a Tcl file, only user-created assignments are written to the Tcl script file, and default assignments are not exported.

When assignments are exported to a **.csv** file, only the assignments displayed in the current view of the Assignment Editor are exported. For example, to export only pin assignments, select **Pin** from the Category bar. Then, choose **Export** (File menu), and select **Comma Separated Value File** in the **Save as type** list.

The first uncommented row of the **.csv** file is a list of the column headings displayed in the Assignment Editor separated by commas. Each row below the header row represents the rows in the spreadsheet of the Assignment Editor (see Figure 1–15) You can view and make edits to the **.csv** file with Excel or other spreadsheet tools.

*Figure 1–15. Assignment Editor with Category set to Pin*



| | To | Location | I/O Bank | I/O Standard | General Function | Special Function | Reserved | SignalProbe So |
|---|---|---|---|---|---|---|---|---|
| 1 | clk | PIN_K5 | 1 | LVTTL | Dedicated Clock | CLK0/LVDSCLK1p | | |
| 2 | button | PIN_W3 | 4 | LVTTL | Column I/O | LVDS128p | | |
| 3 | q[0] | PIN_E14 | 2 | LVTTL | Column I/O | LVDS56n | | |
| 4 | q[1] | PIN_E13 | 2 | LVTTL | Column I/O | LVDS56p | | |
| 5 | q[2] | PIN_C14 | 2 | LVTTL | Column I/O | LVDS55n/DQ0T4 | | |
| 6 | q[3] | PIN_D14 | 2 | LVTTL | Column I/O | LVDS55p/DQ0T5 | | |
| 7 | q[4] | PIN_E12 | 2 | LVTTL | Column I/O | LVDS52n | | |
| 8 | q[5] | PIN_F12 | 2 | LVTTL | Column I/O | LVDS52p | | |
| 9 | q[6] | PIN_B3 | 2 | LVTTL | Column I/O | LVDS32n/DEV_OE | | |
| 10 | q[7] | PIN_B14 | 2 | LVTTL | Column I/O | LVDS57p | | |

Here is an example of an exported **.csv** file from the Assignment Editor.

```
# Note: The column header names should not be changed if you wish to import #
this .csv file into the Quartus II software.
To,Location,I/O Bank,I/O Standard,General Function,Special Function, \
   Reserved ,SignalProbe Source
clk,PIN_K5,1,LVTTL,Dedicated Clock,CLK0/LVDSCLK1p,,
button,PIN_W3,4,LVTTL,Column I/O,LVDS128p,,
q[0],PIN_E14,2,LVTTL,Column I/O,LVDS56n,,
q[1],PIN_E13,2,LVTTL,Column I/O,LVDS56p,,
q[2],PIN_C14,2,LVTTL,Column I/O,LVDS55n/DQ0T4,,
q[3],PIN_D14,2,LVTTL,Column I/O,LVDS55p/DQ0T5,,
q[4],PIN_E12,2,LVTTL,Column I/O,LVDS52n,,,
```

## Importing Assignments

The **Import Assignments** dialog box allows you to import Quartus II assignments from a **.qsf**, **.esf**, **.acf** or **.csv** file (see Figure 1–16). To import assignments from any of the supported assignment files, follow these steps:

1. Choose **Import Assignments** (Assignments menu).

2. In the **File name** text-entry box, enter the file name, or click on the “**...**” box (Browse) to navigate to the assignment file.

3. When the **Select File** dialog box opens**,** select the file, and click **Open** to close the **Select File** dialog box.

4. Click **OK** in the **Import Assignments** dialog box.

When you import a **.csv** file, the first uncommented row of the file must be in the exact same format as it was when exported.

When using the Logiclock flow methodology to import assignments, follow these steps:

1. Choose **Import Assignments** (Assignments menu).

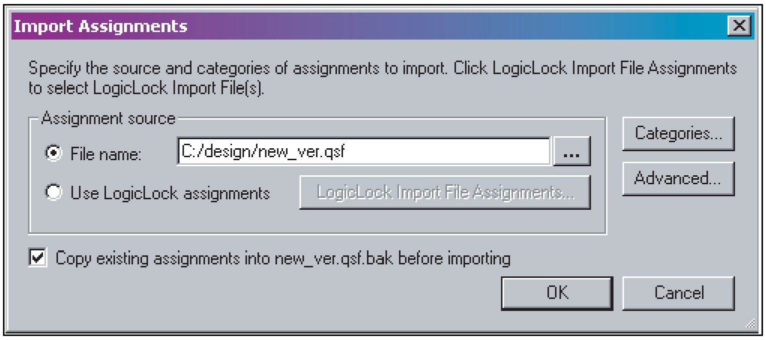
2. Select the **Use LogicLock assignments** button, and click on the **LL_IMPORT_FILE Assignments...** box.

3. When the **LogicLock Import File Assignments** window opens, select the LogicLock import file assignments to use for importing, and click **OK** to close the window.

For more information on using the **Import Assignments** dialog box to import LogicLock regions, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook*.

You can create a copy of your assignments before importing new assignments by selecting the checkbox for **Copy existing assignments into <*revision name*>.qsf.bak before importing** option.

*Figure 1–16. Import Assignments Dialog Box*



Specify the source and categories of assignments to import. Click LogicLock Import File Assignments to select LogicLock Import File(s).

● File name: C:/design/new_ver.qsf

○ Use LogicLock assignments

☑ Copy existing assignments into new_ver.qsf.bak before importing

When importing assignments from a file, you can choose which assignment categories to import by following these steps:

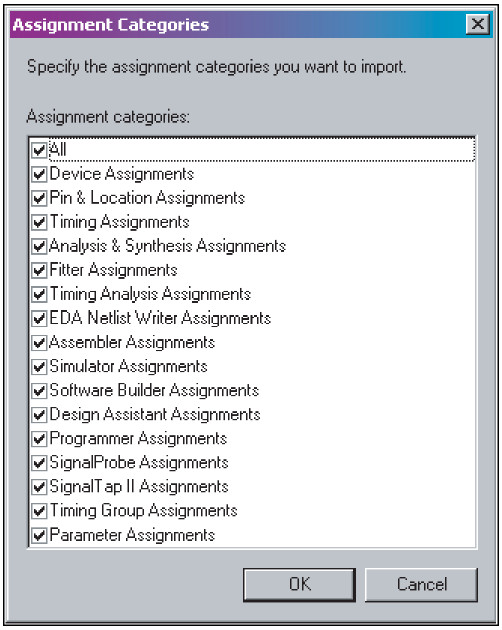
1. Click **Categories** in the **Import Assignments** dialog box.

2. Select the checkbox for each of the **Assignment Categories** you want to import, as shown in Figure 1–17.

To select specific types of assignments to import, click **Advanced** in the **Import Assignments** dialog box. The **Advanced Import Settings** dialog box appears and you can choose to import instance, entity, or global assignments, as well as select various assignment types to import.

For more information on these options, refer to the Quartus II software Help.

*Figure 1–17. Assignment Categories Dialog Box*



Specify the assignment categories you want to import.

Assignment categories:

- ☑ All
- ☑ Device Assignments
- ☑ Pin & Location Assignments
- ☑ Timing Assignments
- ☑ Analysis & Synthesis Assignments
- ☑ Fitter Assignments
- ☑ Timing Analysis Assignments
- ☑ EDA Netlist Writer Assignments
- ☑ Assembler Assignments
- ☑ Simulator Assignments
- ☑ Software Builder Assignments
- ☑ Design Assistant Assignments
- ☑ Programmer Assignments
- ☑ SignalProbe Assignments
- ☑ SignalTap II Assignments
- ☑ Timing Group Assignments
- ☑ Parameter Assignments

## Conclusion

As FPGAs continue to increase in density and pin count, it is essential to be able to quickly create and view design assignments. The Assignment Editor provides an intuitive and effective way of making assignments. With the spreadsheet interface and the **Category, Edit,** and **Node Filter** bars, the Assignment Editor provides an efficient assignment entry solution for FPGA designers.

To learn more about efficiently creating pin assignments with the Assignment Editor, see the *I/O Assignment Planning and Analysis* chapter in Volume 2 of the *Quartus II Handbook.*

# 2. Command-Line Scripting

qii52002-2.0

## Introduction

FPGA design software that is easy to integrate into a design flow saves time and improves productivity. The Altera® Quartus® II software provides designers with modular executables for each step in an FPGA design flow to make the design process customizable and flexible.

The benefits provided by modular executables include command-line control over each step of the design flow, easy integration with scripted design flows including makefiles, reduced memory requirements, and improved performance. The modular executables are also completely compatible with the Quartus II graphical user interface (GUI), allowing you to use the exact combination of tools you prefer.

This chapter describes how to take advantage of Quartus II modular executables, and provides several examples of their use in certain design situations.

## The Benefits of Modular Executables

The Quartus II modular executables reduce the amount of memory required during any step in the design flow. Because it targets only one step in the design flow, each executable is relatively compact, both in terms of file size and the amount of memory used when running. This memory reduction improves performance for all designers and is particularly beneficial in design environments with heavily-used computer networks or mature workstations with low amounts of memory.

Modular executables also provide command-line control over each step of the design flow. Each modular executable has options to control commonly-used software settings. Each modular executable also provides detailed, built-in help describing its function, available options, and settings.

Modular executables allow for easy integration with scripted design flows. It is simple to create scripts in any language with a series of modular executable commands. These scripts can be batch-processed, allowing for integration with distributed computing in server farms. The Quartus II modular executables can also be integrated in makefile-based design flows. All of these features enhance the ease of integration between the Quartus II software and other EDA synthesis, simulation, and verification software.

Modular executables add integration and scripting flexibility for designers who want it without sacrificing the ease-of-use of the Quartus II GUI. You can use the Quartus II GUI and modular executables at different stages in the design flow. As an example, you might use the Quartus II GUI to edit the floorplan for the design, use the modular executables to perform place-and-route, and return to the Quartus II GUI to perform debugging with the Chip Editor.

## Introductory Example

The following introduction to design flow with modular executables shows how to create a project, fit the design, perform timing analysis, and generate programming files.

The tutorial design included with the Quartus II software is used to demonstrate this functionality. If installed, the tutorial design is found in the <*Quartus II directory*>**/qdesigns/tutorial** directory.

Before making changes, copy the tutorial directory and type the following four commands at a command prompt in the new project directory.

☞ The <*quartus*>**/bin** directory must be in your `PATH` environment variable.

```
quartus_map filtref --source=filtref.bdf --family=CYCLONE ↵
quartus_fit filtref --part=EP1C12Q240C6 --fmax=80MHz --tsu=8ns ↵
quartus_tan filtref ↵
quartus_asm filtref ↵
```

The `quartus_map filtref --source=filtref.bdf --family=CYCLONE` command creates a new Quartus II project called **filtref** with the **filtref.bdf** file as the top-level file. It targets the Cyclone device family and performs logic synthesis and technology mapping on the design files.

The `quartus_fit filtref --part=EP1C12Q240C6 --fmax=80MHz --tsu=8ns` command performs fitting on the **filtref** project. The command specifies an EP1C12Q240C6 device and the fitter attempts to meet a global $f_{MAX}$ requirement of 80 MHz and a global $t_{SU}$ requirement of 8 ns.

The `quartus_tan filtref` command performs timing analysis on the **filtref** project to determine whether the design meets the timing requirements that were specified by the `quartus_fit` command.

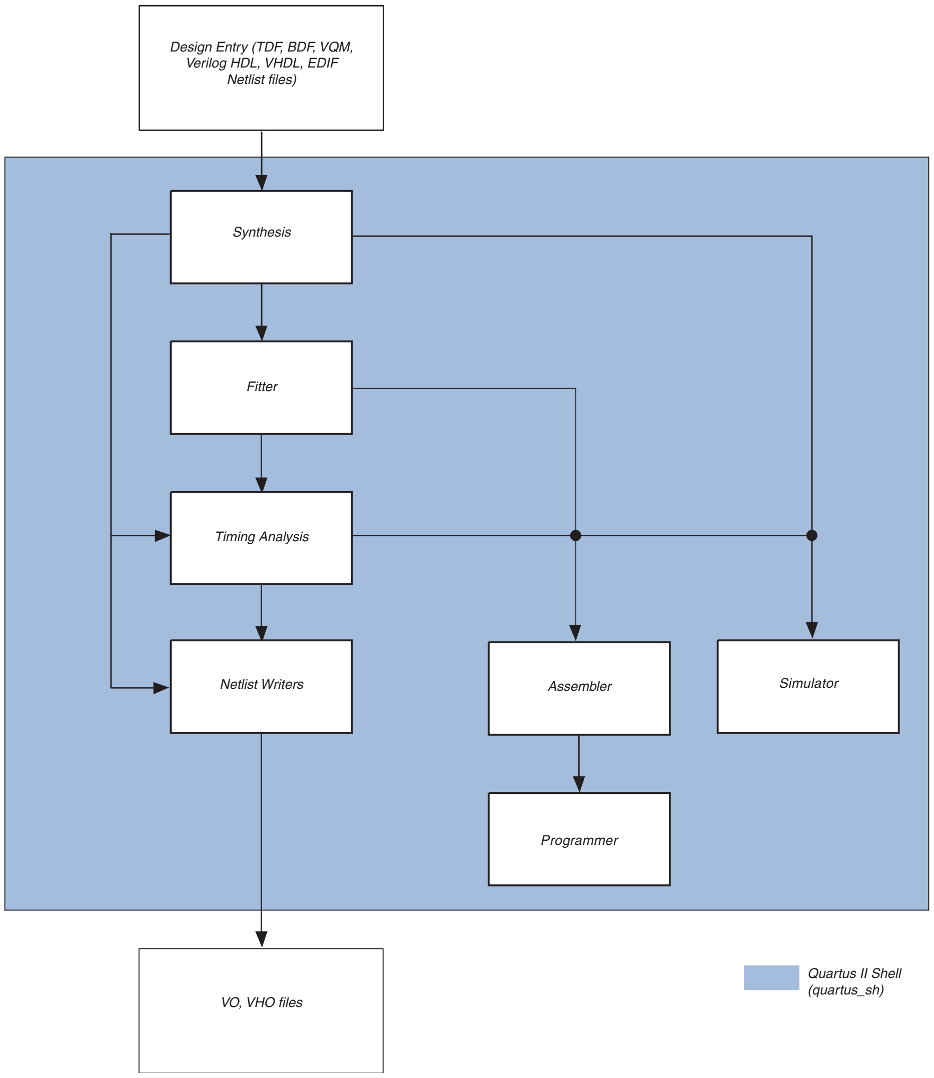
The `quartus_asm filtref` command creates programming files for the **filtref** project.

These four commands can be stored in a batch file for use on PCs or in a shell script file for use on UNIX workstations.

## Design Flow

Figure 2–1 shows a typical design flow.

*Figure 2–1. Typical Design Flow*


Design Entry (TDF, BDF, VQM, Verilog HDL, VHDL, EDIF Netlist files)
Synthesis
Fitter
Timing Analysis
Netlist Writers
Assembler
Simulator
Programmer
VO, VHO files
Quartus II Shell (quartus_sh)

Modular executables are provided for each stage in the design flow shown in Figure 2–1. Additional modular executables are provided for specific tasks. Table 2–1 lists each Quartus II modular executable and provides a brief description of its function.

*Table 2–1. Quartus II Modular Executables & Descriptions (Part 1 of 3)*

| Executable | Description |
|---|---|
| **Analysis** & **Synthesis**<br>**quartus_map** | Quartus II Analysis & Synthesis builds a single project database that integrates all the design files in a design entity or project hierarchy, performs logic synthesis to minimize the logic of the design, and performs technology mapping to implement the design logic using device resources such as logic elements. |
| **Fitter**<br>**quartus_fit** | The Quartus II Fitter performs place-and-route by fitting the logic of a design into a device. The Fitter selects appropriate interconnection paths, pin assignments, and logic cell assignments.<br><br>Quartus II Analysis & Synthesis must be run successfully before running the Fitter. |
| **Timing Analyzer**<br>**quartus_tan** | The Quartus II Timing Analyzer computes delays for the given design and device, and annotates them on the netlist. Then, the Timing Analyzer performs timing analysis, allowing you to analyze the performance of all logic in your design. The quartus_tan executable includes Tcl support.<br><br>Quartus II Analysis & Synthesis or the Fitter must be run successfully before running the Timing Analyzer. |
| **Assembler**<br>**quartus_asm** | The Quartus II Assembler generates a device programming image, in the form of one or more Programmer Object Files (.**pof**), SRAM Object Files (.**sof**), Hexadecimal (Intel-Format) Output Files (.**hexout**), Tabular Text Files (.**ttf**), and Raw Binary Files(.**rbf**), from a successful fit (that is, place-and-route).<br><br>The .**pof** and .**sof** files are then processed by the Quartus II Programmer and downloaded to the device with the MasterBlaster™ or the ByteBlaster™ II Download Cable, or the Altera Programming Unit (APU). The .**hexout**.**ttf**, TTFs, and RBFs can be used by other programming hardware manufacturers that provide support for Altera devices.<br><br>The Quartus II Fitter must be run successfully before running the Assembler. |

*Table 2–1. Quartus II Modular Executables & Descriptions (Part 2 of 3)*

| Executable | Description |
|---|---|
| **Design Assistant**<br>**quartus_drc** | The Quartus II Design Assistant checks the reliability of a design based on a set of design rules. The Design Assistant is especially useful for checking the reliability of a design before converting the design for HardCopy™ devices.<br>The Design Assistant supports designs that target any Altera device supported by the Quartus II software, except MAX® 3000 and MAX 7000 devices.<br><br>Quartus II Analysis & Synthesis or the Fitter must be run successfully before running the Design Assistant. |
| **Compiler Database Interface**<br>**quartus_cdb** | The Quartus II Compiler Database Interface generates incremental netlists for use with LogicLock™ back-annotation, or back-annotates device and resource assignments to preserve the fit for future compilations. The quartus_cdb executable includes Tcl support.<br><br>Analysis & Synthesis must be run successfully before running the Compiler Database Interface. |
| **EDA Netlist Writer**<br>**quartus_eda** | The Quartus II EDA Netlist Writer generates netlist and other output files for use with other EDA tools.<br><br>Analysis & Synthesis, the Fitter, or Timing Analyzer must be run successfully before running the EDA Netlist Writer, depending on the arguments used. |
| **Simulator**<br>**quartus_sim** | The Quartus II Simulator tests and debugs the logical operation and internal timing of the design entities in a project. The Simulator can perform two types of simulation: functional simulation and timing simulation. The quartus_sim executable includes Tcl support.<br><br>Quartus II Analysis & Synthesis must be run successfully before running a functional simulation.<br><br>The Timing Analyzer must be run successfully before running a timing simulation. |
| **Software Build**<br>**quartus_swb** | The Quartus II Software Builder performs a software build, which processes a design for an ARM®-based Excalibur™ device or the Nios® embedded processor. |
| **Programmer**<br>**quartus_pgm** | The Quartus II Programmer programs Altera devices. The Programmer uses one of the valid supported file formats: Programmer Object Files (.**pof**), SRAM Object Files (.**sof**), Jam File (.**jam**), or Jam Byte-Code File (.**jbc**).<br>Make sure you specify a valid programming mode, programming cable, and operation for a specified device. |

*Table 2–1. Quartus II Modular Executables & Descriptions (Part 3 of 3)*

| Executable | Description |
|---|---|
| **Convert Programming File** **quartus_cpf** | The Quartus II Convert Programming File module converts one programing file format to a different possible format.<br><br>Make sure you specify valid options and an input programming file to generate the new requested programming file format. |
| **Quartus Shell** **quartus_sh** | The Quartus II Shell acts as a simple Quartus II Tcl interpreter. The Shell has a smaller memory footprint than the other command-line executables that support Tcl: quartus_tan, quartus_cdb, and quartus_sim. The Shell may be started as an interactive Tcl interpreter (shell), used to run a Tcl script, or used as a quick Tcl command evaluator, evaluating the remaining command-line arguments as one or more Tcl commands. |

## Text-Based Report Files

Each modular executable creates a text-format report file when it is run. These files report success or failure, and contain information on the processing performed by the modular executable.

Report file names contain the revision names and name of the modular executable that generated the report file. For example, the report file name format is *<revision name>*.*<modular executable>***.rpt**. For example, using the **quartus_fit** modular executable to place-and-route a project with the revision name **design_top** generates a report file named **design_top.fit.rpt**. Likewise, using the **quartus_tan** modular executable to perform timing analysis on a project with the revision name **fir_filter** generates a report file named **fir_filter.tan.rpt**.

As an alternative to parsing text-based report files, you can use the Tcl package called `::quartus::report`. For more information on this package, see "More Help with Quartus II Modular Executables" on page 2–17.

### Compilation with quartus_sh --flow

Use the `quartus_sh` executable with the `--flow` option to perform a complete compilation flow with a single command. (For information on specialized flows, type `quartus_sh --help=flow` at a command prompt.) The `--flow` option supports the smart recompile feature, and efficiently sets command-line arguments for each executable in the flow.

☞ If you used the `quartus_cmd` command to perform command-line compilations in earlier versions of the Quartus II software, Altera recommends that you use the `quartus_sh --flow` option in the Quartus II software version 4.1.

The following example runs compilation, timing analysis, and programming file generation—with a single command.

```
quartus_sh --flow compile filtref ↵
```

## Command-Line Scripting Help

Complete help information is integrated with each modular executable. For more information about the modular executable options, use the help information integrated with the modular executables. Access help for a modular executable using the `-h` option. For example, to view help for the `quartus_map` modular executable, run the command `quartus_map -h`. The following example shows the result of running `quartus_map -h`:

```
C:\>quartus_map -h
Quartus II Analysis & Synthesis
Version 4.1 Internal Build 133 04/07/2004 SJ Full Version
Copyright (C) 1991-2004 Altera Corporation

Usage:
------
quartus_map [-h | --help[=<option|topic>] | -v]
quartus_map <project name> [<options>]

Description:
------------
Quartus(R) II Analysis & Synthesis builds a single project
database that integrates all the design files in a design
entity or project hierarchy, performs logic synthesis to
minimize the logic of the design, and performs technology
mapping to implement the design logic using device
resources such as logic elements.

Options:
--------
-f <argument file>
```

```
        -c <revision name> | --rev=<revision name>
        -l <path> | --lib_path=<path>
        --lower_priority
        --optimize=<area|speed|balanced>
        --family=<device family>
        --part=<device>
        --
state_machine_encoding=<auto|minimal_bits|one_hot|user_enco
        --enable_register_retiming[=on|off]
        --enable_wysiwyg_resynthesis[=on|off]
        --ignore_carry_buffers[=on|off]
        --ignore_cascade_buffers[=on|off]
        --analyze_project
        --analyze_file=<design file>
        --generate_symbol=<design file>
        --generate_inc_file=<design file>
        --convert_bdf_to_verilog=<.bdf file>
        --convert_bdf_to_vhdl=<.bdf file>
        --export_settings_files[=on|off]
        --generate_functional_sim_netlist
        --source=<source file>
        --update_wysiwyg_parameters

Help Topics:
------------
arguments
makefiles

For more information on specific options, use --
help=<option|topic>.
```

Detailed help about a particular option is also available. For example, to view detailed help about the `--optimize` option, run `quartus_map --help=optimize`. The following is the result of running `quartus_map --help=optimize`:

```
Option: --optimize=<area|speed|balanced>

Option to optimize the design to achieve maximum speed
performance, minimum area usage, or high speed performance
with miminal area cost during synthesis.

The following table displays available values:

Value       Description
--------    ----------------------------------------------
area        Makes the design as small as possible in order
            to minimize resource usage.

speed       Chooses a design implementation that has the
            fastest fmax.

balanced    Chooses a design implementation that has a
            high-speed performance with minimal logic usage
Note that the current version of the Quartus(R) II software
does not support the "balanced" setting for the following
devices:
```

```
Mercury(TM), MAX(R) 7000B/7000AE/3000A/7000S/7000A,
FLEX(R) 6000, FLEX 10K(R), FLEX 10KE/10KA, and ACEX 1K.
```

Help on Quartus II modular executables is also available by typing `quartus_sh --qhelp` at a command prompt. For more information, see "More Help with Quartus II Modular Executables" on page 2–17.

## Command-Line Option Details

Command-line options are provided for making many common global project settings and performing common tasks. You can use either of two methods to make assignments to an individual entity. If the project exists, open the project in the Quartus II GUI, change the assignment, and close the project. The changed assignment is updated in the Quartus II Settings file. Any modular executables that are run after this update will use the updated assignment. See "Option Precedence" on page 2–9 for more information. You can also make assignments using the Quartus II Tcl scripting API. If you want to completely script the creation of a Quartus II project, you should choose this method.

### Option Precedence

If you are using the modular executables, you need to be aware of the precedence of various project assignments and how to control the precedence. Assignments for a particular project exist in the Quartus II Settings file (.**qsf**) for the project. Assignments for a project can also be made by using command-line options, as described earlier in this document. Project assignments are reflected in compiler database files that hold intermediate compilation results and reflect assignments made in the previous project compilation.

All command-line options override any conflicting assignments found in the QSF or the compiler database files. There are two command-line options to specify whether QSF or compiler database files take precedence for any assignments not specified as command-line options.

Any assignment not specified as a command-line option or found in the QSF or compiler database files will be set to its default value.

The file precedence command-line options are `--import_settings_files` and `--export_settings_files`. By default, the `--import_settings_files` and `--export_settings_files` options are turned on. Turning the

`--import_settings_files` option on causes a modular executable to read assignments from the Quartus II settings file instead of from the compiler database files. Turning the `--export_settings_files` option on causes a modular executable to update the Quartus II settings file to reflect any specified options, as happens when closing a project in the Quartus II GUI.

Table 2–2 lists the precedence for reading assignments depending on the value of the `--import_settings` option.

***Table 2–2. Precedence for Reading Assignments***

| Option Specified | Precedence for Reading Assignments |
|---|---|
| `--import_settings_files=on` (Default) | 1. Command-line options<br>2. Quartus II Settings File (.qsf)<br>3. Project database (**db** directory, if it exists)<br>4. Quartus II software defaults |
| `--import_settings_files=off` | 1. Command-line options<br>2. Project database (**db** directory, if it exists)<br>3. Quartus II software defaults |

Table 2–3 lists the locations to which assignments are written, depending on the value of the `--export_settings` command-line option.

***Table 2–3. Location for Writing Assignments***

| Option Specified | Location for Writing Assignments |
|---|---|
| `--export_settings_files=on` (Default) | Quartus II Settings File (**.qsf**) and compiler database |
| `--export_settings_files=off` | Compiler database |

The following example assumes that a project named `fir_filter` exists, and that the analysis and synthesis step has been performed (using the `quartus_map` command).

```
quartus_fit fir_filter --fmax=80MHz ↵
quartus_tan fir_filter ↵
quartus_tan fir_filter --fmax=100MHz --tao=timing_result-100.tao
--export_settings_files=off ↵
```

The first command, `quartus_fit fir_filter --fmax=80MHz`, runs the `quartus_fit` executable and specifies a global $f_{MAX}$ requirement of 80 MHz.

The second command, `quartus_tan fir_filter`, runs Quartus II timing analysis for the results of the previous fit.

The third command reruns Quartus II timing analysis with a global $f_{MAX}$ requirement of 100 MHz and saves the result in a file called **timing_result-100.tao**. By specifying the `--export_settings_files=off` option, the modular executable does not update the Quartus II settings file to reflect the changed $f_{MAX}$ requirement. The compiler database files reflect the changed $f_{MAX}$ requirement. If the `--export_settings_files=off` option is not specified, the modular executable updates the Quartus II settings file to reflect the 100-MHz global $f_{MAX}$ requirement.

Use the `--import_settings_files=off` and `--export_settings_files=off` options (where appropriate) to optimize the way that the Quartus II software reads and updates settings files. The following example shows how to avoid unnecessary importing and exporting.

```
quartus_map filtref --source=filtref --part=ep1s10f780c5 ↵
quartus_fit filtref --fmax=100MHz --import_settings_files=off ↵
quartus_tan filtref --import_settings_files = off --export_settings_files
= off ↵
quartus_asm filtref --import_settings_files=off --export_settings_files
= off ↵
```

The `quartus_tan` and `quartus_asm` executables do not need to import or export settings files because they do not change any settings in the project.

## Command-Line Scripting Examples

This section of the chapter presents various examples of command-line executable use.

### Check Design File Syntax

This shell script example assumes that the Quartus II software tutorial project called **fir_filter** exists in the current directory. (This project exists in the <*Quartus II directory*>**/qdesigns/fir_filter** directory unless the Quartus II software tutorial files are not installed.) The `--analyze_file` option specifies each file on which to perform a syntax check. The script checks the exit code of the `quartus_map` executable to determine whether there was an error during the syntax check. Files with syntax errors are added to the `FILES_WITH_ERRORS` variable, and when all files have been checked for syntax, the script prints a message indicating whether there were any syntax errors. Any options that are not specified use the values from the project database. If not specified there, then the executable uses the Quartus II software default values. For example, the **fir_filter** project is set to target the Cyclone device family, so it is not necessary to specify the `--family` option.

This shell script is specifically designed for use on UNIX systems employing the `sh` shell.

```
#!/bin/sh
FILES_WITH_ERRORS=""
# Iterate over each file with a .bdf or .v extension
for filename in `ls *.bdf *.v`
do
  # Perform a syntax check on the specified file
    quartus_map fir_filter --analyze_file=$filename
  # If the exit code is non-zero, the file has a syntax error
    if [ $? -ne 0 ]
    then
       FILES_WITH_ERRORS="$FILES_WITH_ERRORS $filename"
    fi
done

if [ -z "$FILES_WITH_ERRORS" ]
then
    echo "All files passed the syntax check"
    exit 0
else
    echo "There were syntax errors in the following file(s)"
    echo $FILES_WITH_ERRORS
    exit 1
fi
```

## Create a Project & Synthesize a Netlist Using Netlist Optimizations

This example creates a new Quartus II project with a file **top.edf** as the top-level entity. The `--enable_register_retiming=on` and `--enable_wysiwyg_resynthesis=on` options allow the technology mapper to optimize the design using gate-level register retiming and technology remapping.

For more details about register retiming, technology remapping, and other netlist optimization options, consult the Quartus II Help.

The `--part` option tells the technology mapper to target an EP20K600EBC652-1X device. To create the project and synthesize it using the netlist optimizations described above, type the following command at a command prompt:

```
quartus_map top --source=top.edf --enable_register_retiming=on
    --enable_wysiwyg_resynthesis=on --part=EP20K600EBC652-1X↵
```

## Attempt to Fit a Design as Quickly as Possible

This example assumes that a project called **top** exists in the current directory, and that the name of the top-level entity is **top**. The `--effort=fast` option forces the Fitter to use the fast fit algorithm to increase compilation speed, possibly at the expense of reduced $f_{MAX}$ performance. The `--one_fit_attempt=on` option restricts the Fitter to only one fitting attempt for the design.

To attempt to fit the project called **top** as quickly as possible, type the following command at a command prompt:

```
quartus_fit top --effort=fast --one_fit_attempt=on ↵
```

## Fit a Design Using Multiple Seeds

This shell script example assumes that the Quartus II software tutorial project called **fir_filter** exists in the current directory (defined in a file called **fir_filter.qpf**). If the tutorial files are installed on your system, this project exists in the <*Quartus II directory*>**/qdesigns/fir_filter** directory. Because the top-level entity in the project does not have the same name as the project, you must specify the revision name for the top-level entity with the `--rev` option. The `--seed` option specifies the seeds to use for fitting.

A seed is a parameter that affects the random initial placement of the Quartus II Fitter. Varying the seed can result in better performance for some designs.

After each fitting attempt, the script creates new directories for the results of each fitting attempt and copies the complete project to the new directory so that the results are available for viewing and debugging after the script has completed.

This shell script is specifically designed for use on UNIX systems employing the `sh` shell.

```
#!/bin/sh
ERROR_SEEDS=""
quartus_map fir_filter --rev=filtref
# Iterate over a number of seeds
for seed in 1 2 3 4 5
do
echo "Starting fit with seed=$seed"
# Perform a fitting attempt with the specified seed
        quartus_fit fir_filter --seed=$seed --rev=filtref
# If the exit-code is non-zero, the fitting attempt was
# successful, so copy the project to a new directory
        if [ $? -eq 0 ]
        then
```

```
                mkdir ../fir_filter-seed_$seed
                mkdir ../fir_filter-seed_$seed/db
                cp * ../fir_filter-seed_$seed
                cp db/* ../fir_filter-seed_$seed/db
            else
                ERROR_SEEDS="$ERROR_SEEDS $seed"
            fi
done
if [ -z "$ERROR_SEEDS" ]
then
            echo "Seed sweeping was successful"
            exit 0
else
            echo "There were errors with the following seed(s)"
            echo $ERROR_SEEDS
            exit 1
fi
```

Use the Design Space Explorer included with the Quartus II software (DSE) script (by typing `quartus_sh --dse` ↵ at a command prompt) to improve design performance by performing automated seed sweeping.

For more information on the DSE, type `quartus_sh --help=dse` ↵ at the command prompt, or see the *Design Space Explorer* chapter in Volume 2 of the *Quartus II Handbook*.

## Makefile Implementation

You can also use the Quartus II modular executables in conjunction with the **make** utility to automatically update files when other files they depend on change. The file dependencies and commands used to update files are specified in a text file called a makefile. The following example is one way of implementing a makefile with modular executables.

```
################################################################
# Project Configuration:
#
# Specify the name of the design (project) and the list of source
# files used.
################################################################

PROJECT = chiptrip
SOURCE_FILES = auto_max.v chiptrip.v speed_ch.v tick_cnt.v
time_cnt.v
ASSIGNMENT_FILES = chiptrip.qpf chiptrip.qsf

################################################################
# Main Targets
#
# all: build everything
# clean: remove output files and database
# clean_all: removes settings files as well as clean.
################################################################
```

```
all: smart.log $(PROJECT).asm.rpt $(PROJECT).tan.rpt

clean:
    rm -rf *.rpt *.chg smart.log *.htm *.eqn *.pin *.sof *.pof db
rm-rf*.summary
clean_all: clean
    rm -rf *.qpf*.qsf *.qws

map: smart.log $(PROJECT).map.rpt
fit: smart.log $(PROJECT).fit.rpt
asm: smart.log $(PROJECT).asm.rpt
tan: smart.log $(PROJECT).tan.rpt
smart: smart.log

#################################################################
# Executable Configuration
#################################################################

MAP_ARGS = --family=Stratix
FIT_ARGS = --part=EP1S20F484C6
ASM_ARGS =
TAN_ARGS =

#################################################################
# Target implementations
#################################################################

STAMP = echo done >

$(PROJECT).map.rpt: map.chg $(SOURCE_FILES)
    quartus_map $(MAP_ARGS) $(PROJECT)
    $(STAMP) fit.chg

$(PROJECT).fit.rpt: fit.chg $(PROJECT).map.rpt
    quartus_fit $(FIT_ARGS) $(PROJECT)
    $(STAMP) asm.chg
    $(STAMP) tan.chg

$(PROJECT).asm.rpt: asm.chg $(PROJECT).fit.rpt
    quartus_asm $(ASM_ARGS) $(PROJECT)

$(PROJECT).tan.rpt: tan.chg $(PROJECT).fit.rpt
    quartus_tan $(TAN_ARGS) $(PROJECT)

smart.log: $(ASSIGNMENT_FILES)
    quartus_sh --determine_smart_action $(PROJECT) > smart.log

#################################################################
# Project initialization
#################################################################

$(ASSIGNMENT_FILES):
    quartus_sh --prepare $(PROJECT)

map.chg:
    $(STAMP) map.chg
fit.chg:
    $(STAMP) fit.chg
tan.chg:
    $(STAMP) tan.chg
asm.chg:
    $(STAMP) asm.chg
```

A Tcl script is provided with the Quartus II software to create or modify files which can be specified as dependencies in the make rules, assisting you in makefile development. Complete information about this Tcl script and how to integrate it with makefiles is available by running the command `quartus_sh --help=determine_smart_action`.

## The QFlow Script

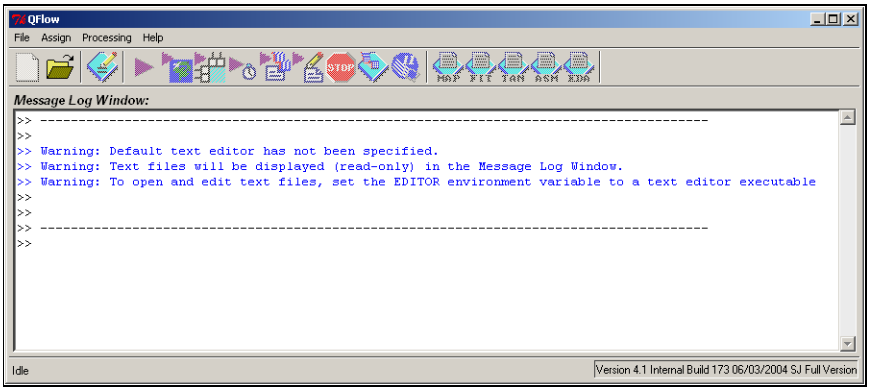
A Tcl/Tk-based graphical interface called QFlow is included with the modular executables. Designers can use the QFlow interface to open projects, launch some of the modular executables, view report files, and make some global project assignments. The QFlow interface can run the following modular executables:

- **quartus_map** (Analysis & Synthesis)
- **quartus_fit** (Fitter)
- **quartus_tan** (Timing Analysis)
- **quartus_asm** (Assembler)
- **quartus_eda** (EDA Netlist Writer)

To view floorplans or perform other GUI-intensive tasks, launch the Quartus II GUI.

Start QFlow by typing the following command at a command prompt: `quartus_sh -g` ↵. Figure 2–2 shows the QFlow interface.

*Figure 2–2. QFlow Interface*



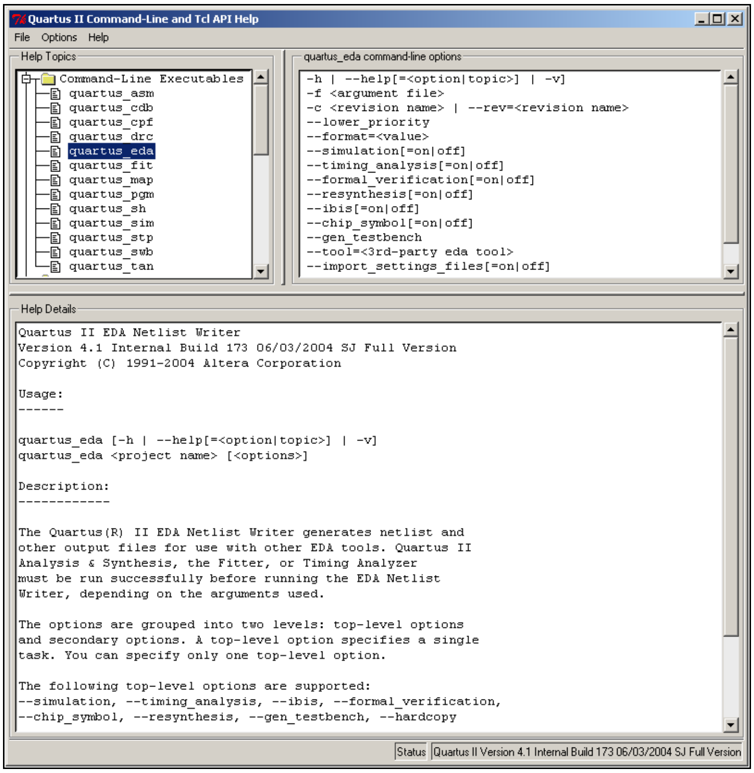
```
>> ----------------------------------------------------------------------------------------
>>
>> Warning: Default text editor has not been specified.
>> Warning: Text files will be displayed (read-only) in the Message Log Window.
>> Warning: To open and edit text files, set the EDITOR environment variable to a text editor executable
>>
>>
>> ----------------------------------------------------------------------------------------
>>
```

The QFlow script is located in the <*Quartus II directory*>**/bin/tcl_scripts/qflow/** directory.

## More Help with Quartus II Modular Executables

More information on modular executable use and the Quartus II Tcl API is available by typing `quartus_sh --qhelp` at a command prompt. This command starts the Quartus II Command Line and Tcl API Help browser, a viewer for information on the Quartus II modular executables and Tcl API (Figure 2–3).

*Figure 2–3. Quartus II Command Line & Tcl API Help Browser*


Quartus II Command-Line and Tcl API Help
File
Options
Help
Help Topics
quartus_eda command-line options
Help Details
Status

```
Command-Line Executables
  quartus_asm
  quartus_cdb
  quartus_cpf
  quartus_drc
  quartus_eda
  quartus_fit
  quartus_map
  quartus_pgm
  quartus_sh
  quartus_sim
  quartus_stp
  quartus_swb
  quartus_tan
```

```
-h | --help[=<option|topic>] | -v]
-f <argument file>
-c <revision name> | --rev=<revision name>
--lower_priority
--format=<value>
--simulation[=on|off]
--timing_analysis[=on|off]
--formal_verification[=on|off]
--resynthesis[=on|off]
--ibis[=on|off]
--chip_symbol[=on|off]
--gen_testbench
--tool=<3rd-party eda tool>
--import_settings_files[=on|off]
```

```
Quartus II EDA Netlist Writer
Version 4.1 Internal Build 173 06/03/2004 SJ Full Version
Copyright (C) 1991-2004 Altera Corporation

Usage:
------

quartus_eda [-h | --help[=<option|topic>] | -v]
quartus_eda <project name> [<options>]

Description:
------------

The Quartus(R) II EDA Netlist Writer generates netlist and
other output files for use with other EDA tools. Quartus II
Analysis & Synthesis, the Fitter, or Timing Analyzer
must be run successfully before running the EDA Netlist
Writer, depending on the arguments used.

The options are grouped into two levels: top-level options
and secondary options. A top-level option specifies a single
task. You can specify only one top-level option.

The following top-level options are supported:
--simulation, --timing_analysis, --ibis, --formal_verification,
--chip_symbol, --resynthesis, --gen_testbench, --hardcopy
```

Quartus II Version 4.1 Internal Build 173 06/03/2004 SJ Full Version

Click items under **Help Topics** to get more information on the topics listed.

## Conclusion

Command-line scripting in Quartus II software provides important benefits to designers, including increased flexibility and easy integration with other EDA software in FPGA design flows. Scripts reduce memory usage, improve performance, and bring true command-line control to all stages of FPGA design.

# 3. Tcl Scripting

qii52003-2.1

## Introduction

Developing and running tool command language (Tcl) scripts to control the Altera® Quartus® II software allows you to perform a wide range of functions, such as compiling a design or writing procedures to automate common tasks.

You can automate your Quartus II assignments using Tcl scripts so that you do not have to create them individually. Tcl scripts also facilitate project or assignment migration. For example, when using the same prototype or development board for different projects, you can automate reassignment of pin locations in each new project. The Quartus II software can also generate a Tcl script based on all the current assignments in the project, which aids in migrating assignments to another project. You can use Tcl scripts to manage a Quartus II project, make assignments, define design constraints, make device assignments, run compilations, perform timing analysis, import LogicLock™ region assignments, use the Quartus II Chip Editor, and access reports.

The Quartus II software Tcl commands follow the electronic design automation (EDA) industry Tcl application programming interface (API) standards for using command-line options to specify arguments. This simplifies learning and using Tcl commands. If you encounter an error using a command argument, the Tcl interpreter gives help information showing correct usage.

This chapter includes sample Tcl scripts for the Quartus II software. You can modify these example scripts for use with your own designs.

## What is Tcl?

Tcl (pronounced tickle) is a popular scripting language that is similar to many shell scripting and high-level programming languages. It provides support for control structures, variables, network socket access, and APIs. Tcl is the EDA industry-standard scripting language used by Synopsys, Mentor Graphics®, Synplicity, and Altera software. It allows you to create custom commands and works seamlessly across most development platforms. For a list of recommended literature on Tcl, see "References" on page 3–28.

You can create your own procedures by writing scripts containing basic Tcl commands, user-defined procedures, and Quartus II API functions. You can then automate your design flow, run the Quartus II software in batch mode, or execute the individual Tcl commands interactively in the Quartus II Tcl interactive shell.

The Quartus II software version 4.1 supports Tcl/Tk version 8.4, supplied by the Tcl DeveloperXchange at **http://tcl.activestate.com**.

## Tcl Scripting Basics

This section is a brief introduction to Tcl, an interpreted scripting language. The core commands support variables, control structures, and procedures. Additionally, there are commands for accessing the file system and network sockets, and running other programs. You can create platform-independent graphical interfaces with the Tk widget set. There are many more Tcl commands and features not covered in this brief introduction.

For more information about Tcl scripting, consult any of the "References" on page 3–28.

### Hello World Example

This example shows the basic "Hello world" in Tcl.

```
puts "Hello world"
```

Use double quotation marks to group the words `hello` and `world` as one argument. Double quotation marks allow substitutions to occur in the group. Substitutions can be simple variable substitutions, or the result of running a nested command, described later in this section. Use curly braces ({}) for grouping when you want to prevent substitutions.

### Variables

Use the `set` command to assign a value to a variable. You do not have to declare a variable before using it. Tcl variable names are case-sensitive. This example assigns the value `1` to the variable named `a`.

```
set a 1
```

To access the contents of a variable, use a dollar sign before the variable name. This example also prints `"Hello world"`.

```
set a Hello
set b world
puts "$a $b"
```

### Nested Commands

Use square brackets to evaluate nested commands. The Tcl interpreter evaluates nested commands, starting with the innermost nested command, and commands nested at the same level from left to right. Each nested command result is substituted in the outer command. This example sets `a` to the length of the string `foo`.

```
set a [string length foo]
```

There are many other operations the string command can perform. Refer to the references at the end of this chapter for more information.

### Arithmetic

Use the `expr` command to perform arithmetic calculations. Using curly braces to group the arguments of this command makes arithmetic calculations more efficient and preserves numeric precision. This example sets `a` to the sum of `1` and the square root of `2`.

```
set a [expr { 1 + sqrt(2) }]
```

Tcl also supports boolean operators such as & (AND), `|` (OR), `!` (NOT), and comparison operators such as < (less than), > (greater than), and == (equal to).

For a complete list of supported operators, refer to "References" on page 3–28.

## Lists

A Tcl list is a series of values. Supported list operations include creating lists, appending lists, extracting elements, computing the length of a list, sorting a list, and more. This example sets `a` to a list with three numbers in it.

```
set a { 1 2 3 }
```

This example prints the 0th element of the list stored in `a`.

```
puts [lindex $a 0]
```

This example sets `b` to the length of the list stored in `a`.

```
set b [llength $a]
```

## Control structures

Tcl supports common control structures, including `if-then-else` conditions and `for`, `foreach`, and `while` loops. Positioning curly braces as shown in the following examples ensures the control structure commands are executed efficiently and correctly. This example prints whether the value of variable `a` is positive, negative, or zero.

```
if { $a > 0 } {
   puts "The value is positive"
} elseif { $a < 0 } {
   puts "The value is negative"
} else {
   puts "The value is zero"
}
```

This example uses a `for` loop to print each element in a list.

```
set a { 1 2 3 }
for { set i 0 } { $i < [llength $a] } { incr i } {
   puts "The list element at index $i is [lindex $a
$i]"
}
```

This example uses a `foreach` loop to print each element in a list

```
set a { 1 2 3 }
foreach element $a {
   puts "The list element is $element"
}
```

This example uses a `while` loop to print each element in a list

```
set a { 1 2 3 } {
set i 0
while { $i < [llength $a] } {
   puts "The list element at index $i is [lindex $a $i]"
   incr i
}
```

You do not need to use the `expr` command in boolean expressions in control structure commands because they invoke the `expr` command automatically.

## Procedures

Use the `proc` command to define a Tcl procedure (known as a subroutine or function in other scripting and programming languages). The scope of variables in a procedure is local to the procedure. If the procedure returns a value, use the return command to return the value from the procedure. This example defines a procedure that multiplies two numbers and returns the result

```
proc multiply { x y } {
   set product [expr { $x * $y }]
   return $product
}
```

This example shows how to use the `multiply` procedure in your code. You must define a procedure before your script calls it, as shown in this example.

```
proc multiply { x y } {
   set product [expr { $x * $y }]
   return $product
}
set a 1
set b 2
puts [multiply $a $b]
```

Altera recommends defining procedures near the beginning of a script. If you want to access global variables in a procedure, use the `global` command in each procedure that uses a global variable. This example defines a procedure that prints an element in a global list of numbers, then calls the procedure.

```
proc print_global_list_element { i } {
   global my_data
   puts "The list element at index $i is [lindex $my_data $i]"
}
set my_data { 1 2 3}
print_global_list_element 0
```

## Quartus II Tcl API Reference

Access the Quartus II Tcl API Help reference by typing the following command at a command prompt:

```
quartus_sh --qhelp
```

This command runs the Quartus II Command-Line and the Tcl API Help browser, which documents all commands and options in the Quartus II Tcl API. It includes detailed descriptions and examples for each command.

## Quartus II Tcl Packages

The Quartus II Tcl commands are grouped in packages by function. Table 3–1 describes each Tcl package.

*Table 3–1. Tcl Commands Grouped in Packages, by Function (Part 1 of 2)*

| Package Name | Package Description |
|---|---|
| project | Create and manage projects and revisions, make any project assignments including timing assignments. |
| flow | Compile a project, run command-line executables and other common flows |
| report | Get information from report tables, create custom reports |
| timing | Annotate timing netlist with delay information, compute and report timing paths |
| timing_report | List timing paths |
| advanced_timing | Traverse the timing netlist and get information about timing nodes |
| device | Get device and family information from the device database |
| backannotate | Back annotate assignments |
| logiclock | Create and manage LogicLock regions |

*Table 3–1. Tcl Commands Grouped in Packages, by Function (Part 2 of 2)*

| Package Name | Package Description |
|---|---|
| chip_editor | Identify and modify resource usage and routing with the Chip Editor |
| simulator | Configure and perform simulations |
| stp | Run the SignalTap II logic analyzer |
| database_manager | Manage version-compatible database files |
| misc | Perform miscellaneous tasks |

By default, only the minimum number of packages are loaded automatically with each Quartus II executable. This keeps the memory requirement for each executable as low as possible. Because the minimum number of packages are automatically loaded, you must load other packages before you can run commands in those packages, or get help on those packages.

Table 3–2 lists the Quartus II Tcl packages available with Quartus II executables and indicates whether a package is loaded by default or is available to be loaded as necessary. A blank space means the package is not available in that executable.

*Table 3–2. Tcl Package Availability by Quartus II Executable*

| Packages | Quartus II Executable | | | | |
|---|---|---|---|---|---|
| | Quartus_sh | Quartus_tan | Quartus_cdb | Quartus_sim | Tcl Console |
| advanced_timing | | Not Loaded | | | |
| backannotate | | | Not Loaded | | Not Loaded |
| chip_editor | | | Not Loaded | | |
| device | Loaded | Not Loaded | Loaded | Loaded | Not Loaded |
| flow | Not Loaded | Not Loaded | Not Loaded | Not Loaded | Not Loaded |
| logiclock | | Not Loaded | Not Loaded | | Not Loaded |
| misc | Loaded | Loaded | Loaded | Loaded | Loaded |
| project | Loaded | Loaded | Loaded | Loaded | Loaded |
| report | Not Loaded | Not Loaded | Not Loaded | Loaded | Not Loaded |
| simulator | | | | Loaded | |
| timing | | Loaded | | | |
| timing_report | | Not Loaded | | | Loaded |
| old_api | | | | | Loaded |

### Loading Packages

To load a Quartus II Tcl package, use the following Tcl command:

`load_package [-version` *<version number>*`]` *<package name>*.

This command is similar to the `package require` Tcl command, but you can easily alternate between different versions of a Quartus II Tcl package with the `load_package` command.

For additional information on these and other Quartus II command-line executables, see the *Command-Line Scripting* chapter in Volume 2 of the *Quartus II Handbook.*

## Executables Supporting Tcl

Some of the Quartus II command-line executables support Tcl scripting. They are listed in Table 3–3. Each executable supports different sets of Tcl packages. Refer to the following table to determine the appropriate executable to run your script.

*Table 3–3. Command-line Executables Supporting Tcl Scripting*

| Executable Name | Executable Description |
|---|---|
| quartus_sh | The Quartus II Shell is a simple Tcl scripting shell, useful for making assignments, general reporting, and compiling. |
| quartus_tan | Use the Quartus II Timing Analyzer to perform simple timing reporting and advanced timing analysis. |
| quartus_cdb | The Quartus II Compiler Database supports back annotation, LogicLock region operations, and chip editor functions |
| quartus_sim | Use the Quartus II Simulator to simulate designs with Tcl testbenches. |

The `quartus_tan` and `quartus_cdb` executables support supersets of the packages supported by the `quartus_sh` executable. You should use the `quartus_sh` executable if you run Tcl scripts with only project management and assignment commands, or need a Quartus II command-line executable with a small memory footprint.

For more information about these command-line executables, refer to the *Command-Line Scripting* chapter in Volume 2 of the *Quartus II Handbook*.

## Command-Line Options (-s, -t, etc)

Table 3–4 lists three command-line options you can use with executables that support Tcl.

*Table 3–4. Command-Line Options Supporting Tcl Scripting*

| Command-Line Option | Description |
|---|---|
| -t <*script file*> [<*script args*>] | Run the specified Tcl script with optional arguments |
| -s | Open the executable in the interactive Tcl shell mode |
| --tcl_eval <*tcl command*> | Evaluate the remaining command-line arguments as Tcl commands. For example, the following command displays help for the project package: `quartus_sh --tcl_eval help -pkg project` |

### Run a Tcl Script

Running an executable with the `-t` option runs the specified Tcl script. You can also specify arguments to the script. Access the arguments through the `argv` variable, or use a package such as `cmdline`, which supports arguments of the following form:

`-`*<argument name> <argument value>*

The `cmdline` package is included in the *<Quartus II directory>*/**bin/tcl_packages/tcllib-1.4/cmdline** directory.

The Quartus II software version 4.0 and earlier does not support the `argv` variable. In those versions of the software, script arguments are in the `quartus(args)` global variable.

### Interactive Shell Mode

Running an executable with the `-s` option starts an interactive Tcl shell session that displays a `tcl>` prompt. Everything you type in the Tcl shell is immediately interpreted by the shell. You can run a Tcl script within the interactive shell with the following command:

`source` *<script name>* [*<script arguments>*] ↵

If a command is not recognized by the shell, it is assumed to be an external command and executed with the `exec` command.

### Evaluate as Tcl

Running an executable with the `--tcl_eval` option causes the executable to immediately evaluate the remaining command-line arguments as Tcl commands. This can be useful if you want to run simple Tcl commands from other scripting languages.

### Using the Quartus II Tcl Console Window

You can run Tcl commands directly in the Quartus II Tcl Console window. To open the window, choose **Utility Windows > Tcl Console** (View menu). By default, the Tcl Console is docked in the bottom-right corner of Quartus II graphical user interface (GUI). Everything typed in the Tcl Console is interpreted by the Quartus II Tcl shell.

☞ The **Quartus II Tcl Console** window supports the Tcl API, used in the Quartus II software version 3.0 and earlier, for backward compatibility with older designs and EDA tools.

Tcl messages appear in the **System** tab (Messages window). Errors and messages written to `stdout` and `stderr` also appear in the Quartus II Tcl Console window.

## Examples

Most chapters in the Quartus II Handbook include information about scripting support. They include feature-specific examples and script information. For scripting help on a specific feature, refer to the corresponding chapter in the handbook.

If you are an advanced Tcl scripting user, you can refer to some Tcl scripts included with the Quartus II software and modify them to suit your needs. The Design Space Explorer (DSE), Quartus II Command-Line and Tcl API reference, and QFlow are written with Tcl and Tk. Files for those scripts are located in the <*Quartus II installation*>/**bin/tcl_scripts** directory.

## Accessing Command-Line Arguments

Virtually all Tcl scripts must accept command-line arguments, such as the name of a project or revision. The global variable `quartus(args)` is a list of the arguments typed on the command-line following the name of the Tcl script. Here is a code example that prints all the arguments in the `quartus(args)` variable:

```
set i 0
foreach arg $quartus(args) {
   puts "The value at index $i is $arg"
   incr i
}
```

If you save these commands in a Tcl script file called **print_args.tcl**, you see the following output when you type this command:

```
quartus_sh -t print_args.tcl my_project 100MHz↵

   The value at index 0 is my_project

   The value at index 1 is 100MHz
```

## Using the cmdline Package

You can use the `cmdline` package included with the Quartus II software for more robust and self-documenting command-line argument passing. The cmdline package supports command-line arguments with the form -*<option>* *<value>*. The following code example uses the `cmdline` package:

```
package require cmdline
variable ::argv0 $::quartus(args)
set options {\
                 { "project.arg" "" "Project name" } \
                 { "frequency.arg" ""  "Frequency" } \
}
set usage "You need to specify options and values"
array set optshash [::cmdline::getoptions ::argv $options $usage]
puts "The project name is $optshash(project)"
puts "The frequency is $optshash(frequency)"
```

If you save those commands in a Tcl script called **print_cmd_args.tcl** you will see the following output when you type this command:

```
quartus_sh -t print_cmd_args.tcl -project my_project -frequency 100MHz↵

                        The project name is my_project
                        The frequency is 100MHz
```

For more information on the cmdline package, refer to the documentation for the package at <*Quartus II installation directory*>**/ bin/tcl_packages/tcllib-1.4/doc/cmdline.html**

## PCreating Projects & Making Assignments

One benefit of the Tcl scripting API is that it is easy to create a script that makes all the assignments for an existing project. You can use the script at any time to restore your project settings to a known state. Choose **Generate Tcl File for Project** (Project menu) to generate a Tcl file with all of your assignments automatically. You can source this file to recreate your project, and you can edit the file to add other commands, such as compiling the design. The file is a good starting point to learn about project management commands and assignment commands.

The following example script is a compilation script for the finite impulse response (FIR) filter example project used in the Quartus II Tutorial. It shows how to set global, location, and instance assignments for a project followed by a complete project compilation using the `::quartus::flow` package.

```
# This Tcl file works with quartus_sh.exe
# This Tcl file will compile the Quartus II tutorial fir_filter
# design
# set the project_name to fir_filter
# set revision to filtref
set project_name fir_filter
set revision_name filtref


# Create a new project and open it
# Project_name is project name
# No need to explicitly require the ::quartus::project package,
# because it's automatically loaded by quartus_sh
if {![project_exists $project_name]} {
   project_new -revision $revision_name $project_name;
} else {
project_open -revision $revision_name $project_name;
}

#------ Make global assignments ------#

# add design files to project
# When the revision name is the same as the project name
# adding design files can be skipped
#set_global_assignment -name "BDF_FILE" "filtref.bdf"
#set_global_assignment -name "VERILOG_FILE" "acc.v"
#set_global_assignment -name "VERILOG_FILE" "accum.v"
#set_global_assignment -name "VERILOG_FILE" "hvalues.v"
#set_global_assignment -name "VERILOG_FILE" "mult.v"
```

```
#set_global_assignment -name "VERILOG_FILE" "state_m.v"
#set_global_assignment -name "VERILOG_FILE" "taps.v"

set_global_assignment -name FAMILY Cyclone

#------ project compilation ------#

# The project is compiled here to see ESB placement following
# what is done in the tutorial
load_package flow
execute_flow -compile

project_close
```

☞ The assignments created or modified while a project is openare not committed to the Quartus II settings files unless you explicitly call `export_assignments` or `project_close` (unless `-dont_export_assignments` is specified). In some cases, such as when running `execute_flow`, the Quartus II software automatically commits the changes.

## Compiling Designs

You can run the Quartus II command-line executables from Tcl scripts either with the included `::quartus::flow` package to run various Quartus II compilation flows, or by running each executable directly.

### *The ::quartus::flow Package*

The `::quartus::flow` package includes two commands for running Quartus II command-line executables, either individually or together in standard compilation sequence. The `execute_module` command allows you to run an individual Quartus II command-line executable. The `execute_flow` command allows you to run some or all of the modules in commonly-used combinations.

Altera recommends using the `::quartus::flow` package instead of using system calls to run compiler executables.

Another way to run a Quartus II command-line executable from the Tcl environment is by using the `qexec` Tcl command, a Quartus II implementation of Tcl's `exec` command. For example, to run the Quartus II technology mapper on a given project, type:

```
qexec "quartus_map <project_name>" ↵
```

When you use the `qexec` command to compile a design, assignments made in the Tcl script (or from the Tcl shell) are not exported to the project database and settings file before compilation. Use the `export_assignments` command or compile the project with commands in the `::quartus::flow` package to ensure assignments are exported to the project database and settings file.

☞ You can also use the Tcl `exec` command to perform command-line system calls. However, Altera recommends using the `qexec` command to avoid limitations with Tcl version 8.3. Whether using `exec` or `qexec`, use caution when making system calls.

You can also run executables directly in a Tcl shell, using the same syntax as at the system command prompt. For example, to run the Quartus II technology mapper on a given project, type the following at the Tcl shell prompt:

`quartus_map` *<project_name>* ↵

## Extracting Report Data

Once a compilation finishes, you may need to extract information from the report to evaluate the results. For example, you may need to know how many device resources the design uses, or whether it meets your performance requirements. The Quartus II Tcl API provides easy access to report data so you don't have to write scripts to parse the text report files.

You can use commands that access report data one row at a time, or a cell at a time. If you know the exact cell or cells you want to access, use the `get_report_panel_data` command and specify the row and column names (or x and y coordinates) and the name of the appropriate report panel. At times you may need to search for data in a report panel. To do this, use a loop that reads the report one row at a time with the `get_report_panel_row` command.

Report panels are arranged hierarchically, and each level of hierarchy is denoted by the string "||" in the panel name. For example, the name of the Fitter Settings report panel is "Fitter||Fitter Settings" because it is in the Fitter folder. Panels at the highest hierarchy level do not use the "||" string. For example, the Flow Settings report panel is named "Flow Settings."

The following example prints the number of failing paths in each clock domain in your design. It uses a loop to access each row of the **Timing Analyzer Summary** report panel. Clock domains are listed in the column named **Type** with the format `Clock Setup:'`*<clock name>*`'`. Other summary information is listed in the **Type** column as well. If the **Type** column matches the pattern "`Clock Setup*`", the script prints the number of failing paths listed in the column named **Failed Paths**.

```
load_report
set report_panel_name "Timing Analyzer||Timing Analyzer Summary"
set num_rows [get_number_of_rows -name $report_panel_name]
set type_column [get_report_panel_column_index -name $report_panel_name \
   "Type"]
set failed_paths_column [get_report_panel_column_index -name \
   $report_panel_name "Failed Paths"]
for {set i 1} {$i < $num_rows} {incr i} {
   set report_row [get_report_panel_row -name $report_panel_name -row $i]
   set row_type [lindex $report_row $type_column]
   set failed_paths [lindex $report_row $failed_paths_column]
   if { [string match "Clock Setup*" $row_type] } {
      puts "$row_type has $failed_paths failing paths"
   }
}
unload_report
```

## Using Collection Commands

Some Quartus II Tcl functions can return very large sets of data which would be inefficient as Tcl lists. These data structures are referred to as collections and the Quartus II Tcl API uses a collection ID to access the collection. There are two Quartus II Tcl commands for working with collection, `foreach_in_collection` and `get_collection_size`. Use the `set` command to assign a collection ID to a variable.

For information about which Quartus II Tcl commands return collection IDs, refer to help for the `foreach_in_collection` command.

### *The foreach_in_collection command*

The `foreach_in_collection` command is similar to the `foreach` Tcl command. Use it to iterate through all elements in a collection. The following example prints all instance assignments in an open project.

```
set all_instance_assignments [get_all_instance_assignments -name *]
foreach_in_collection asgn $all_instance_assignments {
   set to [lindex $asgn 2]
   set name [lindex $asgn 3]
   set value [lindex $asgn 4]
   puts "Assignment to $to: $name = $value"
}
```

### *The get_collection_size command*

Use the `get_collection_size` command to get the number of elements in a collection. The following example prints the quantity of global assignments in an open project:

```
set all_global_assignments [get_all_global_assignments -name *]
set num_global_assignments [get_collection_size $all_global_assignments]
puts "There are $num_global_assignments global assignments in your project"
```

## Timing Analysis

The following example script uses the **quartus_tan** executable to perform a timing analysis on the `fir_filter` tutorial design.

The `fir_filter` design is a two-clock design that requires a base clock and a relative clock relationship for timing analysis. This script first does an analysis of the two-clock relationship and checks for $t_{SU}$ slack between `clk` and `clkx2`. The first pass of the timing analysis discovers a negative slack for one of the clocks. The second part of the script adds a multicycle assignment from `clk` to `clkx2` and re-analyzes the design as a multi-clock, multicycle design.

The script does not recompile the design with the new timing assignments, and timing-driven compilation is not used in the synthesis and placement of this design. New timing assignments are added only for the timing analyzer to analyze the design by using the `create_timing_netlist` and `report_timing` Tcl commands.

☞ You must compile the project before running the script example below.

```
# This Tcl file is to be used with quartus_tan.exe
# This Tcl file will do the Quartus II tutorial fir_filter design
# timing analysis portion by making a global timing assignment and
# creating multi-clock assignments and run timing analysis
# for a multi-clock, multi-cycle design

# set the project_name to fir_filter
# set the revision_name to filtref
set project_name fir_filter
set revision_name filtref

# open the project
# project_name is the project name
project_open -revision $revision_name $project_name;

# Doing TAN tutorial steps this section re-runs the timing
# analysis module with multi-clock and multi-cycle settings
```

```
#------ Make timing assignments ------#

#Specifying a global FMAX requirement (tan tutorial)
set_global_assignment -name FMAX_REQUIREMENT 45.0MHz
set_global_assignment -name CUT_OFF_IO_PIN_FEEDBACK ON

# create a base reference clock "clocka" and specifies the
# following:
#   BASED_ON_CLOCK_SETTINGS = clocka;
#   INCLUDE_EXTERNAL_PIN_DELAYS_IN_FMAX_CALCULATIONS = OFF;
#   FMAX_REQUIREMENT = 50MHZ;
#   DUTY_CYCLE = 50;
# Assigns the reference clocka to the pin "clk"
create_base_clock -fmax 50MHZ -duty_cycle 50 clocka -target clk

# creates a relative clock "clockb" based on reference clock
# "clocka" with the following settings:
#   BASED_ON_CLOCK_SETTINGS = clocka;
#   MULTIPLY_BASE_CLOCK_PERIOD_BY = 1;
#   DIVIDE_BASE_CLOCK_PERIOD_BY = 2;clock period is half the base clk
#   DUTY_CYCLE = 50;
#   OFFSET_FROM_BASE_CLOCK = 500ps;The offset is .5 ns (or 500 ps)
#   INVERT_BASE_CLOCK = OFF;
# Assigns the reference clock to pin "clkx2"
create_relative_clock -base_clock clocka -duty_cycle 50\
-divide 2 -offset 500ps -target clkx2 clockb

# create new timing netlist based on new timing settings
create_timing_netlist

# does an analysis for clkx2
# Limits path listing to 1 path
# Does clock setup analysis for clkx2
report_timing -npaths 1 -clock_setup -file setup_multiclock.tao

# The output file will show a negative slack for clkx2 when only
# specifying a multi-clock design. The negative slack was created
# by the 500 ps offset from the base clock

# removes old timing netlist to allow for creation of a new timing
# netlist for analysis by report_timing
delete_timing_netlist

# adding a multi-cycle setting corrects the negative slack by # adding a
multicycle assignment to clkx2 to allow for more
# set-up time
set_multicycle_assignment 2 -from clk -to clkx2

# create a new timing netlist based on additional timing
# assignments create_timing_netlist

# outputs the result to a file for clkx2 only
```

```
report_timing -npaths 1 -clock_setup -clock_filter clkx2 \
 -file clkx2_setup_multicycle.tao
# The new output file will show a positive slack for the clkx2
project_close
```

## EDA Tool Assignments

You can target EDA tools for a project in the Quartus II software in Tcl by using the `set_global_assignment` Tcl command. To use the default tool settings for each EDA tool, you need only specify the EDA tool to be used. The EDA interfaces available for the Quartus II software cover design entry, simulation, timing analysis and board design tools. More advanced EDA tools such as those for formal verification and resynthesis are supported by their own global assignment.

The global options used for interface to EDA tools in the Quartus II software are shown below:

- `EDA_DESIGN_ENTRY_SYNTHESIS_TOOL`
- `EDA_SIMULATION_TOOL`
- `EDA_TIMING_ANALYSIS_TOOL`
- `EDA_BOARD_DESIGN_TOOL`
- `EDA_FORMAL_VERIFICATION_TOOL`
- `EDA_RESYNTHESIS_TOOL`

By default, these project options are set to `<none>`. Table 3–5 lists the EDA interface options available in the Quartus II software. Enclose interface assignment options that contain spaces in quotation marks.

*Table 3–5. EDA Interface Options*

| Option | Acceptable Values |
|---|---|
| Design Entry<br>`(EDA_DESIGN_ENTRY_SYNTHESIS_TOOL)` | Design Architect<br>Design Compiler<br>FPGA Compiler<br>FPGA Compiler II<br>FPGA Compiler II Altera Edition<br>FPGA Express<br>LeonardoSpectrum<br>LeonardoSpectrum-Altera (Level 1)<br>Synplify<br>Synplify Pro<br>ViewDraw<br>Precision Synthesis<br>Custom |
| Simulation<br>`(EDA_SIMULATION_TOOL)` | ModelSim (VHDL output from the Quartus II software)<br>ModelSim (Verilog HDL output from the Quartus II software)<br>ModelSim-Altera (VHDL output from the Quartus II software)<br>ModelSim-Altera (Verilog HDL output from the Quartus II software)<br>SpeedWave<br>VCS<br>Verilog-XL<br>VSS<br>NC-Verilog (Verilog HDL output from the Quartus II software)<br>NC-VHDL (VHDL output from the Quartus II software)<br>Scirocco (VHDL output from the Quartus II software)<br>Custom Verilog HDL<br>Custom VHDL |
| Timing Analysis<br>`(EDA_TIMING_ANALYSIS_TOOL)` | Prime Time (VHDL output from the Quartus II software)<br>Prime Time (Verilog HDL output from the Quartus II software)<br>Stamp (board model)<br>Custom Verilog HDL<br>Custom VHDL |
| Board level tools<br>`(EDA_BOARD_DESIGN_TOOL)` | Signal Integrity (IBIS)<br>Symbol Generation (ViewDraw) |
| Formal Verification<br>`(EDA_FORMAL_VERIFICATION_TOOL)` | Conformal LEC |
| Resynthesis<br>`(EDA_RESYNTHESIS_TOOL)` | PALACE<br>Amplify |

For example, to generate NC-Sim Verilog simulation output, `EDA_SIMULATION_TOOL` should be set to target NC-Sim Verilog as the desired output, as shown below:

```
set_global_assignment -name eda_simulation_tool\
"NcSim (Verilog HDL output from Quartus II)"
```

The following example shows compilation of the `fir_filter` design files, generating a VHO file output for NC-Sim Verilog simulation:

```
# This script works with the quartus_sh executable
# Set the project name to filtref
set project_name filtref

# Open the Project. If it does not already exist, create it
if [catch {project_open $project_name}] {project_new \
$project_name}

# Set Family
set_global_assignment -name family APEX 20KE

# Set Device
set_global_assignment -name device ep20k100eqc208-1

# Optimize for speed
set_global_assignment -name optimization_technique speed

# Turn-on Fastfit fitter option to reduce compile times
set_global_assignment -name fast_fit_compilation on

# Generate a NC-Sim Verilog simulation Netlist
set_global_assignment -name eda_simulation_tool "NcSim\
(Verilog HDL output from Quartus II)"

# Create an FMAX=50MHz assignment called clk1 to pin clk
create_base_clock -fmax 50MHz -target clk clk1

# Create a pin assignment on pin clk
set_location -to clk Pin_134

# Compilation option 1
# Always write the assignments to the constraint files before
# doing a system call. Else, stand-alone files will not pick up
# the assignments
#export_assignments
#qexec quartus_map <project_name>
#qexec quartus_fit <project_name>
#qexec quartus_asm <project_name>
#qexec quartus_tan <project_name>
#qexec quartus_eda <project_name>

# Compilation option 2 (better)
# Using the ::quartus::flow package, and execute_flow command will
# export_assignments automatically and be equivalent to the steps
# outlined in compilation option 1
load_package flow
```

```
execute_flow -compile

# Close Project
project_close
```

There are custom options available to target other EDA tools. For custom EDA configurations, you can change the individual EDA interface options by making additional assignments.

For a complete list of each EDA setting line available, see "EDA Tool Setting Section (Settings and Configuration Files)" in Quartus II Help.

## Importing LogicLock Functions

The following Tcl script shows how a LogicLock function can be imported into a project. This example is based on the LogicLock tutorial design **topmult**. The script assumes that the Verilog Quartus Mapping file (**.vqm**) named **pipemult.vqm** and the Quartus II Setting File named **pipemult.qsf** have been generated already and placed in the **topmult** project directory. To import LogicLock regions into a project, the **quartus_cdb** executable must be used.

```
# Tcl file created for quartus_cdb to import LogicLock
# pipemult.vqm and pipemult.qsf into the topmult project
# This Tcl script assumes that pipemult.vqm and pipemult.qsf
# have been generated in the lockmult project.

# Since ::quartus::flow is not pre-loaded
# by quartus_cdb, load this package now
# before using the flow Tcl API
# Type "help -pkg flow" to view information
# about the package
load_package flow

set required_fmax 150.00MHz

set project_name topmult

# $project_name contains the project
# name, in this case fir_filter
# Require package ::quartus::project
load_package project

project_open $project_name

#------ Make global assignments ------#

# remove bdf file from project
set_global_assignment -name "BDF_FILE" "pipemult.bdf" -remove
# add VQM file to project
set_global_assignment -name "VQM_FILE" "pipemult.vqm"

# analyze design with VQM file
execute_module -tool map
```

```
# import LogicLock constraints
load_package logiclock
initialize_logiclock

# imports the pipemult.qsf file to the project topmult.qsf
logiclock_import -no_pins

uninitialize_logiclock

# compile entire design
execute_flow -compile

#------ Report Fmax from report ------#
load_package report
load_report
set actual_fmax [get_timing_analysis_summary_results -\
clock_setup clk -actual]
puts ""
puts "-----------------------------------------------------"
puts "Required Fmax: $required_fmax Actual Fmax: $actual_fmax"
puts "-----------------------------------------------------"

project_close
```

For additional information on the LogicLock design methodology, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook*.

## Using the Quartus II Tcl Shell in Interactive Mode

This section presents an example of using the **quartus_sh** interactive shell to make some project assignments and compile the FIR filter tutorial project. This example assumes that you already have the FIR filter tutorial design files in a project directory.

To begin, run the interactive Tcl shell. The command and initial output are shown below:

```
C:\>quartus_sh -s
Info:**************************************************************
Info: Running Quartus II Shell
Info: Version 4.0 Internal Build 131 10/06/2003 SJ Full Version
Info: Copyright (C) 1991-2003 Altera Corporation. All rights reserved.
Info: Quartus is a registered trademark of Altera Corporation in the
Info: US and other countries.  Portions of the Quartus II software
Info: code, and other portions of the code included in this download
Info: or on this CD, are licensed to Altera Corporation and are the
Info: copyrighted property of third parties who may include, without
Info: limitation, Sun Microsystems, The Regents of the University of
Info: California, Softel vdm., and Verific Design Automation Inc.
Info: Warning:  This computer program is protected by copyright law
Info: and international treaties. Unauthorized reproduction or
Info: distribution of this program, or any portion of it, may result
```

```
Info: in severe civil and criminal penalties, and will be prosecuted
Info: to the maximum extent possible under the law.
Info: Processing started: Thu Nov 20 19:54:12 2003
Info:***********************************************************
Info: The Quartus II Shell supports all TCL commands in addition
Info: to Quartus II Tcl commands. All unrecognized commands are
Info: assumed to be external and are run using Tcl's "exec"
Info: command.
Info: - Type "exit" to exit.
Info: - Type "help" to view a list of Quartus II Tcl packages.
Info: - Type "help -pkg <package name>" to view a list of Tcl commands
Info:   available for the specified Quartus II Tcl package.
Info: - Type "help -tcl" to get an overview on Quartus II Tcl usages.
Info: ****************************************************************
tcl>
```

At the Tcl prompt, create a new project called **fir_filter** with a revision name called **filtref** by typing the following command:

```
tcl> project_new -revision filtref fir_filter ↵
```

If the project file and project name are the same, the Quartus II software gives the revision the same name as the project.

Since the revision named **filtref** matches the top-level file, all design files are picked up from the hierarchy tree automatically.

Next, set a global assignment for the device with the following command:

```
tcl> set_global_assignment -name family Cyclone↵
```

To learn more about assignment names that can be used with the `-name` option, see "Settings and Configuration Files Introduction" in Quartus II Help.

For assignment values that contain spaces, the value should be enclosed in quotation marks.

To quickly compile a design, use the `::quartus::flow` package, which properly exports the new project assignments and compiles the design using the proper sequence of the command-line executables. First load the package:

```
tcl> load_package flow↵
1.0
```

For additional help on the `::quartus::flow` package, view the command-line help at the Tcl prompt by typing:

```
tcl> help -pkg  ::quartus::flow↵
```

This sample shows an alternative command and the resulting output:

```
tcl> help -pkg flow
----------------------------------------------------------------

------------------------
Tcl Package and Version:
------------------------

        ::quartus::flow 1.0

------------
Description:
------------

        This package contains the set of Tcl functions
        for running flows or command-line executables.

-------------
Tcl Commands:
-------------

        execute_flow
        execute_module

----------------------------------------------------------------

tcl>
```

This help display gives information on the `::quartus::flow` package and the commands that are available with the package. To read help on the `execute_flow` Tcl command, short help displays the options:

```
tcl> execute_flow -h↵
```

Long help displays additional information and example usage:

```
tcl> execute_flow -long_help↵
```

or

```
tcl> help -cmd execute_flow ↵
```

To perform a full compilation of the FIR filter design, use the `execute_flow` command with the `-compile` option, as shown in the following example:

```
tcl> execute_flow -compile ↵
Info:*************************************************************
Info: Running Quartus II Analysis & Synthesis
Info: Version 4.0 SJ Full Version
Info: Processing started: Mon Nov 18 09:30:47 2003
Info: Command: quartus_map --import_settings_files=on --
export_settings_files=of fir_filter -c filtref
```

```
.
.
.
Info: Writing report file filtref.tan.rpt
tcl>
```

This script compiles the FIR filter tutorial project, exporting the project assignments and running **quartus_map**, **quartus_fit**, **quartus_asm** and **quartus_tan**. This sequence of events is the same as happens when choosing **Start Compilation** (Processing menu) in the Quartus II GUI.

When you are finished with a project, close it using the `project_close` command:

```
tcl> project_close ↵
tcl>
```

Then to exit the interactive Tcl shell, type `exit`.

```
tcl> exit ↵
```

## Getting Help on Tcl & Quartus II Tcl APIs

Quartus II Tcl help allows easy access to information on the Quartus II Tcl commands. To access the help information, type `help` at a command prompt, as shown below (with sample output):

```
tcl> help
-----------------------------------------------------
----------------------------------
Available Quartus II Tcl Packages:
----------------------------------
Loaded                Not Loaded
------------------  -------------------------
::quartus::device   ::quartus::flow
::quartus::misc     ::quartus::report
::quartus::project

* Type "help -tcl"
  to get an overview on Quartus II Tcl usages.
-----------------------------------------------------
tcl>
```

Using the `-tcl` option with `help` displays an introduction to the Quartus II Tcl API that focuses on how to get help for Tcl commands (short help and long help) and Tcl packages.

Table 3–6 summarizes the help options available in the Tcl environment.

*Table 3–6. Help Options Available in the Quartus II Tcl Environment (Part 1 of 2)*

| Help Command | Description |
|---|---|
| `help` | To view a list of available Quartus II Tcl packages, loaded and not loaded. |
| `help -tcl` | To view a list of commands used to load Tcl packages and access command-line help. |
| `help -pkg` *<package_name>* `[-version` *<version number>*`]` | To view help for a specified Quartus II package thatincludes the list of available Tcl commands. For convenience, you can omit the `::quartus::` package prefix, and type `help -pkg` *<package name>* ↵.<br>If you do not specify the `-version` option, help for the currently loaded package is displayed by default. If the package for which you want help is not loaded, help for the latest version of the package is displayed by default.<br><br>Examples:<br>`help -pkg ::quartus::p` ↵<br>`help -pkg ::quartus::project` ↵<br>`help -pkg project rhelp -pkg project -version 1.0` ↵ |
| *<command_name>* `-h`<br>or<br>*<command_name>* `-help` | To view short help for a Quartus II Tcl command for which the package is loaded.<br><br>Examples:<br>project_open -h ↵<br>project_open -help ↵ |
| `package require ::quartus::`*<package name>* `[`*<version>*`]` | To load a Quartus II Tcl package with the specified version. If *<version>* is not specified, the latest version of the package is loaded by default.<br><br>Example:<br>`package require ::quartus::project 1.0` ↵<br><br>This command is similar to the `load_package` command. The advantage of using `load_package` is that you can alternate freely between different versions of the same package.<br>Type *<package name>* `[-version` *<version number>*`]` ↵ to load a Quartus II Tcl package with the specified version. If the `-version` option is not specified, the latest version of the package is loaded by default.<br><br>Example:<br>`load_package ::quartus::project -version 1.0` ↵ |

*Table 3–6. Help Options Available in the Quartus II Tcl Environment (Part 2 of 2)*

| Help Command | Description |
|---|---|
| `help -cmd` <*command name*> `[-version` <*version number*>`]`<br>or<br><*command name*> `-long_help` | To view long help for a Quartus II Tcl command. Only <*command name*> `-long_help`"requires that the associated Tcl package is loaded.<br>If you do not specify the `-version` option, help for the currently loaded package is displayed by default.<br>If the package for which you want help is not loaded,help for the latest version of the package is displayedby default.<br><br>Examples:<br>`project_open -long_help` ↵<br>`help -cmd project_open` ↵<br>`help -cmd project_open -version 1.0` ↵ |
| `help -examples` | To view examples of Quartus II Tcl usage. |
| `help -quartus` | To view help on the predefined global Tcl array that can be accessed to view information about the Quartus II executable that is currently running. |
| `quartus_sh --qhelp` | To launch the Tk viewer for Quartus II command-line help and display help for the command-line executables and Tcl API packages. See "The Tcl/Tk GUI Help Interface" on page 3–28 for more information. |

There are two types of help for Tcl commands:

- For information on the usage and a brief description of a Tcl command type, use the `-help` option. (The `-h` command-line option may be used instead of `-help`, if preferred.) If the Tcl command is part of a Tcl package that is not loaded, using the `-help` option returns "`invalid command name`" as an error message.
- For more detailed help on a given Tcl command, use the `-long_help` option or type `help -cmd` <*Tcl command name*>. If the Tcl command is part of a Tcl package that is not loaded, typing <*command name*> `-long_help` returns the error message "`invalid command name.`"

☞ Using the `-cmd` option does not require that the specific Tcl command be loaded. Only the `-long_help` option requires that the relevant Tcl package be loaded.

### *The Tcl/Tk GUI Help Interface*

For a complete list of package and commands available with the Quartus II software, open the help browser that lists all Quartus II command-line executables and Tcl API packages and their respective commands. To open the help browser, type the following command at a system command prompt:

```
C:\> quartus_sh --qhelp
```

This runs a Tcl/Tk script that provides help for Quartus II Command-line executables and Tcl API packages and commands.

For more information on this utility, see the *Command-Line Scripting* chapter in Volume 2 of the *Quartus II Handbook*.

## Quartus II Legacy Tcl Support

The Quartus II software version 3.0 and later command-line executables do not support the Tcl commands used in previous versions of the Quartus II software. These commands are supported in the GUI by using the Quartus II Tcl console or by using the `quartus_cmd` executable at the command prompt. If you source Tcl scripts developed for an earlier version of the Quartus II software using either of these methods, the project assignments are ported to the project database and settings file. You can then use the command-line executables to process the resulting project. This may be necessary if you create a Tcl file using a third-party EDA tool that does not generate Tcl scripts for the most recent version of the Quartus II software.

Altera recommends creating all new projects and Tcl scripts with the latest version of the Quartus II Tcl API.

## References

For more information on using Tcl, see the following sources:

- *Practical Programming in Tcl and Tk*, Brent B. Welch
- *Tcl and the TK Toolkit*, John Ousterhout
- *Effective Tcl/TK Programming*, Michael McLennan and Mark Harrison
- Tcl Developer Xchange at **http://tcl.activestate.com**

# 4. Quartus II Project Management

qii52012-1.0

## Introduction

FPGA designs once required just one or two engineers, but today's larger and more sophisticated FPGA designs are often developed by several engineers and are constantly changing throughout the project. To ensure efficient design coordination, designers are required to keep track of their changes to the project. To help designers manage their FPGA designs, the Quartus® II software provides the Revisions, Copy Project, and Version-Compatible Database features.

In the Quartus II software, a revision is one set of assignments and settings. A project can have multiple revisions, each with their own set of assignments and settings. Creating multiple revisions allows you to change assignments and settings while preserving previous results.

A version is a Quartus II project that includes one set of design files and one or more revisions (assignments and settings for your project). Creating multiple versions with the Copy Project feature allows you to edit a copy of your design files while preserving the original functionality of your design.

The Version-Compatible Database feature allows databases to be compatible across different versions of the Quartus II software, thus avoiding unnecessary recompilations.

## Using Revisions with Your Design

The Revisions feature allows you to create a new set of assignments and settings for your design without losing your previous assignments and settings. This ability allows you to explore different assignments and settings for your design and then compare the results.

There are several ways to use the revisions feature. The first method is to create a new revision of your design that is not based on any previous revision. For example, early in your design you may want to create a revision containing assignments that target area optimization and another revision containing assignments that target $f_{MAX}$ optimization.

The second method is to create a new revision based on an existing revision and then try new settings and assignments in the new revision. Your new revision will already include all the assignments and settings made in the previous revision. Working on a revision based on another revision allows you to revert to the original revision if you are not satisfied with the results from the new revision.

The third method is to compare different compilation results from different revisions, select the revision that best meets your design requirements, create a new revision based on the best revision, and perform further optimizations until the design meets all design requirements.

## Creating and Deleting Revisions

All Quartus II assignments and settings are stored in the Quartus Settings File (QSF). Each time you create a new revision the Quartus II software creates a new QSF and adds the name of the new revision to the list of revisions in the Quartus Project File (QPF). Revisions are managed with the **Revisions** dialog box, allowing you to set the current revision, create, delete, and compare revisions in a project.

To create a revision:

1. If you have not already done so, create a new project or open an existing project.

2. Choose **Revisions** (Project menu).

3. If you want to base the new revision on an existing revision for the current design, select the existing revision in the **Revisions** list.

4. Click **Create**.

5. In the **Create Revision** dialog box, type the name of the new revision in the **Revision name** box.

6. If you want to base the new revision on an existing revision for the current design, and you did not select the revision in Step 3, then select the revision in the **Based on revision** list.

   *or*

   If you do not want to base the new revision on an existing revision for the current design, select the blank entry in the **Based on revision** list.

7. If you want, edit the description of the revision in the **Description** box.

8. If you based the new revision on an existing revision for the current design, and you want the new revision to contain the database information from the existing revision, turn on **Copy database**.

9. If you want to specify the new revision as the current revision, turn on **Set as current revision**.
10. Click **OK**.
11. In the **Revisions** dialog box, click **Close**.

To delete a revision that is not a design's current revision:

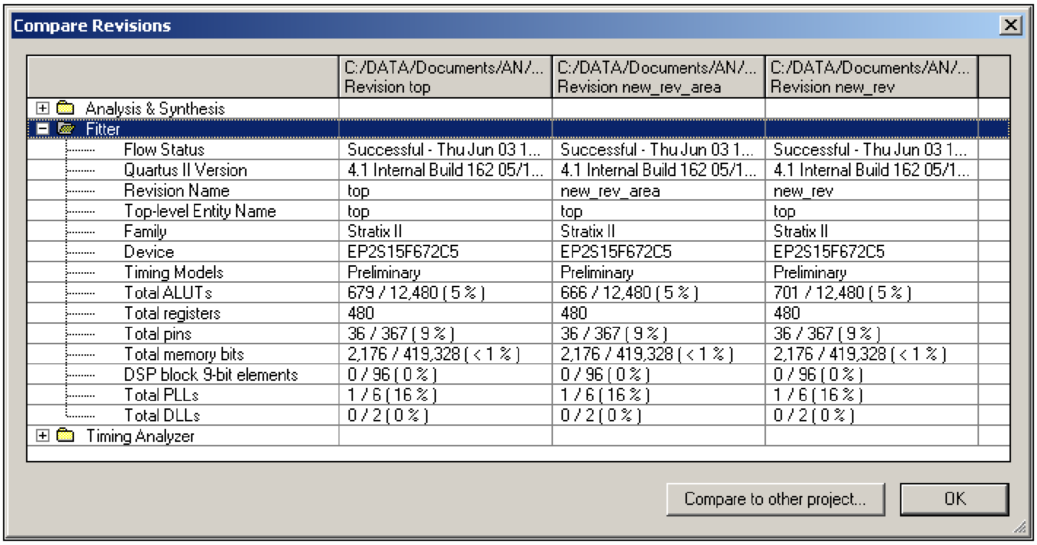
1. If you have not already done so, open an existing project.
2. Choose **Revisions** (Project menu).
3. In the **Revisions** list, select the revision you want to delete.
4. Click **Delete**.
5. Click **Close**.

☞ To delete the current revision, select a different revision as the current revision first.

## Comparing Revisions

You can compare the results of multiple revisions side by side with the **Compare Revisions** dialog box. To compare all revisions in a single window, click **Compare** in the **Revisions** dialog box (Project menu). In the **Compare Revisions** dialog box (see Figure 4–1), the results of each revision in three categories (Analysis & Synthesis, Fitter, and Timing Analyzer) are compared side by side.

*Figure 4–1. Compare Revisions Dialog Box*



Compare Revisions

| | C:/DATA/Documents/AN/... Revision top | C:/DATA/Documents/AN/... Revision new_rev_area | C:/DATA/Documents/AN/... Revision new_rev |
|---|---|---|---|
| ⊞ Analysis & Synthesis | | | |
| ⊟ Fitter | | | |
| Flow Status | Successful - Thu Jun 03 1... | Successful - Thu Jun 03 1... | Successful - Thu Jun 03 1... |
| Quartus II Version | 4.1 Internal Build 162 05/1... | 4.1 Internal Build 162 05/1... | 4.1 Internal Build 162 05/1... |
| Revision Name | top | new_rev_area | new_rev |
| Top-level Entity Name | top | top | top |
| Family | Stratix II | Stratix II | Stratix II |
| Device | EP2S15F672C5 | EP2S15F672C5 | EP2S15F672C5 |
| Timing Models | Preliminary | Preliminary | Preliminary |
| Total ALUTs | 679 / 12,480 ( 5 % ) | 666 / 12,480 ( 5 % ) | 701 / 12,480 ( 5 % ) |
| Total registers | 480 | 480 | 480 |
| Total pins | 36 / 367 ( 9 % ) | 36 / 367 ( 9 % ) | 36 / 367 ( 9 % ) |
| Total memory bits | 2,176 / 419,328 ( < 1 % ) | 2,176 / 419,328 ( < 1 % ) | 2,176 / 419,328 ( < 1 % ) |
| DSP block 9-bit elements | 0 / 96 ( 0 % ) | 0 / 96 ( 0 % ) | 0 / 96 ( 0 % ) |
| Total PLLs | 1 / 6 ( 16 % ) | 1 / 6 ( 16 % ) | 1 / 6 ( 16 % ) |
| Total DLLs | 0 / 2 ( 0 % ) | 0 / 2 ( 0 % ) | 0 / 2 ( 0 % ) |
| ⊞ Timing Analyzer | | | |

You can also compare revisions from another project. To do this, click **Compare to other project** in the **Compare Revisions** dialog box and select a QPF to compare with.

## Creating Different Versions of Your Design

Managing different versions of design files in a large project can become difficult. To assist in this task, the Quartus II software provides utilities to copy and save different versions of your project. Creating a version of your project includes copying all your design files, your Quartus II settings file, and all your associated revisions.

Creating a new version of your project with the Quartus II software involves creating a copy of your project and then editing your design files. For example, you have a design that is compatible with a 32-bit data bus and now you need to create a new version of the design to interface with a 64-bit data bus. To solve this problem, create a new version of your project with the **Copy Project** command (Project menu), and make the necessary changes to your design files.

Creating a new version of your project with an Electronic Data Interchange Format (EDIF) or Verilog Quartus Mapping (VQM) netlist from a third-party EDA synthesis tool involves creating a copy of your project and then replacing the previous netlist file with the newly generated netlist file. Use the **Copy Project** command (Project menu) to create a copy of your design and use the **Add/Remove Files from Project** command (Project menu) to add and remove design files.

To create a new version of your project, use the **Copy Project** command (Project menu).

1. Choose **Copy Project** (Project menu). This opens the Copy Project dialog box (see Figure 4–2).

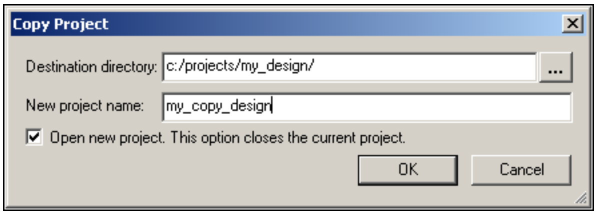
2. **Browse** or type the path to your new project in the **Destination directory** box.

3. Type the new project name in the **New project name** box.

4. To open the new project immediately, turn on the **Open new project in Quartus II** option.

5. Click **OK**.

*Figure 4–2. Copy Project Dialog Box*



Destination directory: c:/projects/my_design/

New project name: my_copy_design

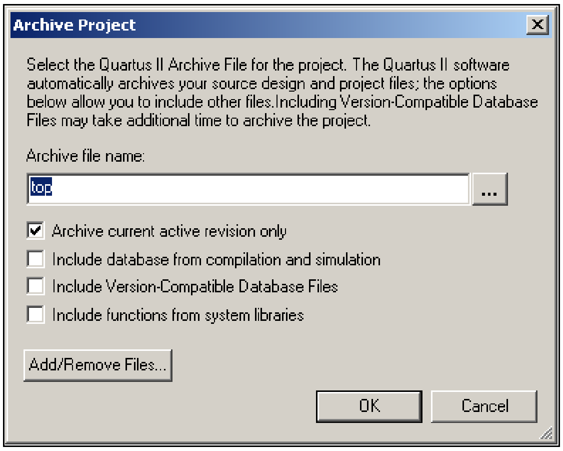
☑ Open new project. This option closes the current project.

## Archiving Projects

You can use the Quartus II Archive Project feature to create a single compressed Quartus II Archive File (**.qar**) of your project containing all your design, project, and settings files. You also have the option to include additional files and the project database. The QAR file contains all the files required to perform a full compilation to restore the original results.

A single project can contain hundreds of files, and it may be difficult to transfer a project between engineers. The archive file generated by the Archive Project feature (see Figure 4–3) can easily be shared between engineers.

*Figure 4–3. Archive Project Dialog Box*



Select the Quartus II Archive File for the project. The Quartus II software automatically archives your source design and project files; the options below allow you to include other files.Including Version-Compatible Database Files may take additional time to archive the project.

Archive file name: top

[✓] Archive current active revision only
[ ] Include database from compilation and simulation
[ ] Include Version-Compatible Database Files
[ ] Include functions from system libraries

To archive a project:

1. If you have not already done so, create a new project or open an existing project.

2. If you want, analyze or compile the design.

3. Choose **Archive Project** (Project menu).

4. Type a name for the Quartus II Archive File (**.qar**) in **Archive file name**, or select a name with **Browse (...)**.

5. To include the outputs of compilation and simulation, turn on **Include database from compilation and simulation**.

6. To include the Version-Compatible Database Files, turn on **Include Version-Compatible Database Files**.

7. To include functions from system libraries, turn on **Include functions from system libraries**.

8. Click **Add/Remove Files** to edit the contents of the QAR file.

9. Click **OK**.

Altera® recommends that you perform Analysis and Synthesis before archiving a project to ensure that all design files are located and archived.

To restore an archived project:

1. Choose **Restore Archived Project** (Project menu).

2. In the **Archive name** box, type the path and file name of the Quartus II Archive File (**.qar**) you wish to restore, or select a QAR File with **Browse (...)**.

3. In the **Destination folder** box, type or select the path of the folder into which you wish to restore the contents of the QAR File, or select a folder with **Browse (...)**.

4. Click **Show log** to view the Quartus II Archive Log File (**.qarlog**) for the project you are restoring from the QAR File.

5. Click **OK**.

6. If necessary, recompile the project.

## Version-Compatible Databases

In the past, compilation databases were locked to the current version of the Quartus II software. With the introduction of the Version-Compatible Database feature in the Quartus II software version 4.1, you can export a version-compatible database and import it into a later version of the Quartus II software. For example, with the same set of design files, you can export a database generated from the Quartus II software version 4.1 and import it into the Quartus II software versions 4.1 and later without having to recompile your design.

Perform the following steps to export a version-compatible database:

1. Choose **Export Database** (Project menu).

2. **Browse** or type in a path in the **Export Directory** box.

3. Click **OK.**

Perform the following steps to import a version-compatible database:

1. Choose **Import Database** (Project menu).

2. **Browse** to the directory to which the database was previously exported. The default directory is <*project name*>**\export_db**.

3. Click **OK**.

To save the database in a version-compatible format during every compilation, perform the following steps:

1. Choose **Settings** (Assignments menu).
2. Select the **Compilation Process** page.
3. Turn on the **Save the database in a version-compatible format** option.
4. **Browse** to the directory in which you want to save the database.
5. Click **OK**.

## Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run some of these procedures at a command prompt.

For detailed information about specific scripting command options and Tcl API packages, type `quartus_sh --qhelp` at a system command prompt to run the Quartus II Command-Line and Tcl API Help browser.

For more information on Quartus II scripting support, including examples, refer to the *Tcl Scripting* and *Command-Line Scripting* chapters of the *Quartus II Handbook*.

### Managing Revisions

You can use the following commands to create and manage revisions. For more information about managing revisions, including creating and deleting revisions, setting the current revision, and getting a list of revisions, see "Creating and Deleting Revisions" on page 4–2.

#### *Creating Revisions*

The following Tcl command creates a new revision called `speed_ch` based on a revision called `chiptrip` and sets it as the current revision. The `-based_on` and `-set_current` options are optional.

```
create_revision speed_ch -based_on chiptrip -set_current
```

#### *Setting the Current Revision*

Use the following Tcl command to set the current revision:

`set_current_revision` <*revision name*>

#### *Getting a List of Revisions*

Use the following Tcl command to get a list of revisions in the opened project:

```
get_project_revisions
```

#### *Deleting Revisions*

Use the following Tcl command to delete a revision:

delete_revision *<revision name>*

### Archiving Projects

You can archive projects with a Tcl command or with a command run at the system command prompt. For more information about archiving projects, see "Archiving Projects" on page 4–5.

The following Tcl command creates a project archive with the default settings and overwrites the specified archived file if it already exists:

```
project_archive archive.qar -overwrite
```

Type the following command at a command prompt to create a project archive:

```
quartus_sh --archive top ↵
```

### Restoring Archived Projects

You can restore archived projects with a Tcl command or with a command run at a command prompt. For more information about restoring archived projects, see page 4–7.

The following Tcl command restores the project archive named **archive.qar** in the subdirectory named **restored** and overwrites existing files:

```
project_restore archive.qar -destination restored -overwrite
```

Type the following command at a command prompt to restore a project archive:

```
quartus_sh --restore archive.qar ↵
```

### Importing and Exporting Version-Compatible Databases

You can import and export version-compatible databases with either a Tcl command or a command run at a command prompt. For more information about importing and exporting version-compatible databases, see "Version-Compatible Databases" on page 4–7.

The `flow` package and the `database_manager` package contain commands to manage version-compatible databases.

Use the following commands from the `database_manager` package to import or export version-compatible databases.

```
export_database <directory>
import_database <directory>
```

Use the following commands available in the `flow` package to import or export version-compatible databases. If you use the `flow` package, you will also need to specify the database directory variable name.

```
set_global_assignment \
-name VER_COMPATIBLE_DB_DIR <directory>
execute_flow –flow export_database
execute_flow –flow import_database
```

Add the following Tcl commands to automatically generate version-compatible databases after every compilation:

```
set_global_assignment \
-name AUTO_EXPORT_VER_COMPATIBLE_DB ON \
set_global_assignment \
-name VER_COMPATIBLE_DB_DIR <directory>
```

The `quartus_cdb` and the `quartus_sh` executables provide commands to manage version-compatible databases:

```
quartus_cdb <project> -c <revision> \
--export_database=<directory>
quartus_cdb <project> -c <revision> \
--import_database=<directory>

quartus_sh –flow export_database <project> -c <revision>
quartus_sh –flow import_database <project>  -c <revision>
```

## Conclusion

Throughout the development of a successful FPGA design, designers often try different settings and versions of their designs. The **Revisions** feature in the Quartus II software facilitates the creation and management of revisions, which are sets of different assignments and settings. The **Copy Project** feature allows you to create a new version of your design by copying a set of design files and one or more revisions.

The Quartus II Version-Compatible Database feature saves compilation time when moving to updated versions of the Quartus II software. These features in the Quartus II software help facilitate efficient management of your design to accommodate today’s more sophisticated FPGA designs.

# Section II. Device & Board Utilities

This section describes the design flow to assign and analyze pin-outs using the **Start I/O Assignment Analysis** command in the Quartus® II software, both with and without a complete design.

This section includes the following chapter:

- Chapter 5, I/O Assignment Planning & Analysis

## Revision History

The table below shows the revision history for Chapter 5.

| Chapter(s) | Date / Version | Changes Made |
|---|---|---|
| 5 | June 2004 v2.0 | ● Scripting support section added.<br>● Updated coding examples. |
| | Feb. 2004 v1.0 | Initial release |

# 5. I/O Assignment Planning & Analysis

qii52004 - 2.0

## Introduction

Today's FPGAs support multiple I/O standards and have high pin counts. You must be able to make pin assignments efficiently for designs in these advanced devices. You also need the ability to easily check the legality of the pin assignments to ensure that the pin-out does not violate any board layout guidelines such as pin spacing and current draw limitations.

This chapter describes the design flow to assign and analyze pin-outs using the **Start I/O Assignment Analysis** command in the Quartus® II software, both before and after completion of your design.

## I/O Assignment Planning & Analysis

Time-to-market constraints means that board layout is often done in parallel with, or even prior to, creating your design. Therefore, checking the legality of your I/O assignments early in the design process is often a requirement.

The **Start I/O Assignment Analysis** command in the Quartus II software provides the capability of checking your I/O assignments early in the process. You can use this command to check the legality of your pin assignments before, during, or after completion of your design. If design files are available, you can use this command to perform more thorough legality checks on your design's I/O pins and surrounding logic. These checks include proper reference voltage pin usage, valid pin location assignments, and acceptable mixed I/O standards.

The **Start I/O Assignment Analysis** command is available for the Stratix® II, Stratix GX, Stratix, MAX® II, and Cyclone™ device families.

## I/O Assignment Planning & Analysis Design Flows

The I/O assignment planning and analysis design flows depend on whether your project contains design files.

- When the board layout must be complete before starting the FPGA design, use the flow shown in Figure 5–1. This flow does not need design files and checks the legality of your pin assignments.
- With a complete design, use the flow shown in Figure 5–3 on page 5–5. This flow uses design files to thoroughly check the legality of your pin assignments and surrounding logic. For more information on creating assignments, see the *Assignment Editor* chapter in Volume 2 of the *Quartus II Handbook.*

Each flow involves creating pin assignments, running the analysis, and reviewing the report file.

Altera suggests that you run the analysis each time you add or modify a pin-related assignment. You can use the **Start I/O Assignment Analysis** command repeatedly since it completes in a short time.

The analysis checks pin assignments and surrounding logic for illegal assignments and violations of board layout rules. For example, the analysis checks whether your pin location supports the I/O standard assigned, current strength, supported $V_{REF}$ voltages, and whether a PCI diode is permitted.

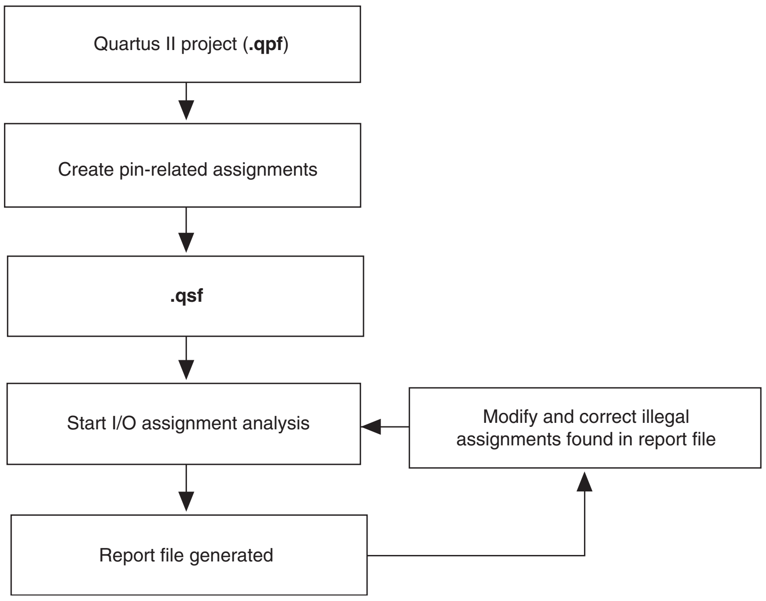
Along with the pin-related assignments, the **Start I/O Assignment Analysis** command also checks blocks that directly feed or are fed by a pin such as a phase-locked loop (PLL), low-voltage differential signal (LVDS), or gigabit transceiver block.

## Design Flow without Design Files

During the early stages of development of an FPGA device, board layout engineers may request preliminary or final pin-outs. It is time consuming to manually check to see whether the pin-outs violate any design rules. Instead, you can use the **Start I/O Assignment Analysis** command to quickly perform basic checks on the legality of your pin assignments.

☞ Without a complete design, the analysis performs limited checks and cannot guarantee that your assignments did not violate design rules.

*Figure 5–1. Assigning & Analyzing Pin-outs without Design Files*


Quartus II project (.qpf)
Create pin-related assignments
.qsf
Start I/O assignment analysis
Modify and correct illegal assignments found in report file
Report file generated

You can assign and analyze pin-outs using the **Start I/O Assignment Analysis** command without design files by following these steps:

1. Create a Quartus II project.

2. Use the **Assignment Editor** or Tcl commands to create pin locations and related assignments. For the I/O assignment analysis to determine the type of a pin, you must reserve your I/O pins. See "Creating I/O Assignments" on page 5–6.

3. Choose **Start > Start I/O Assignment Analysis** (Processing menu) to start the analysis.

4. View the messages in the **Compilation Report** window, **Fitter** report file (<*project name*>.**fit.rpt**) or in the **Messages** window.

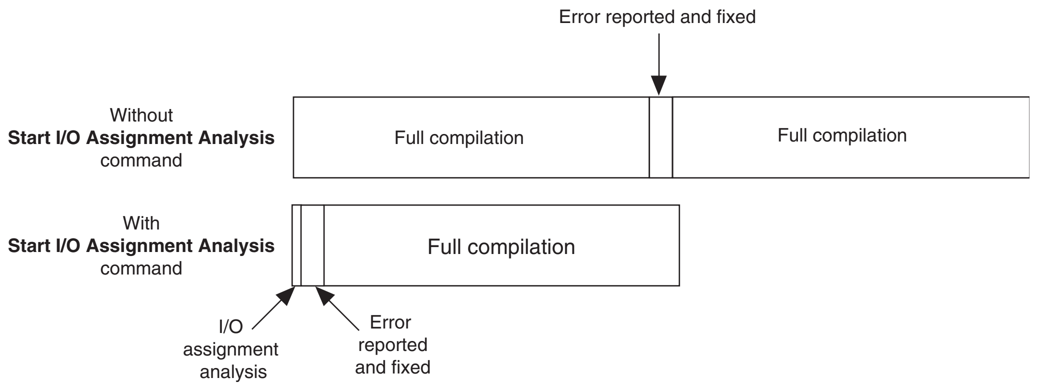
5. Correct any errors and violations reported by the I/O assignment analysis.

6. Rerun the **Start I/O Assignment Analysis** command until all errors are corrected.

## Design Flow with Complete or Partial Design Files

During a full compilation, the Quartus II software does not report illegal pin assignments until the fitter stage. To validate pin assignments sooner, you can run the **Start I/O Assignment Analysis** command after performing analysis and synthesis and before performing a full compilation. Typically, the analysis takes a short time. Figure 5–2 describes the benefits of using the **Start I/O Assignment Analysis** command.

***Figure 5–2. Saving Compilation Time with the Start I/O Assignment Analysis Command***



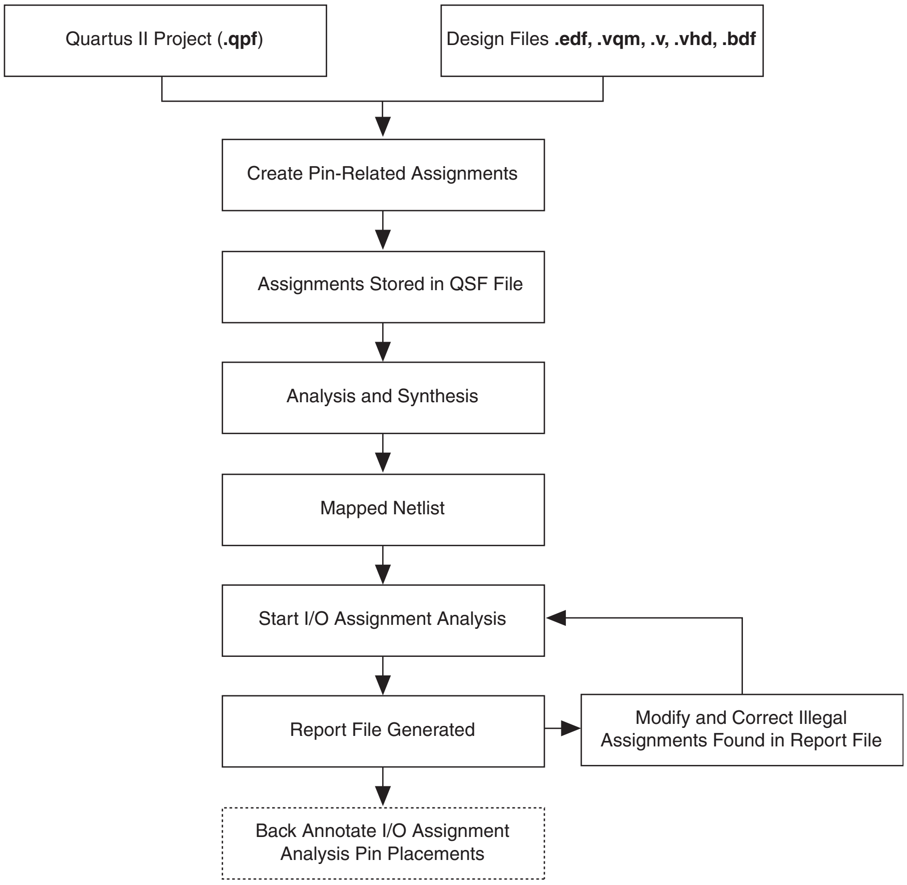
The rules that can be checked by the I/O assignment analysis depends on the completeness of the design. With a complete design, the **Start I/O Assignment Analysis** command thoroughly checks the legality of all pin-related assignments. With a partial design, the **Start I/O Assignment Analysis** command checks the legality of those pin-related assignments for which it has enough information.

For example, you might assign a clock to a user I/O pin instead of assigning it to a dedicated clock pin. You design the clock to drive a PLL that has not yet been instantiated in the design. Because the **Start I/O Assignment Analysis** command is unaware of the logic that the pin drives, it is not able to check that only a dedicated clock input pin can drive the clock port of a PLL.

If you have a partial design, Altera recommends that you provide as much of the design as possible, especially logic that connects to pins, to obtain better coverage.   For example, if your design includes PLLs or LVDS blocks, you should include these MegaWizard® files in your project for analysis.

☞ A top-level wrapper file would be an example of a partial design.

*Figure 5–3. Assigning & Analyzing Pin-outs with Design Files*


Quartus II Project (.qpf)
Design Files .edf, .vqm, .v, .vhd, .bdf
Create Pin-Related Assignments
Assignments Stored in QSF File
Analysis and Synthesis
Mapped Netlist
Start I/O Assignment Analysis
Report File Generated
Modify and Correct Illegal Assignments Found in Report File
Back Annotate I/O Assignment Analysis Pin Placements

Use the following steps to assign and analyze pin-outs using the **Start I/O Assignment Analysis** command with design files:

1. Create a Quartus  II project and include your design files in the project.

2. Create pin-related assignments with the **Assignment Editor**.

3. Choose **Start > Start Analysis & Synthesis** (Processing menu) to generate an internal mapped netlist.

4. Choose **Start > Start I/O Assignment Analysis** (Processing menu) to start the analysis.

5. View the messages in the report file or in the **Messages** window

6. Correct any errors and violations reported

7. Rerun the **Start I/O Assignment Analysis** command until all errors are corrected.

## Inputs Used for I/O Assignment Analysis

The **Start I/O Assignment Analysis** command reads an internal mapped netlist and a Quartus II Settings File (**.qsf**). All assignments are stored in the single QSF.

If you do not have any design files then the **Start I/O Assignment Analysis** command reads only the QSF.

If you have a partial or complete design, the **Start I/O Assignment Analysis** command reads in the QSF and the mapped netlist file.

### Creating I/O Assignments

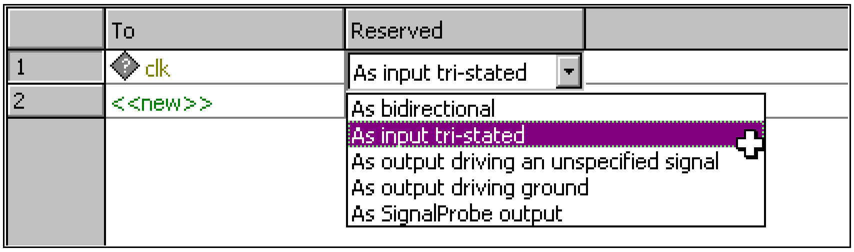
You can create pin-related assignments using the following features:

- Assign SignalProbe Pins dialog box
- Assignment Editor
- Tcl Commands
- Floorplan Editor

### Reserving Pins

If you do not have any design files, you must create reserved pin assignments in addition to your other pin-related assignments. Reserving pins is necessary so that the **Start I/O Assignment Analysis** command will understand the pin type (input, output, or bidirectional) and correctly analyze the pin. You can reserve a pin by choosing **Assignment Editor** (Assignments menu), and selecting **Reserved Pin** from the Category list. In the spreadsheet interface, type in the pin name and select from the reserved list (see Figure 5–4).

*Figure 5–4. Reserving a Pin with the Assignment Editor*



|  | To | Reserved |  |
|---|---|---|---|
| 1 | clk | As input tri-stated |  |
| 2 | <<new>> |  |  |

For more information on using the **Assignment Editor**, see the *Assignment Editor* chapter in Volume 2 of the *Quartus II Handbook*.

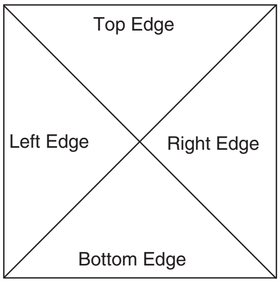
## Location Assignments

You can assign a location to your pins using the **Assignment Editor**. Choose **Assignment Editor** (Assignments menu) to open the Assignment Editor. Select the **pins category** from the **Category** list. In the spreadsheet interface, type in the pin name and select a location from the location list. For Stratix II, Stratix GX, Stratix, and Cyclone devices, you can also assign a pin to an I/O Bank or Edge location.

It is common to place a group of pins (buses) with compatible I/O standards in the same bank. For example, two buses with two I/O standards, 2.5 V and SSTL-II can be placed in the same I/O bank.

An easy way to place large buses that exceed the pins available in a particular I/O bank is to use Edge location assignments. You can also use Edge location assignments to improve circuit board routing ability of large buses, since they are close together near an edge. Figure 5–5 shows the Altera device package edges.

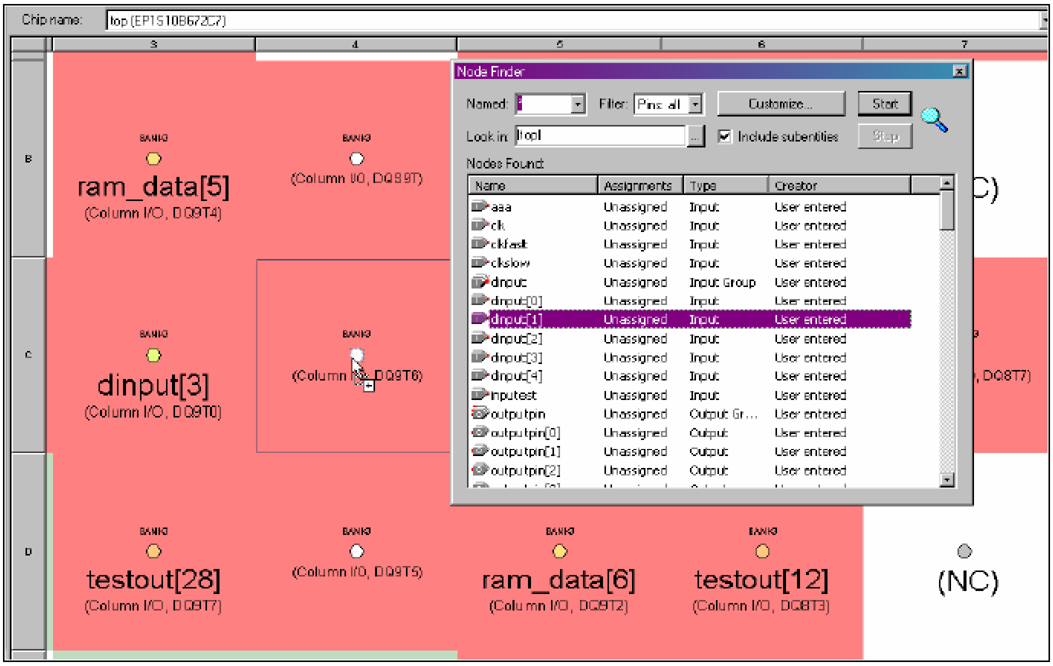
*Figure 5–5. Package View of the Four Edges on an Altera Device*



## Assignments with the Floorplan Editor

You can also make pin location assignments with the **Floorplan Editor**. Open the **Timing Closure Floorplan** by choosing **Timing Closure Floorplan** (Assignments menu). In the **Timing Closure Floorplan**, you can change the view between the package view and the interior cell view from the View menu. You can use the top and bottom package view to view the pins in the desired package. You can find the pad separation between two pins with the interior cell view. In both views, you can drag and drop pins from the Node Finder or from a graphic design file (GDF) or block design file (BDF) file into the desired pin or bank (see Figure 5–6).

*Figure 5–6. Creating Pin Location Assignments with the Node Finder & the Timing Closure Floorplan*



| Name | Assignments | Type | Creator |
|---|---|---|---|
| aaa | Unassigned | Input | User entered |
| clk | Unassigned | Input | User entered |
| clkfast | Unassigned | Input | User entered |
| clkslow | Unassigned | Input | User entered |
| dinput | Unassigned | Input Group | User entered |
| dinput[0] | Unassigned | Input | User entered |
| dinput[1] | Unassigned | Input | User entered |
| dinput[2] | Unassigned | Input | User entered |
| dinput[3] | Unassigned | Input | User entered |
| dinput[4] | Unassigned | Input | User entered |
| inputest | Unassigned | Input | User entered |
| outputpin | Unassigned | Output Gr... | User entered |
| outputpin[0] | Unassigned | Output | User entered |
| outputpin[1] | Unassigned | Output | User entered |
| outputpin[2] | Unassigned | Output | User entered |

## Generating a Mapped Netlist

The **Start I/O Assignment Analysis** command uses a mapped netlist, if available, to identify the pin type and the surrounding logic.

Choose **Start > Start Analysis & Synthesis** (Processing menu) to generate a mapped netlist. You can also use the **quartus_map** executable to run analysis and synthesis.

The mapped netlist is stored internally in the Quartus II database.

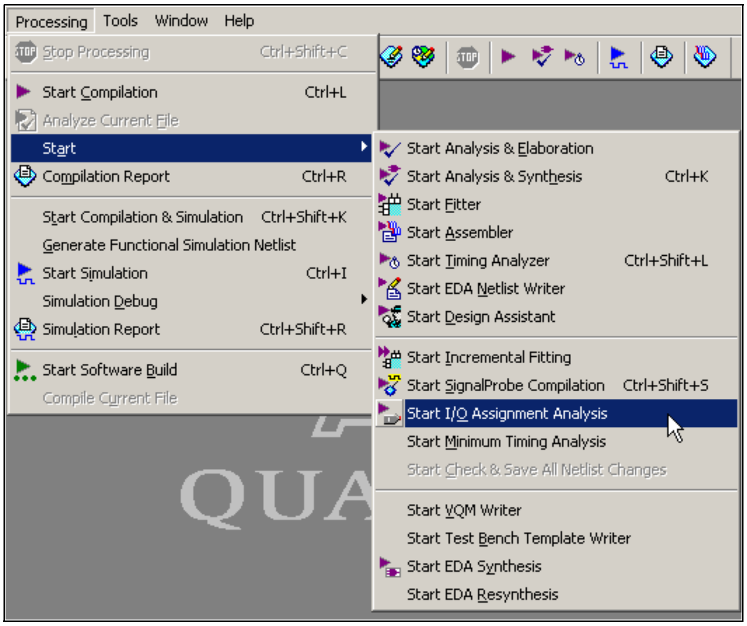
## Running the I/O Assignment Analysis

You can run the **Start I/O Assignment Analysis** command from the Quartus II software menu (see Figure 5–7) or from the command prompt. Choose **Start** > **Start I/O Assignment Analysis** (Processing menu) or type the following command in your project directory.

```
quartus_fit <project-name> --check_ios ↵
```

Running the **Start** I/**O Assignment Analysis** command overwrites any previous fitter database. You can still view the previous compilation report text file.

*Figure 5–7. I/O Assignment Analysis Command from the Quartus II Software Menu*


Processing
Tools
Window
Help
Stop Processing Ctrl+Shift+C
Start Compilation Ctrl+L
Analyze Current File
Start
Compilation Report Ctrl+R
Start Compilation & Simulation Ctrl+Shift+K
Generate Functional Simulation Netlist
Start Simulation Ctrl+I
Simulation Debug
Simulation Report Ctrl+Shift+R
Start Software Build Ctrl+Q
Compile Current File
Start Analysis & Elaboration
Start Analysis & Synthesis Ctrl+K
Start Fitter
Start Assembler
Start Timing Analyzer Ctrl+Shift+L
Start EDA Netlist Writer
Start Design Assistant
Start Incremental Fitting
Start SignalProbe Compilation Ctrl+Shift+S
Start I/O Assignment Analysis
Start Minimum Timing Analysis
Start Check & Save All Netlist Changes
Start VQM Writer
Start Test Bench Template Writer
Start EDA Synthesis
Start EDA Resynthesis

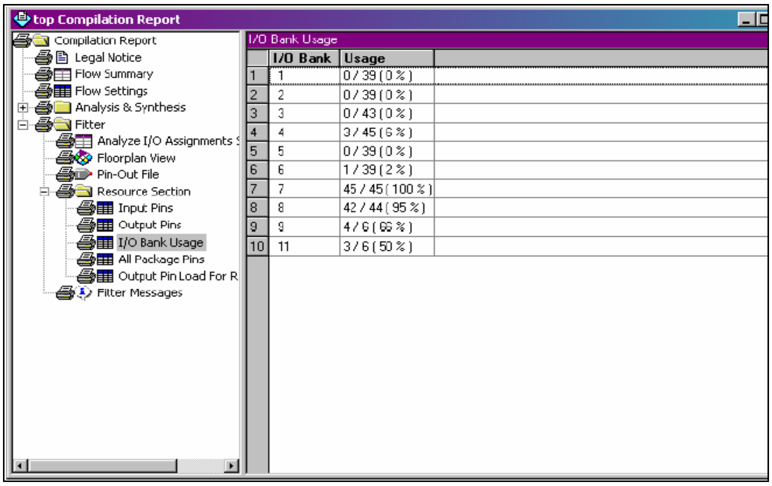
### Understanding the I/O Assignment Analysis Report

The **Start I/O Assignment Analysis** command generates a detailed analysis report (see Figure 5–8) and a Pin-out File (**.pin**). You can view the report file by choosing **Compilation Report** (Project menu). The Fitter page of the Compilation report contains the following five sections:

- Analyze I/O Assignment Summary
- Floorplan View
- Pin-Out File
- Resource Section
- Fitter Messages

The Resource Section categorizes the pins as `Input Pins`, `Output Pins`, and `Bidir Pins`. You can use the **I/O Bank Usage** page under the **Resource Section** to view the utilization of each I/O bank in your device.

*Figure 5–8. Summary of the I/O Bank Usage in the I/O Assignment Analysis Report*



| | I/O Bank | Usage |
|---|---|---|
| 1 | 1 | 0 / 39 ( 0 % ) |
| 2 | 2 | 0 / 39 ( 0 % ) |
| 3 | 3 | 0 / 43 ( 0 % ) |
| 4 | 4 | 3 / 45 ( 6 % ) |
| 5 | 5 | 0 / 39 ( 0 % ) |
| 6 | 6 | 1 / 39 ( 2 % ) |
| 7 | 7 | 45 / 45 ( 100 % ) |
| 8 | 8 | 42 / 44 ( 95 % ) |
| 9 | 9 | 4 / 6 ( 66 % ) |
| 10 | 11 | 3 / 6 ( 50 % ) |

The **Fitter Messages** page stores all messages including errors, warnings, and information messages. See "Detailed Error/Status Messages" on page 5–11 for more information.

## Suggested & Partial Placement

The **Start I/O Assignment Analysis** command automatically assigns locations to pins that do not have pin location assignments. For example, if you assign an Edge location to a group of LVDS pins, the I/O assignment analysis assigns pin locations for each LVDS pin in the specified edge location and then performs legality checks.

Choose **Back-Annotate Assignments** (Assignments menu), select **Pin & device** assignments, and click **OK** to accept the suggested pin locations. Back-annotation saves your pin and device assignments in the QSF.

### Detailed Error/Status Messages

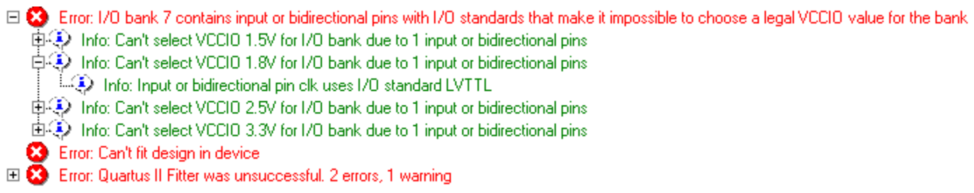
The **Start I/O Assignment Analysis** command provides detailed messages to help you quickly understand and resolve pin assignment errors. Each detailed message includes a related node name and a description of the problem.

You can view the detailed messages in the **Fitter Messages** page in the compilation report, and in the **Processing** tab in the **Messages** window. Choose **Utility Windows > Messages** (View menu) to open the **Messages** window.

Use the location box to help resolve the error messages. Select from the location list and click **Locate**.

Following is an example of error messages reported by I/O assignment analysis:

*Figure 5–9. Error Message Report by I/O Assignment Analysis*



```
Error: I/O bank 7 contains input or bidirectional pins with I/O standards that make it impossible to choose a legal VCCIO value for the bank
    Info: Can't select VCCIO 1.5V for I/O bank due to 1 input or bidirectional pins
    Info: Can't select VCCIO 1.8V for I/O bank due to 1 input or bidirectional pins
        Info: Input or bidirectional pin clk uses I/O standard LVTTL
    Info: Can't select VCCIO 2.5V for I/O bank due to 1 input or bidirectional pins
    Info: Can't select VCCIO 3.3V for I/O bank due to 1 input or bidirectional pins
Error: Can't fit design in device
Error: Quartus II Fitter was unsuccessful. 2 errors, 1 warning
```

## Scripting Support

You can run procedures and make settings described in this chapter with a Tcl script. You can also run some procedures at a command prompt. For more information about Tcl scripting, see the *Tcl Scripting* chapter in Volume 2 of this handbook. For more information about command-line scripting, see the *Command-Line Scripting* chapter in Volume 2 of this handbook. For detailed information about scripting command options type `quartus_sh --qhelp` ↵ at a system command prompt.

### Reserving Pins

Use the following Tcl command to reserve a pin. For more information about reserving pins, see page 5–6.

```
set_instance_assignment -name RESERVE_PIN <value> -to
<signal name>↵
```

Valid values are `“AS BIDIRECTIONAL”`, `”AS INPUT TRI-STATED”`, `“AS OUTPUT DRIVING AN UNSPECIFIED SIGNAL”`, `“AS OUTPUT DRIVING GROUND”` and `“AS SIGNAL PROBE OUTPUT”`. Include the quotes when specifying the value.

## Location Assignments

Use the following Tcl command to assign a signal to a pin or device location. For more information about location assignments, see page page 5–7.

```
set_location_assignment <location> -to <signal name>↵
```

Valid locations are pin location names, such as Pin_A3. The Stratix series products and Cyclone device families also support edge and I/O bank locations. Edge locations are EDGE_BOTTOM, EDGE_LEFT, EDGE_TOP, and EDGE_RIGHT. I/O bank locations include IOBANK_1 up to IOBANK_*n*, where *n* is the number of I/O banks in a particular device.

## Generating a Mapped Netlist

You can generate a mapped netlist with a Tcl command or with a command run at a command prompt. For more information about generating a mapped netlist, see page 5–8.

### *Tcl Command*

Enter the following in a Tcl console or script:

```
execute_module -tool map
```

The **execute_module** command is in the flow package.

### *Command Prompt*

Type the following at a (non-Tcl) system command prompt:

```
quartus_map <project name>↵
```

### Running the I/O Assignment Analysis

You can run the I/O Assignment Analysis with a Tcl command or with a command run at a command prompt. For more information about running the I/O assignment analysis, see page 5–9.

Enter the following in a Tcl console or script:

```
execute_flow -check_ios
```

## Conclusion

The **Start I/O Assignment Analysis** command quickly and thoroughly validates the legality of your pin-related assignments. This helps reduce development time by catching illegal pin assignments early in the design cycle without wasting long design compilations.

By providing the designer with more confidence in the device pin-outs at an early stage, board layout engineers can work in parallel with FPGA designers to achieve a time-to-market advantage.

# Section III. Area Optimization & Timing Closure

Techniques for achieving the highest design performance are important when designing for programmable logic devices (PLDs), especially higher density FPGAs. The Altera® Quartus® II software offers many advanced design analysis tools that allow for detailed timing analysis of your design, including a fully integrated Timing Closure Floorplan Editor. With these tools and options, critical paths can be easily determined and located in the targeted device floorplan. This section explains how to use these tools and options to enhance your FPGA design analysis flow.

This section includes the following chapters:

## Revision History

The table below shows the revision history for Chapters 6 to 12.

| Chapter(s) | Date / Version | Changes Made |
|---|---|---|
| 6 | June 2004 v2.0 | ● Updates to tables, figures.<br>● New functionality in the Quartus II software version 4.1. |
| | Feb. 2004 v1.0 | Initial release |
| 7 | June 2004 v2.0 | ● Updates to tables, figures.<br>● New functionality in the Quartus II software version 4.1. |
| | Feb. 2004 v1.0 | Initial release |
| 8 | June 2004 v2.0 | ● Updates to tables, figures.<br>● New functionality in the Quartus II software version 4.1. |
| | Feb. 2004 v1.0 | Initial release |
| 9 | June 2004 v2.0 | ● Updates to tables, figures.<br>● New functionality in the Quartus II software version 4.1. |
| | Feb. 2004 v1.0 | Initial release |
| 10 | August 2004 v2.1 | ● New functionality in the Quartus II software version 4.1 Sp1 |
| | June 2004 v2.0 | ● Updates to tables, figures.<br>● New functionality in the Quartus II software version 4.1. |
| | Feb. 2004 v1.0 | Initial release |
| 11 | June 2004 v2.0 | ● Updates to tables, figures.<br>● New functionality in the Quartus II software version 4.1. |
| | Feb. 2004 v1.0 | Initial release |
| 12 | Feb. 2004 v1.0 | Initial release. |

# 6. Design Optimization for Altera Devices

qii52005-2.0

## Introduction

Techniques for achieving the highest design performance are important when designing for programmable logic devices (PLDs). The tools that facilitate these techniques must provide the highest level of flexibility without compromising ease-of-use. The optimization features available in the Quartus® II software allow you to meet performance requirements by facilitating optimization at multiple points in the design process. You can apply optimizations to an overall design or to sub-modules of a design that are integrated later.

For more information on a block-based design approach, see the *Hierarchical Block-Based & Team-Based Design Flows* chapter in Volume 1 of the *Quartus II Handbook*.

This chapter explains techniques to reduce resource usage, improve timing performance, and reduce compile times when designing with Altera® devices. It also explains how and when to use some of the Quartus II software features described in detail in other chapters of the *Quartus II Handbook*.

The results of following these recommendations are design-specific. Applying each technique may not always improve your results. Quartus II options and settings are set to default values that, on average, provide the best trade-off between compilation time, resource utilization, and timing performance. The software allows you to adjust these settings to concentrate on your area of interest and see if different settings provide better results for your specific design. Use the optimization flow described in this chapter to explore various compiler settings and determine the combination of techniques that provide the required results for your design.

The first stage in the optimization process is to perform an initial compilation (see "Initial Compilation" on page 6–2) to establish a baseline that you can use to analyze your design. "Design Analysis" on page 6–6 explains how to analyze the results of your design, and provides links to the sections of this chapter where you can proceed with resource or performance optimization. Altera recommends optimizing resource usage first, then I/O timing, then $f_{MAX}$ timing, so this chapter presents the recommendations for each stage in the appropriate order. This chapter first documents this analysis and optimization process for look-up table (LUT)-based devices, including FPGA devices and MAX® II device family

CPLDs. It then focuses on the process for MAX 7000 and MAX 3000 device family macrocell-based CPLDs. The final optimization section covers compilation time optimization, which is device independent.

## Initial Compilation

Ensure that you check all the following suggested compilation assignments before compiling the design in the Quartus II software. Significantly different compilation results can occur depending on assignments made. This section describes the basic assignments and settings to make for your initial compilation.

### Device Setting

Assigning a specific device determines the timing model that the Quartus II software uses during compilation. It is important to choose the correct speed grade to obtain accurate results and the best optimization. The device size and the package determines the device pin-out and how many resources the Quartus II software can use.

Choose the target device on the **Device** page of the **Settings** dialog box (Assignments menu).

### Timing Requirements Settings

An important step in obtaining the highest performance, especially for high performance FPGA designs, is the application of detailed timing requirements. The Quartus II PowerFit™ Fitter attempts to meet or exceed specified timing requirements (depending on the selected options as described in "Fitter Effort Setting" on page 6–4). The Quartus II physical synthesis optimizations are also performed based on the constraints in specified timing requirements (see "Synthesis Netlist Optimizations and Physical Synthesis Optimizations" on page 6–28 for more information). In addition, timing requirements are used during timing analysis. The compilation report shows whether timing requirements were met and provides detailed timing information on which paths violate the timing requirements.

Make timing requirement settings in the **Timing Requirement & Options** page of the **Settings** dialog box (Assignments menu) or with the Assignment Editor. On the **Timing Requirement & Options** page use the **Delay requirements**, **Minimum delay requirements**, and **Clock Settings** boxes to enter global requirements, or select **Settings for individual clock signals** to make settings on individual clocks (recommended for multiple-clock designs). First create the clock setting, then apply it to the clock node in the design. Running the Timing Wizard makes it easy to make individual clock settings.

Every clock signal should have an accurate clock setting assignment. All I/O pins for which $t_{SU}$ $t_H$, or $t_{CO}$ is to be optimized should also have settings. In addition, if you have any $t_{PD}$ or minimum $t_{CO}$ constraints, those should be specified as well. Therefore, if there is more than one clock or there are different I/O requirements for different pins, use the Timing Wizard to make multiple clock settings and the Assignment Editor to make individual I/O assignments rather than using the global settings.

It is important to make any complex timing assignments according to the needs of the design, including multicycle and cut-timing path assignments. This information allows the Quartus II software to make appropriate trade-offs between paths. Make these settings with the Assignment Editor.

When there are any timing constraints in the design, the Quartus II software does not attempt to optimize clocks that are unconstrained. Specify timing constraints on all clock signals in the design wherever possible for best results.

For more information on how to make timing assignments, refer to the *Quartus II Timing Analysis* chapter in Volume 3 of the *Quartus II Handbook.* Also see Quartus II Help.

## Smart Compilation Setting

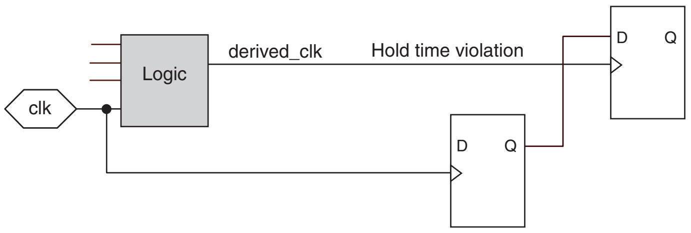
Smart compilation can reduce compile time, especially when you have multiple compilation iterations during the optimization phase of the design process; however, it will use more disk space. Turn on the **Use Smart compilation** option on the **Compilation Process** page of the **Settings** dialog box (Assignments menu).

## Timing Driven Compilation Settings

Ensure that the **Optimize timing** and the **Optimize I/O cell register placement for timing** options on the **Fitter Settings** page of the **Settings** dialog box (Assignments menu) are set appropriately. Turning on these options allows the Quartus II software to optimize your design based on the timing requirements that you have specified with various timing assignments.

The **Optimize hold timing** option is another timing-driven compilation option that directs the Quartus II software to optimize minimum delay timing constraints. This option is available only for Stratix® II, Stratix, Stratix GX, Cyclone™ II, Cyclone, and MAX II devices. When this option is turned on, the Quartus II software adds delay to connections as needed to guarantee that the minimum delays required by these constraints are satisfied. If you choose **I/O Paths and Minimum TPD Paths** (the default choice), the Fitter works to meet hold times ($t_H$) from device input pins to registers, minimum delays from I/O pins or registers to I/O pins or registers ($t_{PD}$), and minimum clock-to-out time ($t_{CO}$) from registers to output pins. If you select **All paths**, the Fitter also works to meet hold requirements from registers to registers, as in Figure 6–1, where a derived clock generated with logic causes a hold time problem on another register. However, if your design has internal hold time violations between registers, this is not the recommended way to fix internal hold violation problems. Altera recommends instead that you fix internal register to register hold problems by making changes to your design, such as using a clock enable instead of a derived or gated clock.

***Figure 6–1. Optimize Hold Timing Option Fixing an Internal Hold Time Violation***



For good design practices that can help eliminate internal hold time violations, see the *Design Recommendations for Altera Devices* chapter in Volume 1 of the *Quartus II Handbook*.

## Fitter Effort Setting

You can modify the **Fitter Effort** setting on the **Fitter Settings** page of the **Settings** dialog box. The default setting in the Quartus II software depends on the device family specified.

The **Standard Fit** option attempts to exceed specified timing requirements and achieve the best possible timing results for your design. This Fitter effort setting usually involves the longest compilation time.

The **Fast Fit** option reduces the amount of optimization effort for each algorithm employed during fitting. This reduces the compilation time by about 50%, while resulting in a fit that has, on average, 10% lower $f_{MAX}$ than that achieved using the **Standard Fit** setting. For a small minority of hard-to-fit circuits, the reduced optimization resulting from using the **Fast Fit** option can result in the first fitting attempt being unroutable, resulting in multiple fitting attempts and a long fitting time.

The **Auto Fit** option (available for Stratix II, Stratix, Stratix GX, Cyclone II, Cyclone, and MAX II devices only) decreases compilation time by directing the Fitter to reduce Fitter effort after meeting the design's timing requirements if it meets internal routability requirements. The internal routability requirements reduce the possibility of routing congestion and help ensure quick, successful routing. If you want the Fitter to try to exceed the timing requirements by a certain margin before reducing Fitter effort, you can specify a minimum slack that the Fitter must try to achieve before reducing Fitter effort in the **Desired worst case slack** box. This option also causes the Quartus II Fitter to optimize for shorter compile times instead of maximum performance when there are no timing constraints. For designs with no timing requirements, the resulting $f_{MAX}$ is an average of 15% lower than using the **Standard Fit** option. If your design has aggressive timing requirements or is hard to route, the placement does not stop early and the compile time is the same as using the **Standard Fit** option. For designs with easy or no timing requirements, the **Auto Fit** option reduces compile time by 40% on average.

☞ Note that selecting this option does not guarantee that the Fitter will meet the design's timing requirements, and specifying a minimum slack does not guarantee that the Fitter will achieve the slack.

## I/O Assignments

The I/O standards and drive strengths specified for a design affect I/O timing. Specify these assignments so that the Quartus II software uses accurate I/O timing delays in timing analysis and Fitter optimizations.

The Quartus II software can choose pin locations automatically for best quality of results. If your pin locations are not fixed due to printed circuit board (PCB) layout requirements, Altera recommends leaving pin locations unconstrained to achieve the best results. If your pin locations are already fixed, make the pin assignments in the Quartus II software to constrain the compilation appropriately. "Optimization Techniques for Macrocell-Based (MAX 7000 and MAX 3000) CPLDs" on page 6–41 includes recommendations for making pin assignments, since your pin assignments can have a larger effect on your quality of results in smaller macrocell-based architectures.

You can assign I/O standards and pin locations with the **Assignment Editor** (Assignments menu) or Tcl script commands.

For more information on I/O standards and pin constraints, see the appropriate device data sheet or handbook. For information on using the Assignment Editor, refer to the *Assignment Editor* chapter in Volume 2 of the *Quartus II Handbook*. For information on scripting, see the *Tcl Scripting* chapter in Volume 2 of the *Quartus II Handbook*.

## Design Analysis

The initial compilation establishes whether the design achieves a successful fit and meets the specified performance. The Compilation Report reports the design results. This section describes how to analyze your design results, which is the first stage in the design optimization process.

After design analysis, proceed to the other optimization stages, as follows.

For LUT-based devices (FPGAs and MAX II CPLDs) see "Optimization Techniques for LUT-Based (FPGA and MAX II) Devices" on page 6–12:

- If your design does not fit, see "Resource Utilization Optimization Techniques (LUT-Based Devices)" on page 6–13 before trying to optimize I/O timing or $f_{MAX}$ timing.
- If the I/O timing performance requirements are not met, see "I/O Timing Optimization Techniques (LUT-Based Devices)" on page 6–21 before trying to optimize $f_{MAX}$ timing.
- If $f_{MAX}$ performance requirements are not met, see "$f_{MAX}$ Timing Optimization Techniques (LUT-Based Devices)" on page 6–27.

For Macrocell-based devices (MAX 7000 and MAX 3000 CPLDs) see "Optimization Techniques for Macrocell-Based (MAX 7000 and MAX 3000) CPLDs" on page 6–41:

- If your design does not fit, see "Resource Utilization Optimization Techniques (Macrocell-based CPLDs)" on page 6–41 before trying to optimize I/O timing or $f_{MAX}$ timing.
- If the timing performance requirements are not met, see "Timing Optimization Techniques (Macrocell-based CPLDs)" on page 6–49.

For techniques to reduce the compilation time, see "Compilation Time Optimization Techniques" on page 6–55.

### Resource Utilization

Determining device utilization is important regardless of whether a successful fit is achieved. If your compilation results in a no-fit error, then resource utilization information is important to analyze the fitting problems in your design. If your fitting is successful, review the resource utilization information to determine whether the future addition of extra logic or any other design changes could introduce fitting difficulties.

To determine resource usage, see the **Flow Summary** section of the Compilation Report. This section reports how many pins are used, as well as other device resources such as memory bits, digital signal processing (DSP) block 9-bit elements, and phase-locked loops (PLLs). The **Flow Summary** indicates whether the design exceeds the available device resources. More detailed information is available by viewing the reports under **Resource Section** in the **Fitter** section of the **Compilation Report** (Processing menu).

Note that for Stratix II devices, a device with low utilization does not have the lowest adaptive logic module (ALM) utilization possible. For Stratix II devices, the Fitter uses adaptive look-up tables (ALUTs) in different ALMs even when the logic could be placed within one ALM. The Quartus II Fitter spreads out a design as much as possible while trying to meet any timing constraints set by the user. As the device fills up, the Fitter automatically searches for logic functions with common inputs to place in one ALM. The number of partnered ALUTs and packed registers also increases.

If resource usage is reported as less than 100% and a successful fit was not achieved, then it is likely that there were not enough routing resources or that some assignments were illegal. In either case, a message appears in the **Processing** tab of the **Messages** window to explain the problem.

If the Fitter finishes very quickly, then a resource may be over-utilized or there may be an illegal assignment (an error message is also reported for illegal assignments). If the Quartus II software runs for a long time, then it is likely that a legal placement or route cannot be found. Look for compilation messages that give an indication of the problem.

You can use the Timing Closure Floorplan to view areas of routing congestion.

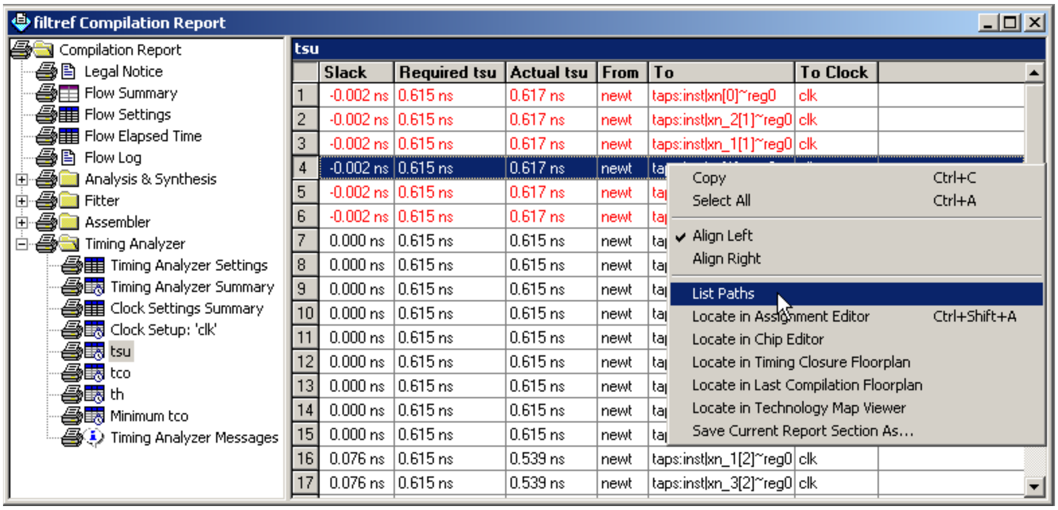
For details on using the Timing Closure Floorplan, see the *Timing Closure Floorplan* chapter in Volume 2 of the *Quartus II Handbook*.

## I/O Timing (including $t_{PD}$)

To determine whether I/O timing has been met, see the **Timing Analyzer** section of the **Compilation Report** (Processing menu). The $t_{SU}$, $t_H$, and $t_{CO}$ reports list the I/O paths, along with the "Required" timing number if you have made a timing requirement, its "Actual" timing number for the parameter as reported by the Quartus II software, and the slack, or difference between your requirement and the actual number as specified by the Quartus II software. If you have any point-to-point propagation delay assignments ($t_{PD}$), the $t_{PD}$ report lists the corresponding paths.

The I/O paths that have not met the required timing performance are reported as having negative slack and are displayed in red, as shown in Figure 6–2. Even if you have not made an I/O timing assignment on that pin, the "Actual" number is the timing number that you must meet when the device runs in your system.

*Figure 6–2. I/O Timing Report*



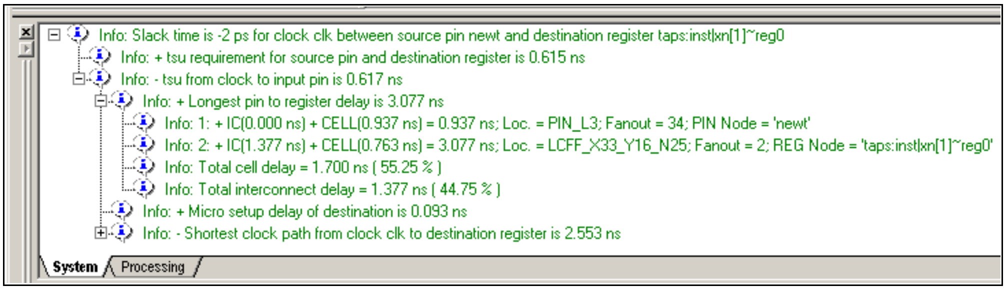
tsu

| | Slack | Required tsu | Actual tsu | From | To | To Clock |
|---|---|---|---|---|---|---|
| 1 | -0.002 ns | 0.615 ns | 0.617 ns | newt | taps:inst\|xn[0]~reg0 | clk |
| 2 | -0.002 ns | 0.615 ns | 0.617 ns | newt | taps:inst\|xn_2[1]~reg0 | clk |
| 3 | -0.002 ns | 0.615 ns | 0.617 ns | newt | taps:inst\|xn_1[1]~reg0 | clk |
| 4 | -0.002 ns | 0.615 ns | 0.617 ns | newt | ta | |
| 5 | -0.002 ns | 0.615 ns | 0.617 ns | newt | ta | |
| 6 | -0.002 ns | 0.615 ns | 0.617 ns | newt | ta | |
| 7 | 0.000 ns | 0.615 ns | 0.615 ns | newt | ta | |
| 8 | 0.000 ns | 0.615 ns | 0.615 ns | newt | ta | |
| 9 | 0.000 ns | 0.615 ns | 0.615 ns | newt | ta | |
| 10 | 0.000 ns | 0.615 ns | 0.615 ns | newt | ta | |
| 11 | 0.000 ns | 0.615 ns | 0.615 ns | newt | ta | |
| 12 | 0.000 ns | 0.615 ns | 0.615 ns | newt | ta | |
| 13 | 0.000 ns | 0.615 ns | 0.615 ns | newt | ta | |
| 14 | 0.000 ns | 0.615 ns | 0.615 ns | newt | ta | |
| 15 | 0.000 ns | 0.615 ns | 0.615 ns | newt | ta | |
| 16 | 0.076 ns | 0.615 ns | 0.539 ns | newt | taps:inst\|xn_1[2]~reg0 | clk |
| 17 | 0.076 ns | 0.615 ns | 0.539 ns | newt | taps:inst\|xn_3[2]~reg0 | clk |

To analyze the reasons that your timing requirements were not met, right-click a particular entry in the report and choose **List Paths** (as shown in Figure 6–2). A message listing the paths appears in the **System** tab of the **Messages** window. You can expand a selection by clicking the "**+**" icon at the beginning of the line, as shown in Figure 6–3. This is a good method of determining where along the path the greatest delay is located.

The List Paths report lists the slack time and how that slack time was calculated. By expanding the different entries, you can see the incremental delay through each node in the path as well as the total delay. The incremental delay is the sum of the interconnect delay (IC) and the cell delay (CELL) through the logic.

*Figure 6–3. I/O Slack Report*



```
Info: Slack time is -2 ps for clock clk between source pin newt and destination register taps:inst|xn[1]~reg0
  Info: + tsu requirement for source pin and destination register is 0.615 ns
  Info: - tsu from clock to input pin is 0.617 ns
    Info: + Longest pin to register delay is 3.077 ns
      Info: 1: + IC(0.000 ns) + CELL(0.937 ns) = 0.937 ns; Loc. = PIN_L3; Fanout = 34; PIN Node = 'newt'
      Info: 2: + IC(1.377 ns) + CELL(0.763 ns) = 3.077 ns; Loc. = LCFF_X33_Y16_N25; Fanout = 2; REG Node = 'taps:inst|xn[1]~reg0'
      Info: Total cell delay = 1.700 ns ( 55.25 % )
      Info: Total interconnect delay = 1.377 ns ( 44.75 % )
    Info: + Micro setup delay of destination is 0.093 ns
    Info: - Shortest clock path from clock clk to destination register is 2.553 ns
```

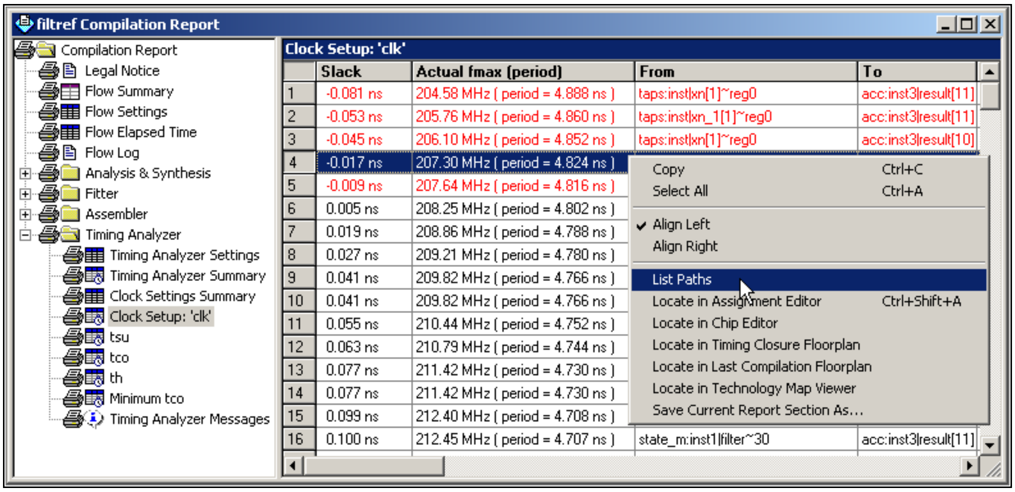
To visually analyze I/O timing, right-click on an I/O entry in the report and select **Locate in Timing Closure Floorplan** (right button pop-up menu) to highlight the I/O path on the floorplan. Negative slack indicates paths that failed to meet their timing requirements. There are also options to allow you to see all the intermediate nodes (i.e., combinational logic cells) on a path and the delay for each level of logic. You can also look at the fan-in and fan-out of a selected node.

For more information on how timing numbers are calculated, refer to the *Quartus II Timing Analysis* chapter in Volume 3 of the *Quartus II Handbook*. For details on using the Timing Closure Floorplan, see the *Timing Closure Floorplan* chapter in Volume 2 of the *Quartus II Handbook*.

## $f_{MAX}$ Timing

To determine whether $f_{MAX}$ timing requirements are met, see the **Timing Analyzer** section of the **Compilation Report** (Processing menu). The **Clock Setup** folder gives figures for the actual register-to-register $f_{MAX}$, as reported by the Quartus II software, and the slack, or difference between the timing requirement you have specified and the actual number specified by the Quartus II software. The paths that do not meet timing requirements are shown with a negative slack and appear in red (see Figure 6–4).

*Figure 6–4. $f_{MAX}$ Timing Analysis Report*



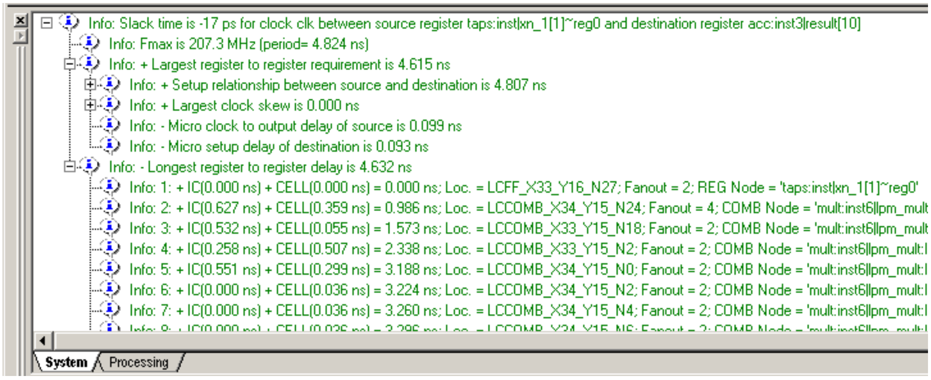
Clock Setup: 'clk'

| | Slack | Actual fmax (period) | From | To |
|---|---|---|---|---|
| 1 | -0.081 ns | 204.58 MHz ( period = 4.888 ns ) | taps:inst\|xn[1]~reg0 | acc:inst3\|result[11] |
| 2 | -0.053 ns | 205.76 MHz ( period = 4.860 ns ) | taps:inst\|xn_1[1]~reg0 | acc:inst3\|result[11] |
| 3 | -0.045 ns | 206.10 MHz ( period = 4.852 ns ) | taps:inst\|xn[1]~reg0 | acc:inst3\|result[10] |
| 4 | -0.017 ns | 207.30 MHz ( period = 4.824 ns ) | | |
| 5 | -0.009 ns | 207.64 MHz ( period = 4.816 ns ) | | |
| 6 | 0.005 ns | 208.25 MHz ( period = 4.802 ns ) | | |
| 7 | 0.019 ns | 208.86 MHz ( period = 4.788 ns ) | | |
| 8 | 0.027 ns | 209.21 MHz ( period = 4.780 ns ) | | |
| 9 | 0.041 ns | 209.82 MHz ( period = 4.766 ns ) | | |
| 10 | 0.041 ns | 209.82 MHz ( period = 4.766 ns ) | | |
| 11 | 0.055 ns | 210.44 MHz ( period = 4.752 ns ) | | |
| 12 | 0.063 ns | 210.79 MHz ( period = 4.744 ns ) | | |
| 13 | 0.077 ns | 211.42 MHz ( period = 4.730 ns ) | | |
| 14 | 0.077 ns | 211.42 MHz ( period = 4.730 ns ) | | |
| 15 | 0.099 ns | 212.40 MHz ( period = 4.708 ns ) | | |
| 16 | 0.100 ns | 212.45 MHz ( period = 4.707 ns ) | state_m:inst1\|filter~30 | acc:inst3\|result[11] |

To analyze why your timing requirements were not met, right click on a particular entry in the report and choose **List Paths** (as shown in Figure 6–4). A message listing the paths appears in the **System** tab of the Messages window. You can expand a selection by clicking the "+" icon at the beginning of the line, as shown in Figure 6–5. This is a good method of determining where along the path the greatest delay is located.

The List Paths report lists the slack time and how that slack time was calculated. By expanding the different entries, you can see the incremental delay through each node in the path as well as the total delay. The incremental delay is the sum of the interconnect delay (IC) and the cell delay (CELL) through the logic.

*Figure 6–5. $f_{MAX}$ Slack Report*



```
Info: Slack time is -17 ps for clock clk between source register taps:inst|xn_1[1]~reg0 and destination register acc:inst3|result[10]
    Info: Fmax is 207.3 MHz (period= 4.824 ns)
    Info: + Largest register to register requirement is 4.615 ns
        Info: + Setup relationship between source and destination is 4.807 ns
        Info: + Largest clock skew is 0.000 ns
        Info: - Micro clock to output delay of source is 0.099 ns
        Info: - Micro setup delay of destination is 0.093 ns
    Info: - Longest register to register delay is 4.632 ns
        Info: 1: + IC(0.000 ns) + CELL(0.000 ns) = 0.000 ns; Loc. = LCFF_X33_Y16_N27; Fanout = 2; REG Node = 'taps:inst|xn_1[1]~reg0'
        Info: 2: + IC(0.627 ns) + CELL(0.359 ns) = 0.986 ns; Loc. = LCCOMB_X34_Y15_N24; Fanout = 4; COMB Node = 'mult:inst6|lpm_mult
        Info: 3: + IC(0.532 ns) + CELL(0.055 ns) = 1.573 ns; Loc. = LCCOMB_X33_Y15_N18; Fanout = 2; COMB Node = 'mult:inst6|lpm_mult
        Info: 4: + IC(0.258 ns) + CELL(0.507 ns) = 2.338 ns; Loc. = LCCOMB_X33_Y15_N2; Fanout = 2; COMB Node = 'mult:inst6|lpm_mult:l
        Info: 5: + IC(0.551 ns) + CELL(0.299 ns) = 3.188 ns; Loc. = LCCOMB_X34_Y15_N0; Fanout = 2; COMB Node = 'mult:inst6|lpm_mult:l
        Info: 6: + IC(0.000 ns) + CELL(0.036 ns) = 3.224 ns; Loc. = LCCOMB_X34_Y15_N2; Fanout = 2; COMB Node = 'mult:inst6|lpm_mult:l
        Info: 7: + IC(0.000 ns) + CELL(0.036 ns) = 3.260 ns; Loc. = LCCOMB_X34_Y15_N4; Fanout = 2; COMB Node = 'mult:inst6|lpm_mult:l
```

You can visually analyze $f_{MAX}$ paths by right-clicking on a path in the report and selecting **Locate in Timing Closure Floorplan** to display the **Timing Closure Floorplan**, which then highlights the path. You can also view all failing paths in the **Timing Closure Floorplan** using the **Show Critical Paths** command.

For more information on how timing analysis results are calculated, refer to the *Quartus II Timing Analysis* chapter in Volume 3 of the *Quartus II Handbook*. For details on using the Timing Closure Floorplan, see the *Timing Closure Floorplan* chapter in Volume 2 of the *Quartus II Handbook.*

## Compilation Time

In long compilations, most of the time is spent in the Analysis & Synthesis and Fitter modules. Analysis & Synthesis includes synthesis netlist optimizations, if you have turned on those options. The Fitter includes two steps, placement and routing, and includes Physical Synthesis if you have turned on those options. The **Flow Elapsed Time** section of the **Compilation Report** shows how much time the Analysis & Synthesis and Fitter modules took. The **Fitter Messages** report in the **Fitter** section of the **Compilation Report** shows how much time was spent in placement and how much time was spent in routing.

The applicable messages say `Info: Fitter placement operations ending: elapsed time = n seconds` and `Info: Fitter routing operations ending: elapsed time = n seconds`.

Placement describes the process of finding optimum locations for the logic in your design. Routing describes the process of connecting the nets between the logic in your design. There are many possible placements for the logic in a design, and finding better placements typically takes more compilation time. Good logic placement allows you to more easily meet your timing requirements and makes the design easy to route.

## Optimization Techniques for LUT-Based (FPGA and MAX II) Devices

This section of the chapter addresses resource and timing optimization issues for LUT-based Altera devices, which consists of all FPGA devices and MAX II device family CPLDs.

For information on optimizing MAX 7000 and MAX 3000 CPLD designs, refer to "Optimization Techniques for Macrocell-Based (MAX 7000 and MAX 3000) CPLDs" on page 6–41. For information on optimizing compilation time (when targeting any device), refer to "Compilation Time Optimization Techniques" on page 6–55.

### Optimization Advisors

The Quartus II software includes the **Resource Optimization Advisor** and the **Timing Optimization Advisor** (Tools menu) that provide guidance for making settings to optimize your design. The advisors cover many of the suggestions listed in this chapter. If you open the advisors after compilation, the Optimization Advisors show icons that indicate which resources or timing constraints were not met.

When you expand one of the categories (such as **Logic Element Usage** or **Maximum Frequency ($f_{MAX}$)**), recommendations are split into stages. The stages show the order in which you should apply the recommended settings. The first stage contains the options that are easiest to change, make the least drastic changes to your design optimization, and have the least effect on compilation time. Icons indicate whether each recommended setting has been made in the current project. Refer to the "How to use" page in the Advisor for a legend that describes each icon.

There is a link from each recommendation to the appropriate location in the Quartus II user interface where you can change the setting. This provides you with the most control over which settings are made, and helps you learn about the settings in the software.

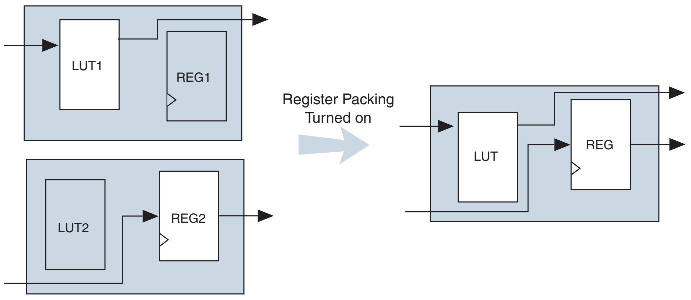
## Resource Utilization Optimization Techniques (LUT-Based Devices)

After design analysis, the next stage of design optimization is to improve resource utilization. Complete this stage before proceeding to I/O timing optimization or $f_{MAX}$ timing optimization. First, ensure that you have set the basic constraints described in "Initial Compilation" on page 6–2. If a design is not fitting into a specified device, use the techniques in this section to achieve a successful fit.

### Use Register Packing

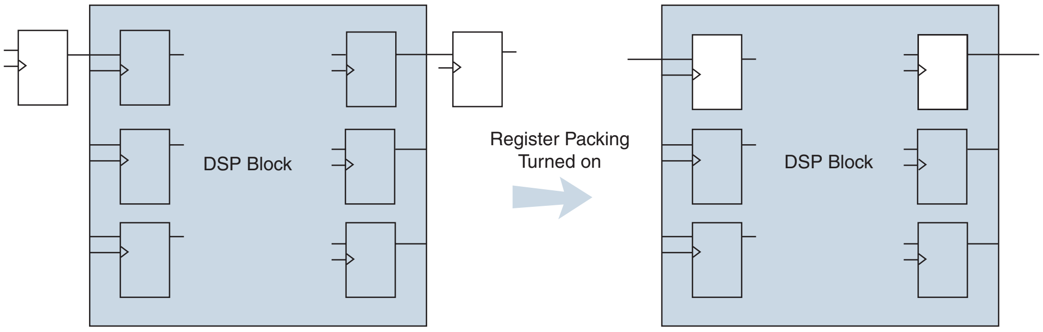
The **Auto Packed Registers** option is available regardless of the tool used to synthesize the design. Register packing combines a logic cell where only the register is used with another logic cell where only the lookup table (LUT) is used, and implements both functions in a single logic cell. Figure 6–6 shows the packing and the gain of one logic cell.

*Figure 6–6. Register Packing*



Registers may also be packed into DSP blocks as shown in Figure 6–7.

*Figure 6–7. Register Packing in DSP Blocks*



The following list indicates the most common cases in which register packing can help to optimize a design:

- A LUT can be implemented in the same cell as an unrelated register with a single data input
- A LUT can be implemented in the same cell as the register that is fed by the LUT
- A LUT can be implemented in the same cell as the register that feeds the LUT
- A register can be packed into a RAM block
- A register can be packed into a DSP block
- A register can be packed into an I/O Element (IOE)

The following options are available for register packing (for certain device families):

- **Off**—Does not pack registers.
- **Normal**—Default setting packs registers when this is not expected to hurt timing results.
- **Minimize Area**—Aggressively packs registers to reduce area.
- **Minimize Area with Chains**—Aggressively packs registers to reduce area. This option packs registers with carry chains. It also converts registers into register cascade chains and packs them with other logic to reduce area. This option is available only for Stratix II, Stratix, Stratix GX, Cyclone II, Cyclone, and MAX II devices.
- **Auto**—Attempts to achieve the best performance while maintaining a fit for the design in the specified device. The Fitter combines all combinational (LUT) and sequential (register) functions that are deemed to benefit circuit speed. In addition, more aggressive combinations of unrelated combinational and sequential functions are performed to the extent required to reduce the area of the design to achieve a fit in the specified device. This option is available only for Stratix II, Stratix, Stratix GX, Cyclone II, Cyclone, and MAX II devices.

Turning on register packing decreases the number of logic elements (LEs) or adaptive logic modules (ALMs) in the design, but could also decrease performance. To turn on register packing, turn on the **Auto Packed Registers** option by clicking **More Settings** on the **Fitter Settings** page of the **Settings** dialog box (Assignments menu).

The area reduction and performance results can vary greatly depending on the design. Typical results for register packing are shown in the following tables. Table 6–1 shows typical results for Stratix II devices, Table 6–2 shows typical results for Cyclone II devices, and Table 6–3 shows typical results for Stratix, Stratix GX, and Cyclone devices.

Note that the **Auto** setting performs more aggressive register packing as needed, so the typical results vary depending on the device logic utilization.

*Table 6–1. Typical Register Packing Results for Stratix II Devices*

| Register Packing Setting | Relative $f_{MAX}$ | Relative LE Count |
|---|---|---|
| Off | 0.95 | 1.29 |
| Normal | 1.00 | 1.00 |
| Minimize Area | 0.98 | 0.97 |
| Minimize Area with Chains | 0.98 | 0.97 |
| Auto (default) | 1.0 until device is very full, then gradually to 0.98 as required | 1.0 until device is very full, then gradually to 0.97 as required |

*Table 6–2. Typical Register Packing Results for Cyclone II Devices*

| Register Packing Setting | Relative $f_{MAX}$ | Relative LE Count |
|---|---|---|
| Off | 0.97 | 1.40 |
| Normal | 1.00 | 1.00 |
| Minimize Area | 0.96 | 0.93 |
| Minimize Area with Chains | 0.94 | 0.91 |
| Auto (default) | 1.0 until device is very full, then gradually to 0.94 as required | 1.0 until device is very full, then gradually to 0.91 as required |

*Table 6–3. Typical Register Packing Results for Stratix, Stratix GX, and Cyclone Devices*

| Register Packing Setting | Relative $f_{MAX}$ | Relative LE Count |
|---|---|---|
| Off | 1.00 | 1.12 |
| Normal | 1.00 | 1.00 |
| Minimize Area | 0.97 | 0.93 |
| Minimize Area with Chains | 0.94 | 0.90 |
| Auto (default) | 1.0 until device is very full, then gradually to 0.94 as required | 1.0 until device is very full, then gradually to 0.90 as required |

## Remove Fitter Constraints

A design with too many user constraints may not fit the targeted device. This occurs when the location or LogicLock™ assignments are too strict and there are not enough routing resources. In this case, use the **Routing Congestion** view in the **Timing Closure Floorplan** to locate routing problems in the floorplan, then remove any location and/or LogicLock region assignments in that area. If your design still does not fit, the design is over-constrained. To correct the problem, remove all location and LogicLock assignments and run successive compilations, incrementally constraining the design before each compilation.

For more information on the **Routing Congestion** view in the **Timing Closure Floorplan**, see the Quartus II Help.

## Perform WYSIWYG Resynthesis for Area

If you use another EDA synthesis tool and wish to see if the Quartus II software can re-map the circuit so that fewer LEs or ALMs are used, perform the following steps:

1. Turn on **Perform WYSIWYG primitive resynthesis (using optimization techniques specified in Analysis & Synthesis settings)** on the **Synthesis Netlist Optimizations** page under **Analysis & Synthesis Settings** in the **Settings** dialog box (Assignments menu), or apply the **Perform WYSIWYG Primitive Resynthesis** logic option to a specific module in your design with the **Assignment Editor** (Assignments menu).

2. Choose **Area** for **Optimization Technique** on the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu), or set the **Optimization Technique** logic option to **Area** for a specific module in your design with the **Assignment Editor** (Assignments menu).

3. Recompile the design.

Performing WYSIWYG resynthesis for **Area** in this way typically reduces $f_{MAX}$.

## Optimize Synthesis for Area

If your design fails to fit because it uses too much logic, resynthesize the design to improve the area utilization, as follows.

First, ensure that you have set your device and timing constraints correctly in your synthesis tool. Particularly when the area utilization of the design is a concern, ensure that you do not over-constrain the timing requirements for the design. Synthesis tools generally try to meet the specified requirements, which may result in higher device resource usage if the constraints are too aggressive.

For information on setting timing requirements and synthesis options in other synthesis tools, see the appropriate chapter in the *Synthesis* section in Volume 1 of the *Quartus II Handbook,* or your synthesis software's documentation.

### *Optimize for Area, Not Speed*

If device utilization is an important concern, some synthesis tools offer an easy way to optimize for area instead of speed. If you are using the Quartus II integrated synthesis, choose **Area** for **Optimization Technique** on the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu). You can also specify this logic option for specific modules in your design with the Assignment Editor in cases where you want to reduce area (potentially at the expense of $f_{MAX}$ timing performance) while leaving the default **Optimization Technique** setting at **Balanced** (for the best trade-off between area and speed for certain device families) or **Speed**. In some synthesis tools, not specifying an $f_{MAX}$ requirement may result in less logic utilization. Other attributes or options may also be available to help improve the quality of results, including the recommendations in the following paragraphs.

### *Change State Machine Encoding*

State machines can be encoded using various techniques. Using binary or Gray code encoding typically results in fewer state registers than one-hot encoding, which requires one register for every state bit. If your design contains state machines, changing the state machine encoding to one that uses the minimal number of registers may reduce device utilization. The effect of state machine encoding differs depending on the way your design is structured.

If your design does not manually encode the state bits, you can specify the state machine encoding in your synthesis tool. In the Quartus II integrated synthesis, choose **Minimal Bits** for **State Machine Processing** on the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu). You can also specify this logic option for specific modules or state machines in your design with the Assignment Editor.

#### *Flatten the Hierarchy*

Synthesis tools typically provide you with the option of preserving hierarchical boundaries, which may be useful for verification or other purposes. Optimizing across hierarchical boundaries, however, allows the synthesis tool to perform the most logic minimization, which may reduce area. Therefore, flatten your design hierarchy whenever possible to achieve best results. If you are using the Quartus II integrated synthesis, ensure that the **Preserve Hierarchical Boundary** logic option is turned off.

### Retarget Memory Blocks

If the design fails to fit because it runs out of device memory resources, it may be due to a lack of a certain type of memory. For example, a design may require two M-RAM blocks and be targeted for a Stratix EP1S10 device, which has only one. By building one of the memories with a different size memory block, such as an M4K memory block, it may be possible to obtain a fit.

If the memory was created with the MegaWizard® Plug-In Manager, simply open the MegaWizard and edit the RAM block type so that it targets a new memory block size.

ROM and RAM memory blocks can also be inferred from your hardware description language (HDL) code, and your synthesis software may place large shift registers into memory blocks with the `altshift_taps` megafunction. This inference can be turned off in your synthesis tool so that the memory is placed in logic instead of in memory blocks. In Quartus II integrated synthesis, disable inference by turning off the **Auto RAM Replacement**, **Auto ROM Replacement**, or **Auto Shift Register Replacement** logic option as appropriate for your whole project on the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu), or by disabling the option for a specific block in the Assignment Editor.

Depending on your synthesis tool, you may be able to set the RAM block type for inferred memory blocks as well. In Quartus II integrated synthesis, set the `ramstyle` attribute to the desired memory type for the inferred RAM blocks: M512, M4K, or M-RAM.

For more information on memory inference control, see the appropriate chapter in the *Synthesis* section in Volume 1 of the *Quartus II Handbook*, or your synthesis software's documentation.

### Retarget DSP Blocks

A design may not fit because it requires too many DSP blocks. All DSP block functions can be implemented with logic cells, making it possible to retarget some of the DSP blocks to logic to obtain a fit.

If the DSP function was created with the MegaWizard Plug-In Manager, simply open the MegaWizard and edit the block so it targets logic instead of DSP blocks.

DSP blocks can be inferred from your HDL code from multipliers, multiply-adders, and multiply-accumulators. This inference can be turned off in your synthesis tool. In Quartus II integrated synthesis, disable inference by turning off the **Auto DSP Block Replacement** logic option for your whole project on the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu), or by disabling the option for a specific block with the Assignment Editor.

For more information on disabling DSP block inference in other synthesis tools, see the appropriate chapter in the *Synthesis* section in Volume 1 of the *Quartus II Handbook,* or your synthesis software's documentation.

### Optimize Source Code

If your design does not fit because of logic utilization, and the methods described in the preceding sections do not sufficiently improve the resource utilization in the design, modify the design at the source to achieve the desired results. You may also be able to improve logic efficiency by making design-specific changes to your source code. In many cases, optimizing the design's source code can have a significant effect on your logic utilization.

If your design does not fit because of logic resources, but you have unused memory or DSP blocks, check whether you have code blocks in your design that describe memory or DSP functions but are not being inferred and placed in dedicated logic. You may be able to modify your source code to allow these functions to be placed into dedicated memory or DSP resources in the target device.

For coding style guidelines including examples of HDL code for inferring memory and DSP functions and other coding examples, refer to the *Recommended HDL Coding Styles* chapter in Volume 1 of the *Quartus II Handbook*.

### Modify Pin Assignments or Choose a Larger Package

If a design with pin assignments fails to fit, try compiling the design without the pin assignments to see whether a fit is possible for the design in the specified device and package. You can also try this approach if a Quartus II error message indicates fit problems due to pin assignments.

If the design fits when all pin assignments are ignored or when several pin assignments are ignored or moved, it may be necessary to modify the pin assignments for the design or choose a larger package.

If the design fails to fit because of lack of available I/Os, a successful fit can often be obtained by using a larger device package with more available user I/O pins.

### Use a Larger Device

If a successful fit cannot be achieved because of a shortage of LEs or ALMs, memory, or DSP blocks, you may need to use a larger device.

### Resolving Resource Utilization Issues Summary

Table 6–4 shows design options used to reduce excess resource utilization and the recommended order in which to try the options, starting with those requiring the least effort and having the greatest effect.

The Quartus II software includes the Design Space Explorer (DSE) Tcl/Tk script for automating successive compilations of a design, each employing different design options.

For more information on the DSE script, see the *Design Space Explorer* chapter in Volume 2 of the *Quartus II Handbook*.

*Table 6–4. Techniques for Resolving Resource Utilization Issues (Part 1 of 2)*

| Issue | Design Options to Employ (in Order from Left to Right) | | | | | |
|---|---|---|---|---|---|---|
| Too many logic cells used or logic cells do not fit | Use register packing | Remove Fitter constraints | Perform WYSIWYG Primitive Resynthesis | Optimize synthesis for area / change state machine encoding | Optimize source code | Use a larger device |
| Too many memory blocks used | Retarget memory blocks | Modify synthesis options | Remove Fitter constraints | Optimize source code | Use a larger device | |

*Table 6–4. Techniques for Resolving Resource Utilization Issues (Part 2 of 2)*

| Issue | Design Options to Employ (in Order from Left to Right) | | | | | |
|---|---|---|---|---|---|---|
| Too many DSP blocks used | Retarget DSP blocks | Modify synthesis options | Remove Fitter constraints | Optimize source code | Use a larger device | |
| Problems placing I/O pins | Change pin assignments | Use a larger package with the same device density | Use a larger device with a larger pin count | | | |
| Too many routing resources used | Remove Fitter constraints | Modify synthesis options | Optimize source code | Use a larger device | | |

Once resource utilization has been optimized and your design fits in the desired target device, you can proceed to optimize I/O timing, as described in the "I/O Timing Optimization Techniques (LUT-Based Devices)" section.

## I/O Timing Optimization Techniques (LUT-Based Devices)

The next stage of design optimization focuses on I/O timing. Ensure that you have made the appropriate assignments as described in "Initial Compilation" on page 6–2, and that the resource utilization is satisfactory, before proceeding with I/O timing optimization. Because changes to the I/O path affect the internal $f_{MAX}$, complete this stage before proceeding to the $f_{MAX}$ timing optimization stage.

The options presented in this section address how to improve I/O timing, including the setup delay ($t_{SU}$), hold time ($t_H$), and clock-to-output ($t_{CO}$) parameters.

### Timing-Driven Compilation

Perform I/O timing optimization using the **Optimize I/O cell register placement for timing** assignment located on the **Fitter Settings** page of the **Settings** dialog box (Assignments menu). This option moves registers into I/O elements if required to meet $t_{SU}$ or $t_{CO}$ assignments, duplicating the register if necessary (as in the case where a register fans out to multiple output locations). This option is on by default and is a global setting. The option does not apply to MAX II devices because they do not contain I/O registers.

For APEX™ 20KE and APEX 20KC devices, if the I/O register is not available, the Fitter tries to move the register into the logic array block (LAB) adjacent to the I/O element.

The **Optimize I/O cell register placement for timing** option only affects pins that have a $t_{SU}$ or $t_{CO}$ requirement. Using the I/O register is only possible if the register directly feeds a pin or is fed directly by a pin. This setting does not affect registers with the following characteristics:

- Have combinational logic between the register and the pin
- Are part of a carry or cascade chain
- Have an overriding location assignment
- Use the synchronous load or asynchronous load port, and the value is not **1** (Stratix, Stratix GX, and Cyclone devices only)
- Use the synchronous load or asynchronous clear port (APEX and APEX II devices only)

Registers with the above characteristics are optimized using the regular Quartus II Fitter optimizations.

## Fast Input, Output, & Output Enable Registers

You can manually place individual registers in I/O cells by making fast I/O assignments with the Assignment Editor. For an input register, use the **Fast Input Register** option; for an output register, use the **Fast Output Register** option; and for an output enable register, use the **Fast Output Enable Register** option. In MAX II devices, which have no I/O registers, these assignments lock the register into the LAB adjacent to the I/O pin if there is a pin location assignment on that I/O pin.

If the fast I/O setting is on, the register is always placed in the I/O element. If the fast I/O setting is off, the register is never placed in the I/O element. This is true even if the **Optimize I/O cell register placement for timing** option, located on the **Fitter Settings** page of the **Settings** dialog box (Assignments menu), is turned on. If there is no fast I/O assignment, the Quartus II software determines whether to place registers in I/O elements if the **Optimize I/O cell register placement for timing** option is turned on.

The three fast I/O options (**Fast Input Register**, **Fast Output Register**, and **Fast Output Enable Register**) can also be used to override the location of a register that is in a LogicLock region and force it into an I/O cell. If this assignment is applied to a register that feeds multiple pins, the register is duplicated and placed in all relevant I/O elements. In MAX II devices, the register is duplicated and placed in each distinct LAB location that is next to an I/O pin with a pin location assignment.

## Programmable Delays

Various programmable delay options can be used to minimize the $t_{SU}$ and $t_{CO}$ times. For Stratix II, Stratix, Stratix GX, Cyclone II, Cyclone, and MAX II devices, the Quartus II software automatically adjusts the applicable programmable delays to help meet timing requirements. For the APEX families of devices, the default values are set to generally avoid any hold time problems. Programmable delays are advanced options that should be used only after you have compiled a project, checked the I/O timing, and determined that the timing is unsatisfactory. For detailed information on the effect of these options, see the device family handbook or data sheet.

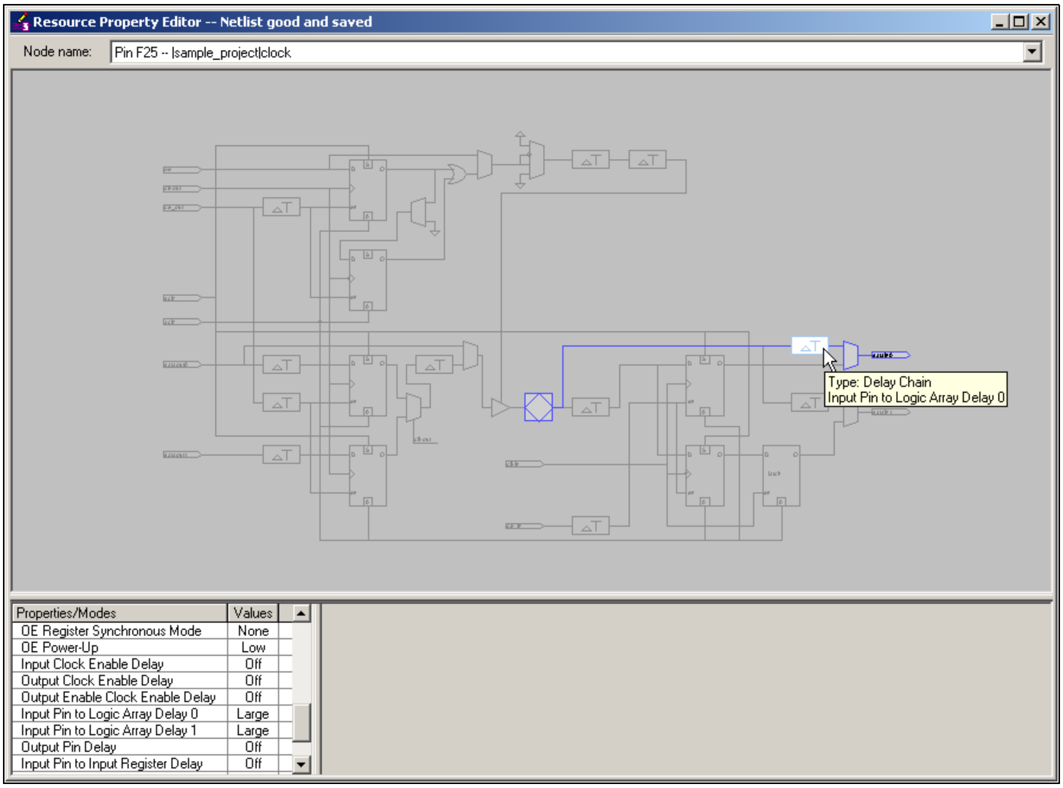
Assign programmable delay options to supported nodes with the Assignment Editor.

After you have made a programmable delay assignment and compiled the design, you can view the value of every delay chain for every I/O pin in the **Delay Chain Summary** section of the Quartus II Compilation Report.

You can also view and modify the delay chain setting for the target device with the Quartus II Chip Editor and Resource Property Editor. Figure 6–8 shows the Resource Property Editor window displaying a programmable delay implemented in the delay chain of a Stratix device. When you use the Resource Property Editor to make changes after performing a full compilation, you don't need to recompile the entire design; you can write changes directly to the netlist.

For more information on using the Quartus II Chip Editor and Resource Property Editor, see the *Design Analysis and Engineering Change Management with Chip Editor* chapter in Volume 3 of the *Quartus II Handbook*.

*Figure 6–8. Delay Chain Shown in the Quartus II Resource Property Editor*



Resource Property Editor -- Netlist good and saved

Node name: Pin F25 -- |sample_project|clock

Type: Delay Chain
Input Pin to Logic Array Delay 0

| Properties/Modes | Values |
|---|---|
| OE Register Synchronous Mode | None |
| OE Power-Up | Low |
| Input Clock Enable Delay | Off |
| Output Clock Enable Delay | Off |
| Output Enable Clock Enable Delay | Off |
| Input Pin to Logic Array Delay 0 | Large |
| Input Pin to Logic Array Delay 1 | Large |
| Output Pin Delay | Off |
| Input Pin to Input Register Delay | Off |

Table 6–5 summarizes the programmable delays available for Altera devices.

*Table 6–5. Programmable Delays for Altera Devices (Part 1 of 3)*

| Programmable Delay | Description | I/O Timing Impact | Device Families |
|---|---|---|---|
| Decrease input delay to input register | Decreases propagation delay from an input pin to the data input of the input register in the I/O cell associated with the pin. Applied to input/bidirectional pin or register it feeds. | Decreases $t_{SU}$ Increases $t_H$ | Stratix, Stratix GX, Cyclone, APEX II, APEX 20KE, APEX 20KC, Mercury™, MAX 7000B |
| Input delay from pin to input register | Sets propagation delay from an input pin to the data input of the input register implemented in the I/O cell associated with the pin. Applied to input/bidirectional pin. | Changes $t_{SU}$ Changes $t_H$ | Stratix II, Cyclone II |

*Table 6–5. Programmable Delays for Altera Devices (Part 2 of 3)*

| Programmable Delay | Description | I/O Timing Impact | Device Families |
|---|---|---|---|
| Decrease input delay to internal cells | Decreases the propagation delay from an input or bidirectional pin to logic cells and embedded cells in the device. Applied to input/bidirectional pin or register it feeds. | Decreases $t_{SU}$ Increases $t_H$ | Stratix, Stratix GX, Cyclone, APEX II, APEX 20KE, APEX 20KC, Mercury, FLEX 10K®, FLEX® 6000, ACEX® 1K |
| Input delay from pin to internal cells | Sets the propagation delay from an input or bidirectional pin to logic and embedded cells in the device. Applied to a input or bidirectional pin. | Changes $t_{SU}$ Changes $t_H$ | Stratix II, Cyclone II, MAX II |
| Decrease input delay to output register | Decreases the propagation delay from the interior of the device to an output register in an I/O cell. Applied to input/bidirectional pin or register it feeds. | Decreases $t_{PD}$ | Stratix, Stratix GX, APEX II, APEX 20KE, APEX 20KC |
| Increase delay to output enable pin | Increases the propagation delay through the tri-state output to the pin. The signal can either come from internal logic or the output enable register in an I/O cell. Applied to output/bidirectional pin or register feeding it. | Increases $t_{CO}$ | Stratix, Stratix GX, APEX II, Mercury |
| Delay to output enable pin | Sets the propagation delay to an output enable pin from internal logic or the output enable register implemented in an I/O cell. | Changes $t_{CO}$ | Stratix II |
| Increase delay to output pin | Increases the propagation delay to the output or bidirectional pin from internal logic or the output register in an I/O cell. Applied to output/bidirectional pin or register feeding it. | Increases $t_{CO}$ | Stratix, Stratix GX, Cyclone, APEX II, APEX 20KE, APEX 20KC, Mercury |
| Delay from output register to output pin | Sets the propagation delay to the output or bidirectional pin from the output register implemented in an I/O cell. This option is off by default. | Changes $t_{CO}$ | Stratix II, Cyclone II |
| Increase input clock enable delay | Increases the propagation delay from the interior of the device to the clock enable input of an I/O input register. | N/A | Stratix, Stratix GX, APEX II, APEX 20KE, APEX 20KC |
| Input Delay from Dual Purpose Clock Pin to Fan-Out Destinations | Sets the propagation delay from a dual-purpose clock pin to its fan-out destinations that are routed on the global clock network. Applied to an input or bidirectional dual-purpose clock pin. | N/A | Cyclone II |

*Table 6–5. Programmable Delays for Altera Devices (Part 3 of 3)*

| Programmable Delay | Description | I/O Timing Impact | Device Families |
|---|---|---|---|
| Increase output clock enable delay | Increases the propagation delay from the interior of the device to the clock enable input of the I/O output register and output enable register. | N/A | Stratix, Stratix GX, APEX II, APEX 20KE, APEX 20KC |
| Increase output enable clock enable delay | Increases the propagation delay from the interior of the device to the clock enable input of an output enable register. | N/A | Stratix, Stratix GX |
| Increase $t_{ZX}$ delay to output pin | Used for zero bus-turnaround (ZBT) by increasing the propagation delay of the falling edge of the output enable signal. | Increases $t_{CO}$ | Stratix, Stratix GX, APEX II, Mercury |

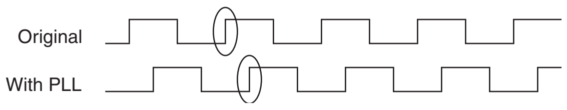
## Using Fast Regional Clocks in Stratix Devices

Stratix EP1S25, EP1S20, and EP1S10 devices and Stratix GX EP1SGX10 and EP1SGX25 devices contain two fast regional clock networks, `FCLK[1..0]`, in each quadrant, fed by input pins that can connect to other fast regional clock networks. In Stratix EP1S30, Stratix GX EP1SGX40, and larger devices in both families, there are two fast regional clock networks in each half-quadrant. Dedicated `FCLK` input pins can feed these clock nets directly. Fast regional clocks have less delay to I/O elements than regional or global clocks and are used for high fan-out control signals. Placing clocks on fast regional clock nets provides better $t_{CO}$ performance.

## Using PLLs to Shift Clock Edges

Using a PLL should improve I/O timing automatically. If the timing requirements are still not met, most devices allow the PLL to be phase shifted in order to change the I/O timing. Shifting the clock backwards gives a better $t_{CO}$ at the expense of the $t_{SU}$, while shifting it forward gives a better $t_{SU}$ at the expense of $t_{CO}$ and $t_H$. This technique can be used only in devices that offer PLLs with the phase shift option. See Figure 6–9.

*Figure 6–9. Shift Clock Edges Forward to Improve $t_{SU}$ at the Expense of $t_{CO}$*



## Improving Setup & Clock-to-Output Times Summary

Table 6–6 shows the recommended order in which to use techniques to reduce $t_{SU}$ and $t_{CO}$ times. Keep in mind that reducing $t_{SU}$ times increases hold ($t_H$) times.

*Table 6–6. Improving Setup & Clock-to-Output Times* Note (1)

| Technique | $t_{SU}$ | $t_{CO}$ |
|---|---|---|
| Ensure that the appropriate constraints are set for the failing I/Os | ✓ | ✓ |
| Use timing-driven compilation for I/O | ✓ | ✓ |
| Use fast input register | ✓ | |
| Use fast output register and fast output enable register | | ✓ |
| Set Decrease Input Delays to Input Register = ON or decrease the value of Input Delay from Pin to Input Register | ✓ | |
| Set Decrease Input Delays to Internal Cells = ON or decrease the value of Input Delay from Pin to Internal Cells | ✓ | |
| Set Increase Delay to Output Pin = OFF or decrease the value of Delay from Output Register to Output Pin | | ✓ |
| Use PLLs to shift clock edges | ✓ | ✓ |
| Use the Fast Regional Clock option | | ✓ |

*Note to Table 6–6:*
(1) These options may not apply for all device families.

Once I/O timing has been optimized, you can proceed to optimize $f_{MAX}$, as described in the "$f_{MAX}$ Timing Optimization Techniques (LUT-Based Devices)" section.

## $f_{MAX}$ Timing Optimization Techniques (LUT-Based Devices)

The next stage of design optimization is to improve the $f_{MAX}$ timing. There are a number of options available if the performance requirements are not achieved after compiling with the Quartus II software.

☞ It is important to understand your design and apply appropriate assignments to increase performance. It is possible to decrease performance if assignments are applied without full understanding of the design or the effect of the assignments.

## Synthesis Netlist Optimizations and Physical Synthesis Optimizations

The Quartus II software offers advanced netlist optimization options, including physical synthesis, for certain device families, to optimize your design further than the optimization performed in the course of the standard Quartus II compilation.

The effect of these options depends on the structure of your design, but netlist optimizations can help improve the performance of your design regardless of the synthesis tool used. Netlist optimizations can be applied both during synthesis and during fitting.

The synthesis netlist optimizations occur during the synthesis stage of the Quartus II compilation. Operating either on the output from another EDA synthesis tool or as an intermediate step in the Quartus II standard integrated synthesis, these optimizations make changes to the synthesis netlist that improve either area or speed, depending on your selected optimization technique.

The following synthesis netlist optimizations are available:

- **WYSIWYG Primitive Resynthesis**
- **Gate-level Register Re-timing**

You can view and modify the synthesis netlist optimization options on the **Synthesis Netlist Optimizations** page under **Analysis & Synthesis Settings** in the **Settings dialog** box (Assignments menu).

The physical synthesis optimizations take place during the Fitter stage of Quartus II compilation. Physical synthesis optimizations make placement-specific changes to the netlist that improve speed performance results for a specific Altera device.

The following physical synthesis optimizations are available:

- Physical synthesis for combinational logic
- Physical synthesis for registers:
  - Register duplication
  - Register retiming

You can also specify the **Physical synthesis effort**, which sets the level of physical synthesis optimization you want the Quartus II software to perform. You can specify the physical synthesis optimization options on the **Physical Synthesis Optimizations** page under **Fitter Settings** in the **Settings** dialog box (Assignments menu).

For more information and detailed descriptions of these netlist optimization options, see the *Netlist Optimizations & Physical Synthesis* chapter in Volume 2 of the *Quartus II Handbook*.

To achieve the best results, use these options in different combinations. Performance results are design dependant. Typical benchmark results with netlists from a leading third-party synthesis tool and compiled with the Quartus II software version 4.1 are shown in Table 6–7. These results were obtained for Stratix devices, using various designs and numbers of LEs.

***Table 6–7. Average Performance of Different Netlist Optimizations***

| Optimization Method | $f_{MAX}$ Gain (%) | Win Ratio (%) *(1)* | Winner's $f_{MAX}$ Gain (%) *(2)* | Change Logic (%) | Increase in Compile Time (×) |
|---|---|---|---|---|---|
| WYSIWYG primitive resynthesis | 2 | 60 | 6 | -8 | 1.0 |
| Physical synthesis for combinational logic and registers | | | | | |
| Using physical synthesis Fast effort level | 10 | 86 | 14 | 4 | 1.7 |
| Using physical synthesis Normal effort level | 15 | 86 | 14 | 4 | 2.7 |
| Using physical synthesis Extra effort level | 17 | 86 | 14 | 4 | 4.2 |
| WYSIWYG primitive re-synthesis as well as physical synthesis for combinational logic and registers | | | | | |
| Using physical synthesis Fast effort level | 12 | 87 | 16 | -5 | 1.7 |
| Using physical synthesis Normal effort level | 17 | 87 | 16 | -5 | 2.7 |
| Using physical synthesis Extra effort level | 19 | 87 | 16 | -5 | 4.2 |
| All options on (WYSIWYG primitive re-synthesis, gate level register re-timing, and physical synthesis for combinational logic and registers) | | | | | |
| Using physical synthesis Extra effort level | 19 | 82 | 17 | -6 | 4.3 |

***Notes to Table 6–7:***

(1) Win is the percentage of designs that showed better performance with the option on, than without the option on.

(2) Winner's $f_{MAX}$ gain refers to the average improvement for the designs that showed better performance with these settings (designs considered a Win).

The results for the **WYSIWYG primitive re-synthesis** option depend on the **Optimization Technique** selected on the **Analysis & Synthesis** page of the **Settings** dialog box (Assignments menu). These results use the default **Balanced** setting. Changing the setting to **Speed** or **Area** can affect your results.

The DSE Tcl/Tk script can automate successive compilations of a design, each employing different netlist optimization options.

For more information on the DSE script, see the *Design Space Explorer* chapter in Volume 2 of the *Quartus II Handbook*.

## Seed

Changing the seed affects the initial placement configuration and often causes different Fitter results. To obtain a better $f_{MAX}$ value, you can experiment with different settings. This method should only be attempted if the design is finalized and is failing timing on a small number of paths. The $f_{MAX}$ variation is typically about 3% for Stratix devices.

Changing the seed changes Fitter results because all Fitter algorithms have random variations when initial conditions change, and changing the seed takes advantage of this behavior. However, note that if anything in the design changes, the results from seed to seed changes.

The seed for initial placement is controlled by the **Seed** setting on the **Fitter Settings** page of the **Settings** dialog box (Assignments menu).

The DSE Tcl/Tk script can automate successive compilations of a design, each employing different seeds.

For more information on the DSE script, see the *Design Space Explorer* chapter in Volume 2 of the *Quartus II Handbook*.

## Optimize Synthesis for Speed

The manner in which the design is synthesized has a large impact on its performance. Performance varies depending on the way the design is coded, which synthesis tool is used, and which options are specified when synthesizing. Synthesis options should be changed if a large number of paths are failing or specific paths are failing by a large amount and have many levels of logic.

Ensure that you have set your device and timing constraints correctly in your synthesis tool. Your synthesis tool tries to meet the specified requirements. If a target frequency is not specified, some synthesis tools optimize for area.

To achieve best performance with push-button compilation, use the recommendations in the following paragraphs.

For information on setting timing requirements and synthesis options in other synthesis tools, see the appropriate chapter in the *Synthesis* section in Volume 1 of the *Quartus II Handbook*, or see your synthesis software's documentation.

You can use the DSE to experiment with different Quartus II synthesis options to optimize for best performance.

For more information, see the *Design Space Explorer* chapter in Volume 2 of the *Quartus II Handbook*.

### *Optimize for Speed, Not Area*

Most synthesis tools optimize to meet your speed requirements. Some synthesis tools offer an easy way to optimize for speed instead of area. For the Quartus II integrated synthesis, specify **Speed** as the **Optimization Technique** option on the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu). You can also specify this logic option for specific modules in your design with the Assignment Editor while leaving the default **Optimization Technique** setting at **Balanced** (for the best trade-off between area and speed for certain device families) or **Area** (if area is an important concern).

### *Flatten the Hierarchy*

Synthesis tools typically provide the option of preserving hierarchical boundaries, which may be useful for verification or other purposes. However, optimizing across hierarchical boundaries allows the synthesis tool to perform the most logic minimization, which may improve performance. Therefore, whenever possible, flatten your design hierarchy to achieve best results. If you are using the Quartus II integrated synthesis, ensure that the **Preserve Hierarchical Boundary** logic option is turned off.

### *Set the Synthesis Effort to High (where applicable)*

Some synthesis tools offer varying synthesis effort levels to trade off compilation time with synthesis results. Set the synthesis effort to high to achieve best results.

### *Change State Machine Encoding*

State machines can be encoded using various techniques. One-hot encoding, which uses one register for every state bit, usually provides the best performance. If your design contains state machines, changing the state machine encoding to one-hot can improve performance at the cost of area.

If your design does not manually encode the state bits, you can select the state machine encoding chosen in your synthesis tool. In Quartus II integrated synthesis, choose **One-Hot** for **State Machine Processing** on the **Analysis & Synthesis Settings** page of the **Settings** dialog box

(Assignments menu). You can also specify this logic option for specific modules or state machines in your design with the Assignment Editor. In some cases (especially in Stratix II devices), other encoding styles can offer better performance. You can experiment with different encoding styles to see what effect the style has on your resource utilization and timing performance.

#### *Duplicate Logic for Fan-Out Control*

Duplicating logic or registers can help improve timing in cases where moving a register in a failing timing path to reduce routing delay creates other failing paths, or where there are timing problems due to the fan-out of the registers.

Many synthesis tools support options or attributes to set the maximum fan-out of a register. In the Quartus II integrated synthesis, you can set the **Maximum Fan-Out** logic option in the Assignment Editor to control the number of destinations for a node so that the fan-out count does not exceed a specified value. You can also use the `maxfan` attribute in your HDL code. The software duplicates the node as needed to achieve the specified maximum fan-out.

You can manually duplicate registers in the Quartus II software regardless of the synthesis tool used. To duplicate a register, apply the **Manual Logic Duplication** option to the register with the Assignment Editor. For more information on the **Manual Logic Duplication** option, see the Quartus II Help.

#### *Other Synthesis Options*

With your synthesis tool, experiment with the following options if they are available:

- Register balancing or retiming
- Register pipelining

### LogicLock Assignments

You can make LogicLock assignments for optimization based on nodes, design hierarchy, or critical paths. This method can be used if a large number of paths are failing, but recoding the design is thought to be unnecessary. LogicLock assignments can help if routing delays form a large portion of your critical path delay, and placing logic closer together on the device will help improve the routing delay.

Note that improving fitting results, especially for larger devices such as Stratix and Stratix II, can be difficult. LogicLock assignments will not always improve the performance of the design. In many cases you will not be able to improve upon the results from the Fitter.

When making LogicLock assignments, it is important to consider how much flexibility to leave the Fitter. LogicLock assignments provide more flexibility than hard location assignments. Assignments that are more flexible require higher Fitter effort, but reduce the chance of design over-constraint. The following types of LogicLock assignments are available, listed in order of decreasing flexibility:

- Soft LogicLock regions
- Auto size, floating location regions
- Fixed size, floating location regions
- Fixed size, locked location regions

To determine what to put into a LogicLock region, see the timing analysis results and the Timing Closure Floorplan. The register-to-register $f_{MAX}$ paths in the Timing Analyzer section of the Compilation Report can provide a helpful method of recognizing patterns. The following paragraphs describe cases in which LogicLock regions can help to optimize a design.

For more information on the LogicLock design methodology, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook*.

### *Hierarchy Assignments*

For a design with the hierarchy shown in Figure 6–10, which has failing paths in the timing analysis results similar to those shown in Table 6–8, `mod_A` is probably a problem module. In this case, `mod_A` could be placed in a LogicLock region to attempt to put all the nodes in the module closer together in the floorplan.

*Figure 6–10. Design Hierarchy*

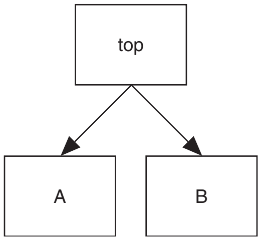


Table 6–8 shows the failing module paths in timing analysis.

*Table 6–8. Failing Module Paths in Timing Analysis*

| | |
|---|---|
| \|mod_A\|reg1 | \|mod_A\|reg9 |
| \|mod_A\|reg3 | \|mod_A\|reg5 |
| \|mod_A\|reg4 | \|mod_A\|reg6 |
| \|mod_A\|reg7 | \|mod_A\|reg10 |
| \|mod_A\|reg0 | \|mod_A\|reg2 |

### Path Assignments

If you see a pattern such as the one shown in Figure 6–11 and Table 6–9, it is probably an indication of paths with a common problem. In this case, a path-based assignment could be made from all `d_reg` registers to all `memaddr` registers. A path-based assignment can be made to place all source registers, destination registers, and the nodes between them in a LogicLock region using the wild cards characters "*" and "?".

You can also explicitly place the nodes of a critical path in a LogicLock region. There may be alternate paths between the source and destination registers that could become critical if you use this method instead of path-based assignments.

For information on making path-based assignments, using wild cards, and individual node assignments, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook*.

*Figure 6–11. Failing Paths in Timing Analysis*

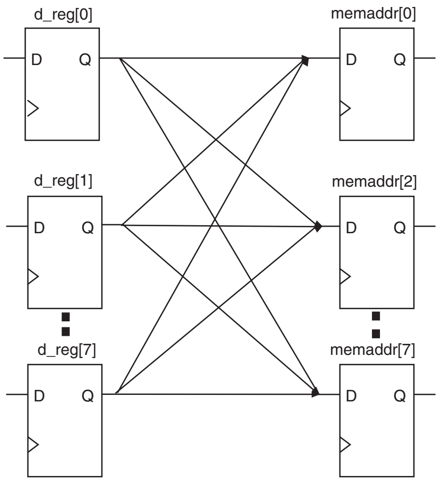


*Table 6–9. Failing Paths in Timing Analysis*

| From | To |
|---|---|
| \|d_reg[1] | \|memaddr[5] |
| \|d_reg[1] | \|memaddr[6] |
| \|d_reg[1] | \|memaddr[7] |
| \|d_reg[2] | \|memaddr[0] |
| \|d_reg[2] | \|memaddr[1] |

## Location Assignments & Back Annotation

If a small number of paths are failing, you can use hard location assignments to optimize placement. Location assignments are less flexible for the Quartus II Fitter than LogicLock assignments. In some cases when you are very familiar with your design, you may be able to enter location constraints in a way that produces better results than the Quartus II Fitter.

☞ Note that improving fitting results, especially for larger devices such as Stratix and Stratix II, can be difficult; location assignments will not always improve the performance of the design. In many cases you will not be able to improve upon the results from the Fitter.

The following are commonly used location assignments, listed in order of decreasing flexibility:

- Custom regions
- Back-annotated LAB location assignments
- Back-annotated LE or ALM location assignments

### *Custom Regions*

A custom region is a rectangular region containing user-assigned nodes. These assigned nodes are then constrained in the region's boundaries. If any portion of a block in the device floorplan overlaps with a custom region, such as part of a M-RAM, it is considered to be entirely in that region.

Custom regions are hard location assignments that cannot be overridden and are very similar to fixed-size, locked-location LogicLock regions. Custom regions are commonly used when logic must be constrained to a specific portion of the device.

### *Back Annotation and Manual Placement*

Fixing the location of nodes in a design in the locations resulting from the last compilation is known as back-annotation. When all the nodes are back-annotated, manually moving nodes does not affect the locations of other design nodes that are locked down. This is referred to as manual placement.

☞ Locking down node locations is very restrictive to the Compiler, so you should only back-annotate when the design has been finalized and no further changes are expected. The assignments may become invalid if the design is changed. Combinational nodes often change names when a design is resynthesized, even if they are unrelated to the logic that was changed.

☞ Moving nodes manually can be very difficult for large devices, and in many cases you will not be able to improve upon the results from the Fitter.

☞ Illegal or unroutable location constraints may cause “no fit” errors.

Before making location assignments, determine whether to lock down the location of all nodes in the design. When you are using a hierarchical design flow, you can choose to lock down node locations in only one LogicLock region, while the other node locations are left as floating in a fixed LogicLock region. A hierarchical approach using the LogicLock design methodology can reduce the dependence of logic blocks with other logic blocks in the device.

For more information on a block-based design approach, see the *Hierarchical Block-Based Design & Team-Based Design Flows* chapter in Volume 1 of the *Quartus II Handbook*.

When you back-annotate a design, you can choose that the nodes be assigned either to LABs (this is preferred because of increased flexibility) or LEs/ALMs. You can also choose to back-annotate routing to further restrict the Fitter and force a specific routing within the device.

Using back-annotated routing with physical synthesis optimizations may cause a routing failure.

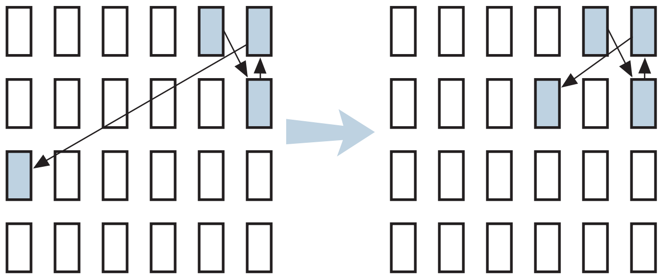
For more information on back-annotation of routing, see Quartus II Help.

When performing manual placement on a detailed level, Altera suggests that you move LABs, not logic cells (LEs or ALMs). The Quartus II software places nodes that share the same control signals in appropriate LABs. Successful place-and-route is more difficult when you move individual logic cells.

In general, when you are performing manual place-and-route, it is best to fix all I/O paths first. This is because there are often fewer options available to meet I/O timing. After I/O timing has been met, focus on manually placing $f_{MAX}$ paths. This strategy follows the methodology outlined in this chapter.

The best way to meet performance is to move nodes closer together. For a critical path such as the one shown in Figure 6–12, moving the destination node closer to the other nodes reduces the delay and may cause it to meet your timing requirements.

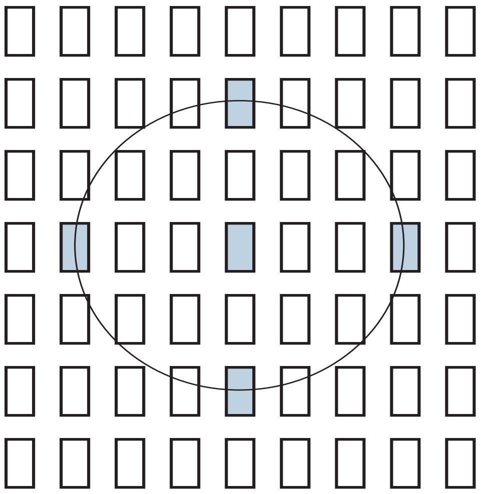
*Figure 6–12. Reducing Delay of Critical Path*



### *Optimizing Placement for Stratix II, Stratix, Stratix GX, & Cyclone II Devices*

In Stratix II, Stratix, Stratix GX, and Cyclone II architectures, the row interconnect delay is slightly faster than the column interconnect delay. Therefore, when placing nodes, optimal placement is typically an ellipse around the source or destination node. In Figure 6–13, if the source is located in the center, any of the shaded LABs should give approximately the same delay.

*Figure 6–13. Possible Optimal Placement Ellipse*



In addition, you should avoid crossing any M-RAM memory blocks for node-to-node routing, if possible, because routing paths across M-RAM blocks requires using *R24* or *C16* routing lines.

To determine the actual delays to and from a resource, use the **Show Physical Timing Estimate** feature in the Timing Closure Floorplan.

For more information on using the Timing Closure Floorplan, see the *Timing Closure Floorplan* chapter in Volume 2 of the *Quartus II Handbook*.

### *Optimizing Placement for Cyclone Devices*

In Cyclone devices, the row and column interconnect delays are similar; therefore, when placing nodes, optimal placement is typically a circle around the source or destination node.

Try to avoid long routes across the device because they require more than one routing line to cross the Cyclone device.

### *Optimizing Placement for Mercury, APEX II, & APEX 20KE/C Devices*

For the Mercury, APEX II, and APEX 20KE/C architectures, the delay for paths should be reduced by placing the source and destination nodes in the same geographical resource location. The following list shows the device resources in order from fastest to slowest:

- LAB
- MegaLAB™ structure
- MegaLAB column
- Row

For example, if the nodes cannot be place in the same MegaLAB structure to reduce the delay, they should be place in the same MegaLAB column. For the actual delays to and from resources, use the **Show Physical Timing Estimate** feature in the Timing Closure Floorplan.

## Optimize Source Code

If the methods described in the preceding sections do not sufficiently improve the timing in the design, you must modify the design at the source to achieve the desired results. You may be able to rearchitect the design using pipelining or more efficient coding techniques. In many cases, optimizing the design's source code can have a very significant effect on your design performance. In fact, optimizing your source code is often a better choice of optimization than using LogicLock or location assignments.

If your critical path involved memory or DSP functions, check whether you have code blocks in your design that describe memory or DSP functions that are not being inferred and placed in dedicated logic. You may be able to modify your source code to allow these functions to be placed into high-performance dedicated memory or DSP resources in the target device.

For coding style guidelines including examples of HDL code for inferring memory and DSP functions, refer to the *Inferring and Instantiating Altera Megafunctions* section of the *Recommended HDL Coding Styles* chapter in Volume 1 of the *Quartus II Handbook*.

Ensure that your state machines are recognized as state machine logic and optimized appropriately in your synthesis tool. State machines that are recognized are generally optimized better than if the synthesis tool treats them as generic logic. In the Quartus II software, you can check for the **State Machine** report under **Analysis & Synthesis** in the **Compilation Report** (Processing menu). This report provides details, including the state encoding for each state machine that was recognized during compilation. If your state machine is not being recognized, you may need to change your source code to enable it to be recognized.

For guidelines and sample HDL code for state machines, refer to the *State Machines* section in the *Recommended HDL Coding Styles* chapter in Volume 1 of the *Quartus II Handbook*.

## Improving $f_{MAX}$ Summary

The choice of options and the adjustment of settings to improve $f_{MAX}$ depends on the failing paths in the design. To achieve the best results relative to your performance requirements, apply the following options, compiling after each:

1. Apply netlist optimization options (including physical synthesis).

2. Modify the seed. (This step may be omitted if a large number of critical paths are failing, or if paths are failing by large amounts.)

3. Apply synthesis options to optimize for speed.

4. Use the DSE Tcl/Tk script as appropriate to automate successive compilations of a design, each employing the different options in steps 1 through 3.

5. Make LogicLock assignments.

6. Make location assignments, or perform manual placement by back-annotating the design.

If these options do not achieve performance requirements, design source code modifications may be required.

For more information on the DSE script, see the *Design Space Explorer* chapter in Volume 2 of the *Quartus II Handbook*.

## Optimization Techniques for Macrocell-Based (MAX 7000 and MAX 3000) CPLDs

This section of the chapter addresses resource and timing optimization issues for Macrocell-based Altera devices, MAX 7000 and MAX 3000 CPLD device families.

For information on optimizing FPGA and MAX II CPLD designs, refer to "Optimization Techniques for LUT-Based (FPGA and MAX II) Devices" on page 6–12. For information on optimizing compilation time (when targeting any device), refer to "Compilation Time Optimization Techniques" on page 6–55.

## Resource Utilization Optimization Techniques (Macrocell-based CPLDs)

The following recommendations will help you take advantage of the macrocell-based architecture in the MAX 7000 and MAX 3000 device families to yield maximum speed, reliability, and device resource utilization while minimizing fitting difficulties.

After design analysis, the first stage of design optimization is to improve resource utilization. Complete this stage before proceeding to timing optimization. First, ensure that you have set the basic constraints described in "Initial Compilation" on page 6–2. If your design is not fitting into a specified device, use the techniques in this section to achieve a successful fit.

### Use Dedicated Inputs for Global Control Signals

MAX 7000 and MAX 3000 devices have four dedicated inputs that can be used for global register control. Because the global register control signals can bypass the logic cell array and directly feed registers, product terms for primary logic can be preserved. Also, because each signal has a dedicated path into the LAB, global signals can also bypass logic and data path interconnect resources.

Because the dedicated input pins are designed for high fan-out control signals and provide low skew, you should always assign global signals (e.g., clock, clear, and output enable) to the dedicated input pins.

You can use logic-generated control signals for global control signals instead of dedicated inputs. However, the disadvantages to using logic-generated controls signals include:

- More resources are required (i.e., logic cells, interconnect)
- May result in more data skew
- If the logic-generated control signals have high fan-out, the design may be more difficult to fit

By default, the Quartus II software uses dedicated inputs for global control signals automatically. You can assign the control signals to dedicated input pins in one of four ways:

- In the Assignment Editor, choose one of two methods:
  - Assign pins to dedicated pin locations
  - Assign global signal settings to the pins
- Choose **Register Control Signals** in the **Auto Global Options** section of the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu)
- Insert a global primitive after the pins

If you have already assigned pins in the MAX+PLUS II software for the design, choose **Import Assignments** (Assignments menu).

## Reserve Device Resources

Because pin and logic option assignments might be necessary for board layout and performance requirements, and because full utilization of the device resources may cause the design to be more difficult to fit, Altera recommends that you leave 10% of the device's logic cells and 5% of the I/O pins unused to accommodate future design modifications. Following the Altera-recommended device resource reservation guidelines increases the chance that the Quartus II software will be able to fit the design during recompilation after changes or assignments have been made.

## Pin Assignment Guidelines & Procedures

Sometimes user-specified pin assignments are necessary for board layout. This section discusses pin assignment guidelines and procedures.

To minimize fitting issues with pin assignments, follow these guidelines:

- Assign speed-critical control signals to dedicated inputs
- Assign output enables to appropriate locations
- Estimate fan-in to assign output pins to appropriate LAB
- Assign output pins in need of parallel expanders to macrocells numbered 4 to 16

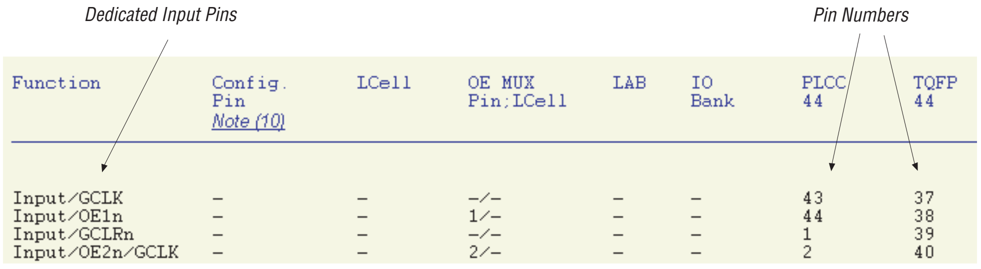
Altera recommends that you allow the Quartus II software to automatically choose pin assignments when possible.

### Control Signal Pin Assignments

You should assign speed-critical control signals to dedicated input pins. Every MAX 7000 and MAX 3000 device has four dedicated input pins (GCLK1, OE2/GCLK2, OE1, GCLRn). You can assign clocks to global clock dedicated inputs (GCLK1, OE2/GCLK2), clear to the global clear dedicated input (GCLRn), and speed-critical output enable to global OE dedicated inputs (OE1, OE2/GCLK2).

Figure 6–14 shows the EPM3032A device's pin-out information for the dedicated pins. You can use the Quartus II Help to determine the dedicated input pin numbers.

**Figure 6–14. Quartus II Help EPM3032A Dedicated Pin-Out Information**



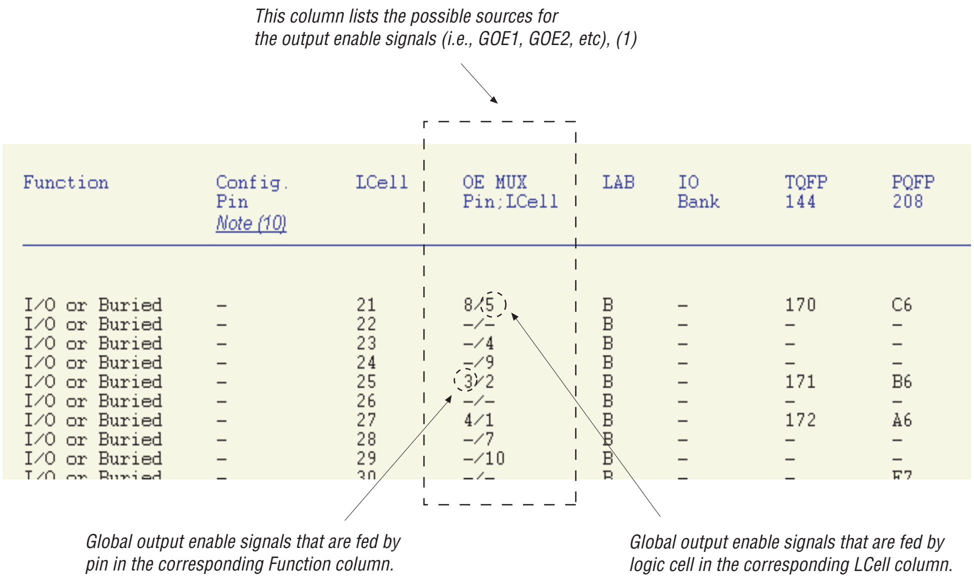
| Function | Config. Pin Note (10) | LCell | OE MUX Pin;LCell | LAB | IO Bank | PLCC 44 | TQFP 44 |
|---|---|---|---|---|---|---|---|
| Input/GCLK | – | – | –/– | – | – | 43 | 37 |
| Input/OE1n | – | – | 1/– | – | – | 44 | 38 |
| Input/GCLRn | – | – | –/– | – | – | 1 | 39 |
| Input/OE2n/GCLK | – | – | 2/– | – | – | 2 | 40 |

### Output Enable Pin Assignments

Occasionally, because the total number of required output enable pins is more than the dedicated input pins, output enable signals may need to be assigned to I/O pins. Therefore, to minimize the possibility of fitting errors, refer to Quartus II Help when assigning the output enable pins for MAX 7000 and MAX 3000 devices. Search for the device name (e.g., EPM3032A) in Quartus II Help to bring up the device pin table with output enable information.

Figure 6–15 shows the dedicated pin-out information for the EPM3512A device from Quartus II Help. Specifically, Figure 6–15 shows that the first row *Pin;LCell* value is 8/5; which means that GOE8 can be driven by pin 170 or C6 (depending on package) and GOE5 can be driven by logic cell 21.

*Figure 6–15. Quartus II Help EPM3512A Dedicated Pin-Out Information*


This column lists the possible sources for the output enable signals (i.e., GOE1, GOE2, etc), (1)
Global output enable signals that are fed by pin in the corresponding Function column.
Global output enable signals that are fed by logic cell in the corresponding LCell column.

| Function | Config. Pin Note (10) | LCell | OE MUX Pin;LCell | LAB | IO Bank | TQFP 144 | PQFP 208 |
|---|---|---|---|---|---|---|---|
| I/O or Buried | – | 21 | 8/5 | B | – | 170 | C6 |
| I/O or Buried | – | 22 | –/– | B | – | – | – |
| I/O or Buried | – | 23 | –/4 | B | – | – | – |
| I/O or Buried | – | 24 | –/9 | B | – | – | – |
| I/O or Buried | – | 25 | 3/2 | B | – | 171 | B6 |
| I/O or Buried | – | 26 | –/– | B | – | – | – |
| I/O or Buried | – | 27 | 4/1 | B | – | 172 | A6 |
| I/O or Buried | – | 28 | –/7 | B | – | – | – |
| I/O or Buried | – | 29 | –/10 | B | – | – | – |
| I/O or Buried | – | 30 | –/– | B | – | – | E7 |

### Estimate Fan-In When Assigning Output Pins

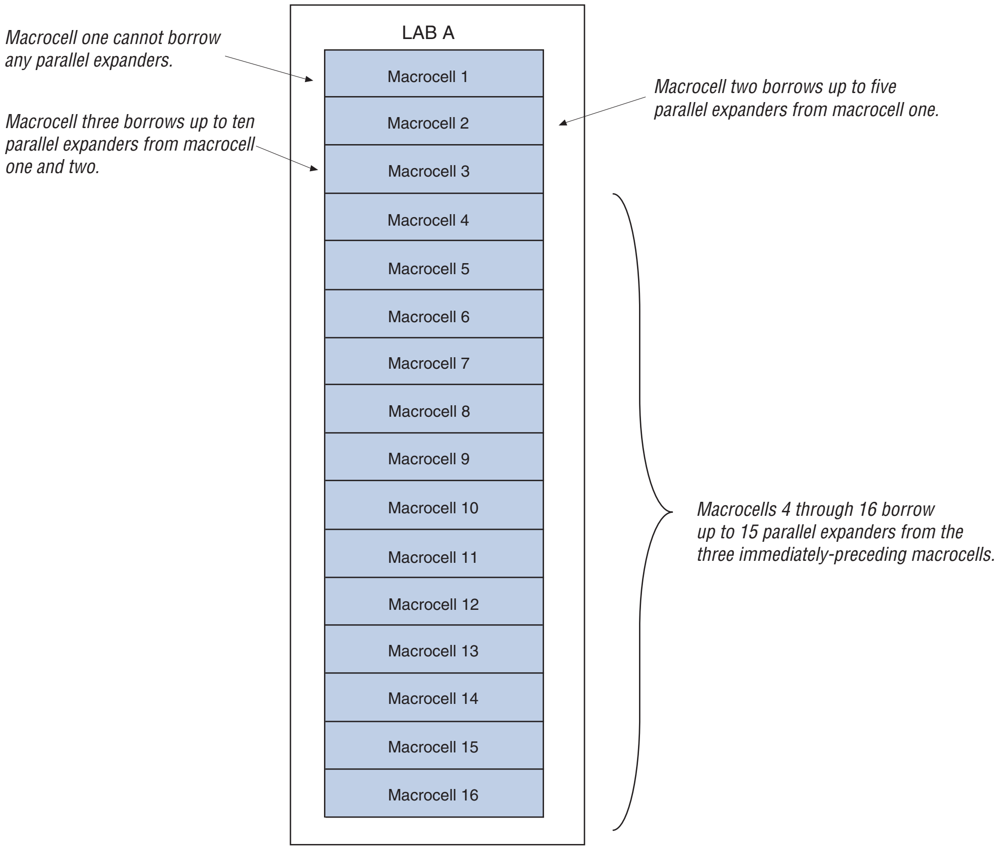
Macrocells with high fan-in can cause more placement problems for the Quartus II Fitter than those with low fan-in. The maximum number of fan-in per LAB should not exceed 36 in MAX 7000 and MAX 3000 devices. Therefore, it is important to estimate the fan-in of logic (e.g., x-input AND gate) that feeds each output pin. If the total fan-in of logic that feeds each output pin in the same LAB exceeds 36, compilation may fail. To save resources and prevent compilation errors, avoid assigning pins that have high fan-in.

### Outputs Using Parallel Expander Pin Assignments

Figure 6–16 illustrates how parallel expanders are used within a LAB. MAX 7000 and MAX 3000 devices contain chains that can lend or borrow parallel expanders. The Quartus II Fitter places macrocells in a location that allows them to lend and borrow parallel expanders appropriately.

As shown in Figure 6–16, only macrocells 2 through 16 can borrow parallel expanders. Therefore, you should assign output pins that may need parallel expanders to pins adjacent to macrocells 4 through 16. Altera recommends using macrocells 4 through 16 because they can borrow the largest number of parallel expanders.

*Figure 6–16. LAB Macrocells & Parallel Expander Associations*


Macrocell one cannot borrow any parallel expanders.
Macrocell three borrows up to ten parallel expanders from macrocell one and two.
LAB A
Macrocell 1
Macrocell 2
Macrocell 3
Macrocell 4
Macrocell 5
Macrocell 6
Macrocell 7
Macrocell 8
Macrocell 9
Macrocell 10
Macrocell 11
Macrocell 12
Macrocell 13
Macrocell 14
Macrocell 15
Macrocell 16
Macrocell two borrows up to five parallel expanders from macrocell one.
Macrocells 4 through 16 borrow up to 15 parallel expanders from the three immediately-preceding macrocells.

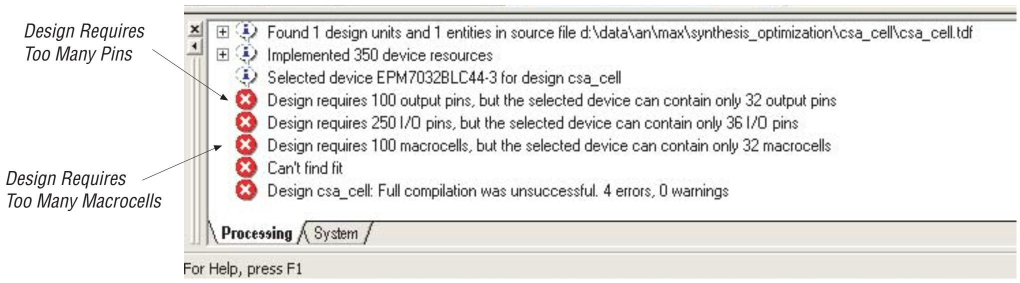
## Resolving Resource Utilization Problems

During compilation with the Quartus II software, you may receive an error message (see Figure 6–17) alerting you that the compilation was not successful.

There are two common Quartus II compilation fitting issues: macrocell usage and routing resources. Macrocell usage errors occur when the total number of macrocells in the design exceeds the available macrocells in the device. Routing errors occur when the available routing resources cannot implement the design. To resolve your design issues, check the Message Window (see Figure 6–17) for the no-fit compilation results.

Messages in the Message Window are also copied in the Report Files. Right-click on a message and select **Help** (right button pop-up menu) for more information.

*Figure 6–17. Quartus II Software Compilation No-Fit Error Message WIndow*


Design Requires Too Many Pins
Design Requires Too Many Macrocells
Processing
System

```
Found 1 design units and 1 entities in source file d:\data\an\max\synthesis_optimization\csa_cell\csa_cell.tdf
Implemented 350 device resources
Selected device EPM7032BLC44-3 for design csa_cell
Design requires 100 output pins, but the selected device can contain only 32 output pins
Design requires 250 I/O pins, but the selected device can contain only 36 I/O pins
Design requires 100 macrocells, but the selected device can contain only 32 macrocells
Can't find fit
Design csa_cell: Full compilation was unsuccessful. 4 errors, 0 warnings
```

For Help, press F1

### *Resolving Macrocell Usage Issues*

Occasionally, a design requires more macrocell resources than are available in the selected device, resulting in a no-fit compilation. The following list provides tips for resolving macrocell-usage issues as well as tips to minimize the amount of macrocells used:

- Turn off **Auto Parallel Expanders** on the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu)—If the design's clock frequency ($f_{MAX}$) is not an important part of the design requirements, you should turn off the parallel expanders for all or part of the project. The design will usually require more macrocells if parallel expanders are turned on.
- Change **Optimization Technique** from **Speed** to **Area**—An algorithm that is written to give preference to device-fitting rather than device-speed ($f_{MAX}$) is selected when the Area Optimization technique is enabled. As expected, the device-fitting algorithm produces a slower compilation result. You can change the **Optimization Technique** option in the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu).
- Use D-flip-flops instead of latches—Altera recommends that you always use D-flip-flops instead of latches in your design because D-flip-flops may reduce the macrocell fan-in, and thus reduce macrocell usage. The Quartus II software uses extra logic to implement latches in MAX 7000 and MAX 3000 designs because individual MAX 7000 and MAX 3000 macrocells contain D-flip-flops instead of latches.

- Use asynchronous clear and preset instead of synchronous clear and preset—To reduce the product term usage, use asynchronous clear and preset in your design whenever possible. Using other control signals such as synchronous clear will produce macrocells and pins with higher fan-out.

☞ If you have followed the suggestions listed in this section and your project still does not fit the targeted device, consider using a larger device. When upgrading to a different density device, the vertical-package-migration feature of the MAX 7000 and MAX 3000 device families allows pin assignments to be maintained.

### *Resolving Routing Issues*

The other resource that can cause design-fitting issues is routing. For example, if the total fan-in into a LAB exceeds the maximum allowed, the result may be a no-fit error during compilation. If your design does not fit the targeted device because of routing issues, consider the following suggestions:

- Use dedicated inputs/global signals for high fan-out signals—The dedicated inputs in MAX 7000 and MAX 3000 devices are designed for speed-critical and high fan-out signals. Therefore, Altera recommends that you always assign high fan-out signals to dedicated inputs/global signals.
- Change the **Optimization Technique** option from **Speed** to **Area**—This option may resolve routing resource and the macrocell usage issues. See the same suggestion in "Resolving Macrocell Usage Issues" on page 6–46.
- Reduce the fan-in per cell—If you are not limited by the number of macrocells used in the design, you can use the **Fanin per cell (%)** option to reduce the fan-in per cell. The allowable values are 20-100% and the default value is 100%. Reducing the fan-in can reduce localized routing congestion but increase the macrocell count. You can set this logic option in the **Assignment Editor** (Assignments menu) or under **More Settings** in the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu).
- Turn off **Auto Parallel Expanders** in the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu)—By turning off the parallel expanders, the Quartus II software will have more fitting flexibility for each macrocell, i.e., allowing macrocells to relocate. For example, each macrocell (previously grouped together in the same LAB) may move to a different LAB to reduce routing constraints.

- Inserting logic cells—Inserting logic cells reduces fan-in and shared expanders used per macrocell, increasing routability. By default, the Quartus II software will automatically insert logic cells when necessary. You can turn this feature off by turning off **Auto Logic Cell Insertion** under **More Settings** in the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu). See "Using LCELL Buffers to Reduce Required Resources" on page 6–48 for more information.
- Change pin assignments—If you are willing to discard your pin assignments, you can let the Quartus II Fitter automatically ignore all the assignments, the minimum number of assignments, or specific assignments.

> If you prefer reassigning the pins to increase the device-routing efficiency, refer to "Pin Assignment Guidelines & Procedures" on page 6–42.

### *Using LCELL Buffers to Reduce Required Resources*

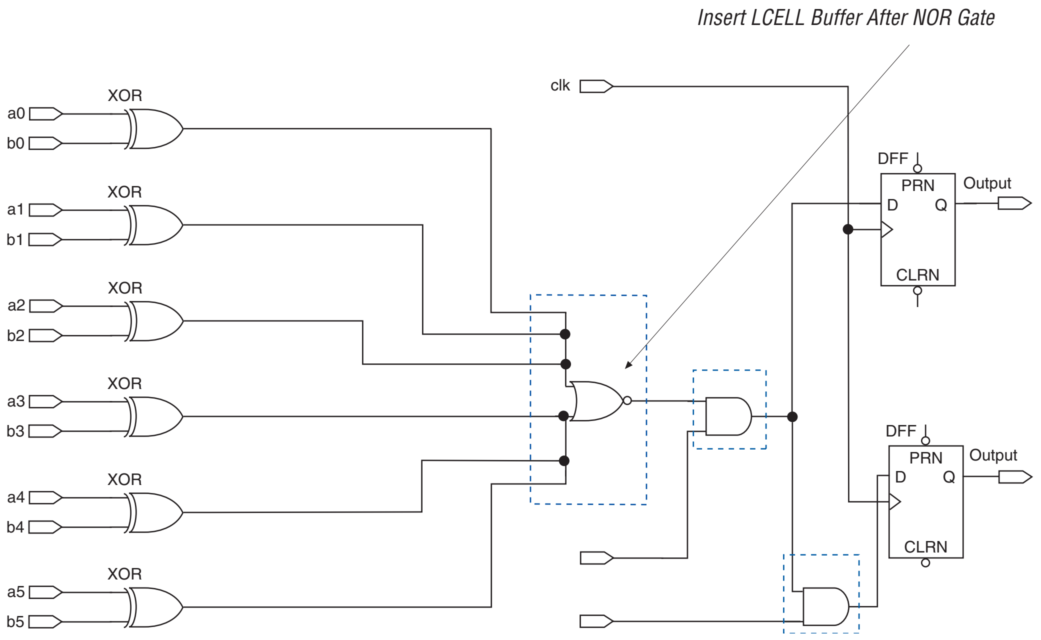
Complex logic, such as multi-level XOR gates, will often be implemented with more than one macrocell. When this occurs, the Quartus II software automatically allocates shareable expanders—or additional macrocells (called synthesized logic cells)—to supplement the logic resources that are available in a single macrocell. You can also break down complex logic by inserting logic cells in the project to reduce the average fan-in and total number of shareable expanders needed. Manually inserting logic cells can provide greater control over speed-critical paths.

Instead of using the Quartus II software's **Auto Logic Cell Insertion** option, you can manually insert logic cells. However, Altera recommends that you use the **Auto Logic Cell Insertion** option unless you know which part of the design is causing the congestion.

A good location to manually insert LCELL buffers is where a single complex logic expression feeds multiple destinations in your design. You can insert an LCELL buffer just after the complex expression; the Quartus II Fitter extracts this complex expression and places it in a separate logic cell. Rather than duplicating all the logic for each destination, the Quartus II software feeds the single output from the logic cell to all destinations.

To reduce fan-in and prevent no-fit compilations caused by routing resource issues, insert an LCELL buffer after a NOR gate, see Figure 6–18. The Figure 6–18 design was compiled for a MAX 7000AE device. Without the LCELL buffer, the design requires two macrocells, eight shareable expanders, and the average fan-in is 14.5. However, with the LCELL buffer, the design requires three macrocells, eight shareable expanders, and the average fan-in is just 6.33.

***Figure 6–18. Reducing the Average Fan-In by Inserting LCELL Buffers***



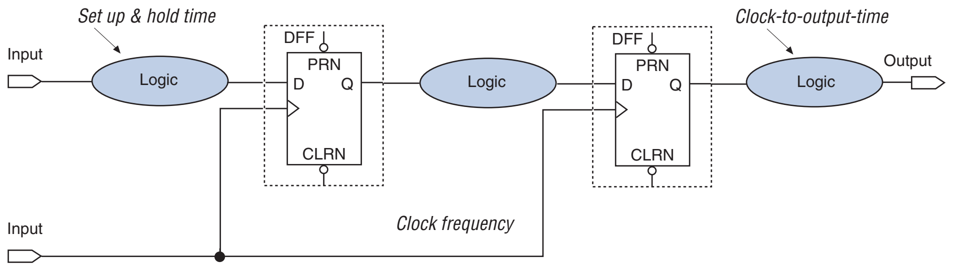
## Timing Optimization Techniques (Macrocell-based CPLDs)

The stage of design optimization after resource optimization focuses on timing. Ensure that you have made the appropriate assignments as described in "Initial Compilation" on page 6–2, and that the resource utilization is satisfactory, before proceeding with timing optimization.

Maintaining the system's performance at or above certain timing requirements is an important goal of circuit designs. The five main timing parameters that determine a design's system performance are: setup time ($t_{SU}$), hold time ($t_H$), clock-to-output time ($t_{CO}$), pin-to-pin delays ($t_{PD}$), and maximum clock frequency ($f_{MAX}$). The setup and hold times are the propagation time for input data signals. Clock-to-output time is the propagation time for output signals, pin-to-pin delay is the time required for a signal from an input pin to propagate through combinational logic and appear at an external output pin, and the maximum clock frequency is the internal register-to-register performance.

This section provides guidelines to improve the timing if the timing requirements are not met. Figure 6–19 shows the parts of the design that determine the $t_{SU}$, $t_H$, $t_{CO}$, $t_{PD}$, and $f_{MAX}$ timing parameters.

*Figure 6–19. Main Timing Parameters That Determine the System's Performance*



Timing results for $t_{SU}$, $t_H$, $t_{CO}$, $t_{PD}$, and $f_{MAX}$ are found in the Compilation Report, as discussed in "Design Analysis" on page 6–6.

When you are analyzing a design to improve its performance, be sure to consider the two major contributors to long delay paths:

- Excessive levels of logic
- Excessive loading (high fan-out)

For MAX 7000 and MAX 3000 devices, when a signal drives out to more than one LAB, the programmable interconnect array (PIA) delay increases by 0.1 ns per additional LAB fan-out. Therefore, to minimize the added delay, you should concentrate the destination macrocells into fewer LABs, minimizing the number of LABs that are driven. The main cause of long delays in circuit design is excessive levels of logic.

## Improving Setup Time

Sometimes the $t_{SU}$ timing reported by the Quartus II Fitter may not meet your timing requirements. To improve the $t_{SU}$ timing, refer to the guidelines listed below:

- Turn on the **Fast Input Register** option—The **Fast Input Register** option allows input pins to directly drive macrocell registers via the fast-input path, thus minimizing the pin-to-register delay. This option is helpful when a pin drives a D-flip-flop without combinational logic between the pin and the register.

- Reduce the amount of logic between the input and the register—Excessive logic between the input pin and register will cause more delays. Therefore, to improve setup time, Altera recommends that you reduce the amount of logic between the input pin and the register whenever possible.
- Reduce fan-out—The delay from input pins to macrocell registers increases when the fan-out of the pins increases. Therefore, to improve the setup time, minimize the fan-out.

## Improving Clock-to-Output Time

To improve a design's clock-to-output time, you should minimize the register-to-output-pin delay. To improve the $t_{CO}$ timing, refer to the guidelines listed below:

- Use the global clock—Besides minimizing the delay from the register to output pin, minimizing the delay from the clock pin to the register can also improve the $t_{CO}$ timing. Altera recommends that you always use the global clock for low-skew and speed-critical signals.
- Reduce the amount of logic between the register and output pin—Excessive logic between the register and the output pin will cause more delay. Always minimize the amount of logic between the register and output pin for faster clock-to-output time.

Table 6–10 lists timing results for an EPM7064AETC100-4 device when a combination of the **Fast Input Register** option, global clock, and minimal logic is used. When the **Fast Input Register** option is turned on, the $t_{SU}$ timing is improved ($t_{SU}$ decreases from 1.6 ns to 1.3 ns and from 2.8 ns to 2.5 ns). The $t_{CO}$ timing is improved when the global clock is used for low-skew and speed-critical signals ($t_{CO}$ decreases from 4.3 ns to 3.1 ns). However, if there is additional logic used between the input pin and the register or the register and the output pin, the $t_{SU}$ and $t_{CO}$ timing will increase.

*Table 6–10. EPM7064AETC100-4 Device Timing Results (Part 1 of 2)*

| Number of Registers | $t_{SU}$ | $t_H$ | $t_{CO}$ | Global Clock Used | Fast Input Register Option | D Input Location | Q Output Location | Additional Logic Between D Input Location & Register | Additional Logic Between Register & Q Output Location |
|---|---|---|---|---|---|---|---|---|---|
| One | 1.3 ns | 1.2 ns | 4.3 ns | No | On | LAB A | LAB A | No | No |
| One | 1.6 ns | 0.3 ns | 4.3 ns | No | Off | LAB A | LAB A | No | No |
| One | 2.5 ns | 0 ns | 3.1 ns | Yes | On | LAB A | LAB A | No | No |

*Table 6–10. EPM7064AETC100-4 Device Timing Results (Part 2 of 2)*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| One | 2.8 ns | 0 ns | 3.1 ns | Yes | Off | LAB A | LAB A | No | No |
| One | 3.6 ns | 0 ns | 3.1 ns | Yes | Off | LAB A | LAB A | Yes | No |
| One | 2.8 ns | 0 ns | 7.0 ns | Yes | Off | LAB D | LAB A | No | Yes |
| 16 registers with the same D and clock inputs | 2.8 ns | 0 ns | All 6.2 ns | Yes | Off | LAB D | LAB A, B | No | No |
| 32 registers with the same D and clock inputs | 2.8 ns | 0 ns | All 6.4 ns | Yes | Off | LAB C | LAB A, B, C | No | No |

## Improving Propagation Delay ($t_{PD}$)

Achieving fast propagation delay ($t_{PD}$) timing is required in many system designs. However, if there are long delay paths through complex logic, achieving fast propagation delays can be difficult. To improve your design's $t_{PD}$, Altera recommends that you follow the guidelines discussed in this section.

- Turn on **Auto Parallel Expanders** in the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu)—Turning on the parallel expanders for individual nodes or subdesigns can increase the performance of complex logic functions. However, if the project's pin or logic cell assignments use parallel expanders placed physically together with macrocells (which can reduce routability), parallel expanders can cause the Quartus II Fitter to have difficulties finding and optimizing a fit. Additionally, the number of macrocells required to implement the design will also increase and result in a no fit error during compilation if the device's resources are limited. For more information on turning the **Auto Parallel Expanders** option on, refer to "Resolving Macrocell Usage Issues" on page 6–46.
- Set the **Optimization Technique** to **Speed**—By default, the Quartus II software sets the **Optimization Technique** option to **Speed** for MAX 7000 and MAX 3000 devices. Thus, you should only need to reset the **Optimization Technique** option back to **Speed** if you have previously set it to **Area**. You can reset the **Optimization Technique** option in the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu).

## Improving Maximum Frequency ($f_{MAX}$)

Maintaining the system clock at or above a certain frequency is a major goal in circuit design. For example, if you have a fully synchronous system that must run at 100 MHz, the longest delay path from the output of any register to the input(s) of the register(s) it feeds must be less than 10 ns. Maintaining the system clock speed can be difficult if there are long delay paths through complex logic. Altera recommends that you follow the guidelines below to improve your design's clock speed (i.e., $f_{MAX}$).

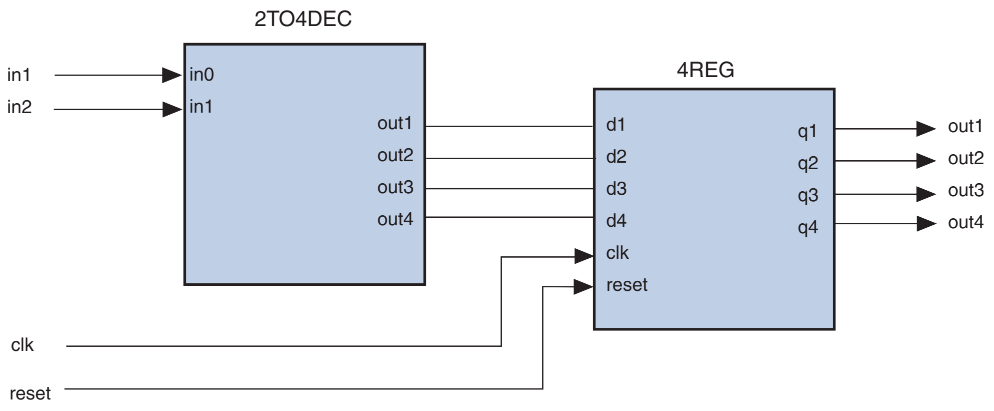
- Turn on **Auto Parallel Expanders** in the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu)—Turning on the parallel expanders for individual nodes or subdesigns can increase the performance of complex logic functions. However, if the project's pin or logic cell assignments use parallel expanders placed physically together with macrocells (which can reduce routability), parallel expanders can cause the Quartus II Compiler to have difficulties finding and optimizing a fit. Additionally, the amount of macrocells required to implement the design will also increase and result in a no fit error during compilation if the device's resources are limited. For more information on turning the **Auto Parallel Expanders** option on, refer to "Resolving Macrocell Usage Issues" on page 6–46.
- Use global signals/dedicated inputs—Altera MAX 7000 and MAX 3000 devices' dedicated inputs provide low skew and high speed for high fan-out signals. Thus, Altera recommends that you always minimize the number of control signals in the design and use the dedicated inputs to implement them.
- Set the **Optimization Technique** to **Speed**—By default, the Quartus II software sets the **Optimization Technique** option to **Speed** for MAX 7000 and MAX 3000 devices. Thus, you should only need to reset the **Optimization Technique** option back to **Speed** if you have previously set it to **Area**. You can reset the **Optimization Technique** option in the **Analysis & Synthesis Settings** page of the **Settings** dialog box (Assignments menu).
- Pipeline the design—Pipelining, which increases clock frequency ($f_{MAX}$), refers to dividing large blocks of combinational logic by inserting registers. For more information on pipelining, see "Optimizing Source Code—Pipelining for Complex Register Logic".

## Optimizing Source Code—Pipelining for Complex Register Logic

If the methods described in the preceding sections do not sufficiently improve your results, modify the design at the source to achieve the desired results. Using a pipelining technique can consume device resources, but it also lowers the propagation delay between registers, allowing you to maintain high system clock speed.

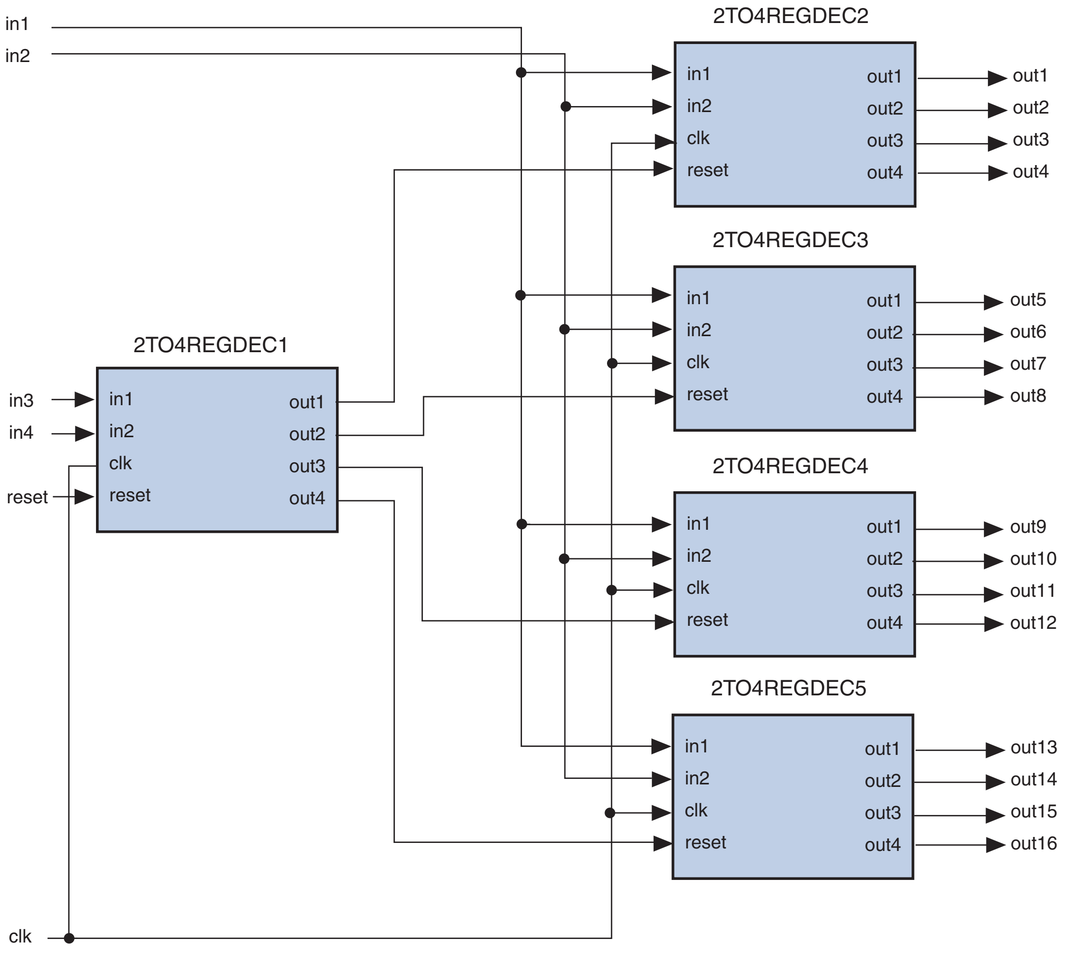
The benefits of pipelining can be demonstrated with a 4- to 16-pipelined decoder that decodes the 4-bit numbers. The decoder is based on five 2- to 4-pipelined decoders with outputs that are registered using D-flip-flops. Figures 6–20 shows one of the 2- to 4-pipelined decoders. The function `2TO4DEC` is the 2- to 4-decoder that feeds all four decoded outputs (i.e., `out1`, `out2`, `out3`, and `out4`) to the D-flip-flops in `4REG`.

*Figure 6–20. A 2- to 4-Pipelined Decoder*



Figures 6–21 shows five 2- to 4-decoders (`2TO4REGDEC`) that are combined to form a 4- to 16-pipelined decoder. The first decoder (`2TO4REGDEC1`) will decode the two most significant bits (MSB) (i.e., `in3` and `in4`) of the 4- to 16-decoder. The decoded output from the `2TO4REGDEC1` decoder will only enable one of the rest of the 2- to 4-decoders (i.e., `2TO4REGDEC2`, `2TO4REGDEC3`, `2TO4REGDEC4`, or `2TO4REGDEC5`). The inputs `in1` and `in2` are decoded by the enabled 2- to 4-decoder. Because the time to generate the decoded output increases with the size of the decoder, pipelining the design reduces the time consumed to generate the decoded output, thus improving the maximum frequency. In Figures 6–21, the MSBs (i.e.,`in3` and `in4`) are decoded in the first clock cycle, while the other bits (i.e., `in1`, and `in2`) are decoded in the following clock cycle.

*Figure 6–21. Five 2- to 4-Pipelined Decoders Combined to Form a 4- to 16-Pipelined Decoder*



## Compilation Time Optimization Techniques

If optimizing the compilation time of your design is important, use the techniques in this section. Be aware that reducing compilation time using these techniques may reduce the overall quality of results.

### Reducing Synthesis and Synthesis Netlist Optimization Time

You can use Quartus II integrated synthesis to synthesize and optimize HDL designs. You can also use synthesis netlist optimizations to optimize netlists synthesized by third-party EDA software. Using these optimizations can make the Analysis & Synthesis module take much longer to run. Look at the Analysis & Synthesis messages to find out how much time these optimizations take. Note that the compilation time spent in Analysis & Synthesis is typically small compared to the compilation time spent in the Fitter.

If your design meets your performance requirements without synthesis netlist optimizations, turn the optimizations off to save time. If you need to turn on synthesis netlist optimizations to meet performance, separately optimize parts of your design hierarchy to reduce analysis and synthesis. Create ATOM netlists for parts of your design you have already synthesized and optimized. The Quartus II Analysis & Synthesis module will not need to reoptimize those netlists, resulting in reduced synthesis and netlist optimization time.

For more information on creating hierarchical designs with multiple netlists, refer to the *Hierarchical Block-Based & Team-Based Design Flows* chapter in Volume 1 of the *Quartus II Handbook*.

## Reducing Placement Time

The time needed to place a design depends on two factors:

- The number of ways the logic in the design can be placed in the device
- The settings that control how hard the placer works to find a good placement

You can reduce the placement time in two ways: change the settings for the placement algorithm, or use LogicLock regions to manually control where parts of the design are placed. Sometimes there is a trade-off between placement time and routing time. Routing time can increase if the placer does not run long enough to find a good placement. When you reduce placement time, make sure that it does not increase routing time and cancel out the time reduction.

### *Fitter Effort Setting*

Use the **Fitter effort** setting on the **Fitter Settings** page of the **Settings** dialog box (Assignments menu) to shorten run time by changing the effort level to **Auto Fit** or **Fast Fit**.

### *Physical Synthesis Effort Settings*

You can use the physical synthesis options to optimize your post-synthesis netlist and improve your timing performance. These options, which affect placement, can significantly increase compilation time. Refer to Table 6–7 on page 6–29 for detailed results.

If your design meets your performance requirements without physical synthesis options, turn them off to save time. You can also use the **Physical synthesis effort** setting on the **Physical Synthesis Optimizations** page under **Fitter Settings** in the **Settings** dialog box (Assignments menu) to reduce the amount of extra compilation time used by these optimizations. The **Fast** setting directs the Quartus II software to use a lower level of physical synthesis optimization that, compared to the normal level, may cause a smaller increase in compilation time. However, the lower level of optimization may result in a smaller increase in design performance.

### *Incremental Fitting*

Incremental fitting can reduce placement time after an initial compilation because the placer tries to place unchanged nodes in your design in their previous locations. The matching is based on the nodes' logic and connectivity, not just their names. Even if all of the combinational node names have changed, incremental fitting should be able to match the original nodes' functionality and recreate the same placement. Not all nodes need to match, making this mode perfect for Engineering Change Orders (ECOs). Incremental fitting can start an entirely new placement under some conditions:

- More than 500 nodes in the design do not match
- Performance drops by more than 5%
- You significantly change LogicLock regions
- You target a new device
- You delete the design database

Start incremental fitting by choosing **Start > Start Incremental Fitting** (Processing menu).

### *LogicLock Regions*

Preserving information about previous placements can make future placements take less time. To successfully preserve information, node names must not change from placement to placement, and node locations must be preserved so they will not change from placement to placement.

To preserve node names, you must use atom netlists. Atom netlists include Verilog Quartus Mapping (**.vqm**) files and EDIF files, which are the outputs of third-party synthesis software. If you use Quartus II integrated synthesis, or turn on any Quartus II netlist optimizations, you must generate VQM files and turn off netlist optimizations in future compilations.

To preserve node locations, use back-annotated LogicLock regions. After you back-annotate a LogicLock region, the node locations are fixed and the placer skips those nodes, saving time. If you change part of your design in a back-annotated LogicLock region, delete the back-annotated contents of the region and recompile the design. The placer will find a new placement for the changed logic and any logic that is not in a LogicLock region.

Follow these steps to reduce placement time with atom netlists and LogicLock regions:

1. Choose hierarchies in your design to assign to LogicLock regions. You do not have to use LogicLock regions for all hierarchies in your design, just the hierarchies for which you want to reduce placement time.

2. Create separate atom netlists for the chosen hierarchies and assign them to LogicLock regions

3. Turn off netlist optimizations on each LogicLock region

4. Compile the design

5. Back-annotate the LogicLock regions

Follow these steps when you change logic in a back-annotated LogicLock region

1. Create a new atom netlist for the hierarchy

2. Delete the back-annotated contents of the appropriate LogicLock region

3. Recompile the design

4. Back-annotate the LogicLock region

For more information on creating hierarchical designs with multiple netlists, refer to the *Hierarchical Block-Based & Team-Based Design Flows* chapter in Volume 1 of the *Quartus II Handbook*.

### Reducing Routing Time

The time needed to route a design depends on three factors: the device architecture, the placement of the design in the device, and the connectivity between different parts of the design. Typically the routing time is not a significant amount of the compilation time. If your design takes a long time to route, perform one or more of the following actions:

- Check for routing congestion
- Let the placer run longer to find a more routable placement
- Use LogicLock regions to preserve routing information

#### *Routing Congestion*

To identify congested routing areas in your design, open the Timing Closure Floorplan. Choose **Timing Closure Floorplan** (Assignments menu) and turn on **Show Routing Congestion**. A routing resource usage above 90% indicates routing congestion.

If the area with routing congestion is in a LogicLock region or between LogicLock regions, remove the LogicLock regions and recompile the design. If the routing time remains the same, then the time is a characteristic of the design and the placement. If the routing time decreases, you should consider changing the size, location, or contents of the LogicLock regions to reduce congestion and decrease routing time.

#### *LogicLock Regions*

You can use LogicLock regions back-annotated to the routing level to preserve routing information between compilations. This can reduce the time required to route a design. Follow the same steps as for using LogicLock regions to reduce placement time, but back-annotate to the routing level.

## Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run some of these procedures at a command prompt.

For detailed information about specific scripting command options and Tcl API packages, type `quartus_sh --qhelp` at a system command prompt to run the Quartus II Command-Line and Tcl API Help browser.

For more information on Quartus II scripting support, including examples, refer to the *Tcl Scripting* and *Command-Line Scripting* chapters in Volume 2 of the *Quartus II Handbook*.

You can specify many of the options described in this section either in an instance, or at a global level, or both.

Use the following Tcl command to make a global assignment:

`set_global_assignment -name` *<QSF Variable Name> <Value>*

Use the following Tcl command to make an instance assignment:

`set_instance_assignment -name` *<QSF Variable Name> <Value>* \
`-to` *<Instance Name>*

### Initial Compilation Settings

Table 6–11 lists the QSF variable name and applicable values for the settings discussed in "Initial Compilation" on page 6–2. The QSF variable name is used in the Tcl assignment to make the setting along with the appropriate value. The Type column indicates whether the setting is supported as a Global setting, an Instance setting, or both.

*Table 6–11. Initial Compilation Settings*

| Setting Name | QSF Variable Name | Values | Type |
|---|---|---|---|
| Use Smart Compilation | SPEED_DISK_USAGE_TRADEOFF | SMART, NORMAL | Global |
| Optimize Timing | OPTIMIZE_TIMING | OFF, "NORMAL COMPLIATION", "EXTRA EFFORT" | Global |
| Optimize I/O Cell Register Placement | OPTIMIZE_IOC_REGISTER_PLACEMENT_FOR_TIMING | ON,OFF | Global |
| Optimize Hold Timing | OPTIMIZE_HOLD_TIMING | OFF, "IO PATHS AND MINIMUM TPD PATHS", "ALL PATHS" | Global |
| Fitter Effort | FITTER_EFFORT | "STANDARD FIT", "FAST FIT", "AUTO FIT" | Global |

### Resource Utilization Optimization Techniques (LUT-Based Devices)

Table 6–12 lists the QSF variable name and applicable values for the settings discussed in "Resource Utilization Optimization Techniques (LUT-Based Devices)" on page 6–13. The QSF variable name is used in the

Tcl assignment to make the setting along with the appropriate value. The Type column indicates whether the setting is supported as a Global setting, an Instance setting, or both.

*Table 6–12. Resource Utilization Optimization Settings*

| Setting Name | QSF Variable Name | Values | Type |
|---|---|---|---|
| Auto Packed Registers | AUTO_PACKED_REGISTERS _<*Device Family Name*> | OFF, NORMAL, "MINIMIZE AREA" | Global, Instance |
| Auto Packed Registers | AUTO_PACKED_REGISTERS _<CYCLONE\|MAXII\|STRATIX\|STRATIXII> | OFF, NORMAL, "MINIMIZE AREA", "MINIMIZE AREA WITH CHAINS", AUTO | Global, Instance |
| Perform WYSIWYG Primitive Resynthesis | ADV_NETLIST_OPT_SYNTH_WYSIWYG_ REMAP | ON, OFF | Global, Instance |
| Optimization Technique | <*Device Family Name*>_OPTIMIZATION_ TECHNIQUE | AREA, SPEED, BALANCED | Global, Instance |
| State Machine Encoding | STATE_MACHINE_PROCESSING | AUTO, "ONE-HOT", "MINIMAL BITS", "USER-ENCODED" | Global, Instance |
| Preserve Hierarchy | PRESERVE_HIERARCHICAL_BOUNDARY | OFF, RELAXED, FIRM, | Instance |
| Auto RAM Replacement | AUTO_RAM_RECOGNITION | ON, OFF | Global, Instance |
| Auto ROM Replacement | AUTO_ROM_RECOGNITION | ON, OFF | Global, Instance |
| Auto Shift Register Replacement | AUTO_SHIFT_REGISTER_RECOGNITION | ON, OFF | Global, Instance |
| Auto DSP Block Replacement | AUTO_DSP_RECOGNITION | ON, OFF | Global, Instance |

## I/O Timing Optimization Techniques (LUT-Based Devices)

Table 6–13 lists the QSF variable name and applicable values for the settings discussed in "I/O Timing Optimization Techniques (LUT-Based Devices)" on page 6–21. The QSF variable name is used in the Tcl assignment to make the setting along with the appropriate value. The Type column indicates whether the setting is supported as a Global setting, an Instance setting, or both.

*Table 6–13. I/O Timing Optimization Settings*

| Setting Name | QSF Variable Name | Values | Type |
|---|---|---|---|
| Optimize I/O cell register placement for timing | OPTIMIZE_IOC_REGISTER_PLACEMENT_FOR_TIMING | ON, OFF | Global |
| Fast Input Register | FAST_INPUT_REGISTER | ON, OFF | Instance |
| Fast Output Register | FAST_OUTPUT_REGISTER | ON, OFF | Instance |
| Fast Output Enable Register | FAST_OUTPUT_ENABLE_REGISTER | ON, OFF | Instance |

## $F_{MAX}$ Timing Optimization Techniques (LUT-Based Devices)

Table 6–14 lists the QSF variable name and applicable values for the settings discussed in "$f_{MAX}$ Timing Optimization Techniques (LUT-Based Devices)" on page 6–27. The QSF variable name is used in the Tcl assignment to make the setting along with the appropriate value. The Type column indicates whether the setting is supported as a Global setting, an Instance setting, or both.

*Table 6–14. $F_{MAX}$ Timing Optimization Settings (Part 1 of 2)*

| Setting Name | QSF Variable Name | Values | Type |
|---|---|---|---|
| Perform WYSIWYG Primitive Resynthesis | ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP | ON, OFF | Global, Instance |
| Perform Gate Level Register Retiming | ADV_NETLIST_OPT_SYNTH_GATE_RETIME | ON, OFF | Global |
| Allow Register Retiming to trade off Tsu/Tco with $f_{MAX}$ | ADV_NETLIST_OPT_RETIME_CORE_AND_IO | ON, OFF | Global |
| Perform Physical Synthesis for Combinational Logic | PHYSICAL_SYNTHESIS_COMBO_LOGIC | ON, OFF | Global |

*Table 6–14. $F_{MAX}$ Timing Optimization Settings (Part 2 of 2)*

| Setting Name | QSF Variable Name | Values | Type |
|---|---|---|---|
| Perform Register Duplication | PHYSICAL_SYNTHESIS_REGISTER_ DUPLICATION | ON, OFF | Global |
| Perform Register Retiming | PHYSICAL_SYNTHESIS_REGISTER_RETIMING | ON, OFF | Global |
| Physical Synthesis Effort | PHYSICAL_SYNTHESIS_EFFORT | NORMAL, EXTRA, FAST | Global |
| Seed | SEED | *<integer>* | Global |
| Maximum Fan-Out | MAX_FANOUT | *<integer>* | Instance |
| Manual Logic Duplication | DUPLICATE_ATOM | *<node name>* | Instance |

## Conclusion

Today's complex designs have complex requirements. Methodologies for fitting your design and for achieving timing closure are fundamental to optimal performance in today's designs. Using the Quartus II design optimization methodology closes timing quickly on complex designs, reduces iterations by providing more intelligent and better linkage between analysis and assignment tools, and balances multiple design constraints including multiple clocks, routing resources, and area constraints.

The Quartus II software provides many features to effectively achieve optimal results. Follow the techniques presented in this chapter to efficiently optimize a design for area or timing performance or to reduce compilation time.

# 7. Timing Closure Floorplan

qii52006 2.0

## Introduction

With FPGA designs surpassing the million-gate mark, designers need advanced tools to better analyze timing closure issues to achieve their system performance goals.

The Altera® Quartus® II software offers many advanced design analysis tools that allow detailed timing analysis of your designs, including a fully integrated Timing Closure Floorplan Editor. With these tools and options, the critical paths in your design can be easily determined and located in the floorplan of the targeted device. This chapter explains how to use these tools and options to enhance your FPGA design analysis.

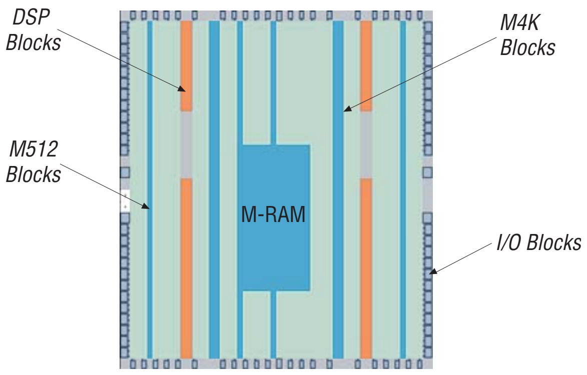
## Design Analysis Using the Timing Closure Floorplan

The Timing Closure Floorplan Editor assists you in visually analyzing your designs before and after performing a full design compilation in the Quartus II software. This floorplan editor, used in conjunction with traditional Quartus II timing analysis features, provides a powerful method to perform design analysis.

### Timing Closure Floorplan Views

The Timing Closure Floorplan Editor allows you to customize the views of your design. The Field View is a color-coded, high-level view of resources. Figure 7–1 shows the Field View of a Stratix® device.

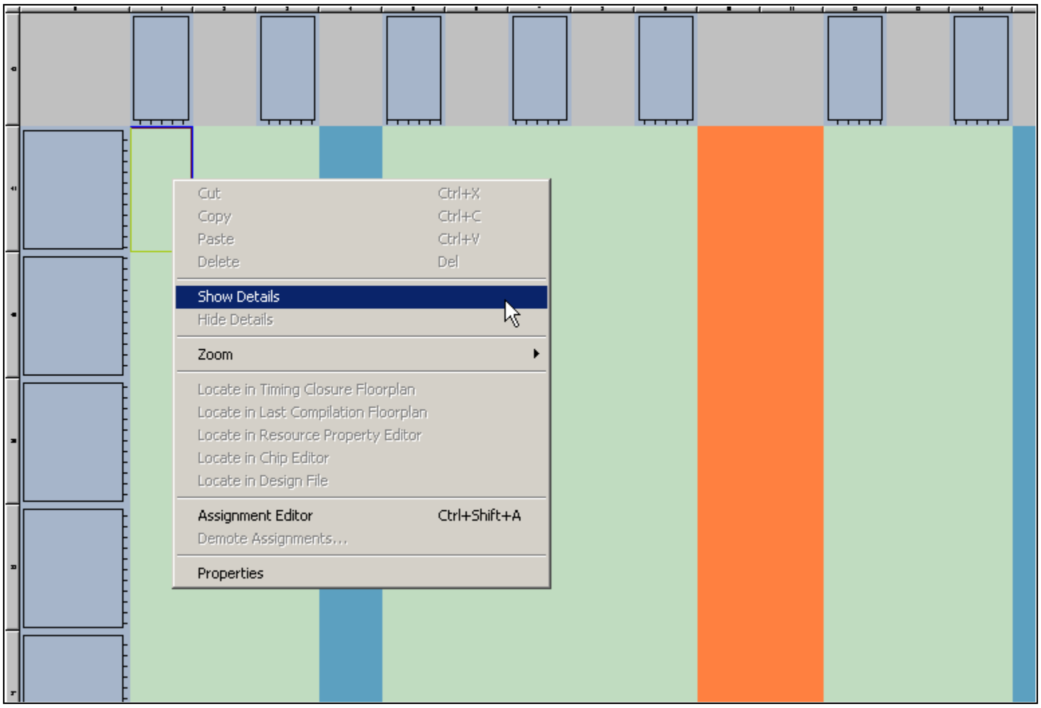
*Figure 7–1. Field View of a Stratix Device*



In the field view, you can view the details of a resource by selecting the **resource,** right-clicking, then selecting **Show Details** from the right-button pop-up menu. To hide the details, select all the resources, right-click, and select **Hide Details**. See Figure 7–2.

You can also view your design in the Timing Closure Floorplan Editor with the traditional Interior Cells, Package Top, and Package Bottom views. Use the View menu to change to the various floorplan views.

*Figure 7–2. Show Details & Hide Details of a LAB in Field View*

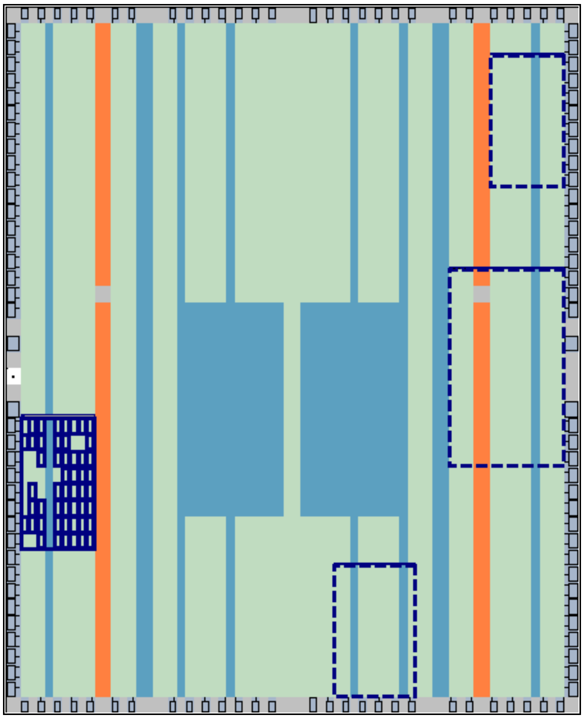


## Viewing Assignments

The Timing Closure Floorplan Editor differentiates between user assignments and fitter placements. User assignments are location and LogicLock™ assignments that you make. Fitter placements are the locations where the Quartus II software placed all nodes after the last compilation. You can view both user assignments and fitter placements at the same time.

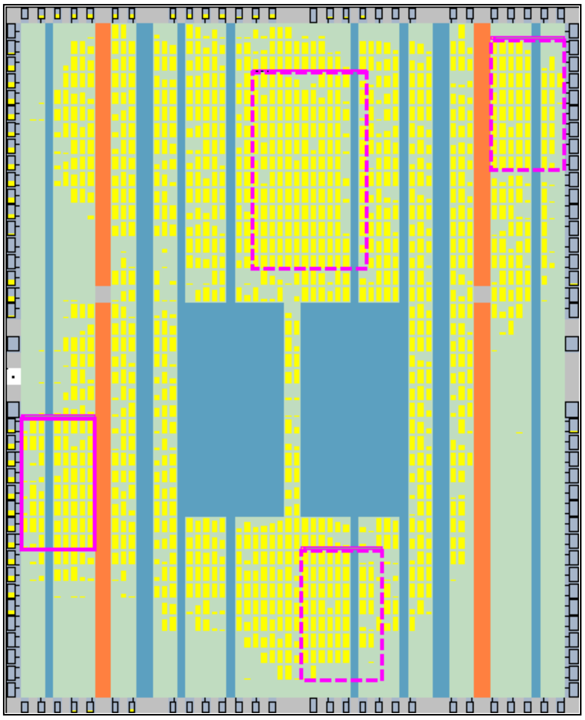
To see user assignments, click the **User Assignments** icon in the **Floorplan Editor** toolbar, or choose **Assignments** (View menu) and select **Show User Assignments**. See Figure 7–3.

*Figure 7–3. User Assignments*



To see fitter placements, click the **Fitter Assignments** icon in the F**loorplan Editor** toolbar, or choose **Assignments** (View menu) and select **Show Fitter Placements**. See Figure 7–4.

*Figure 7–4. Fitter Placements*



## Viewing Critical Paths

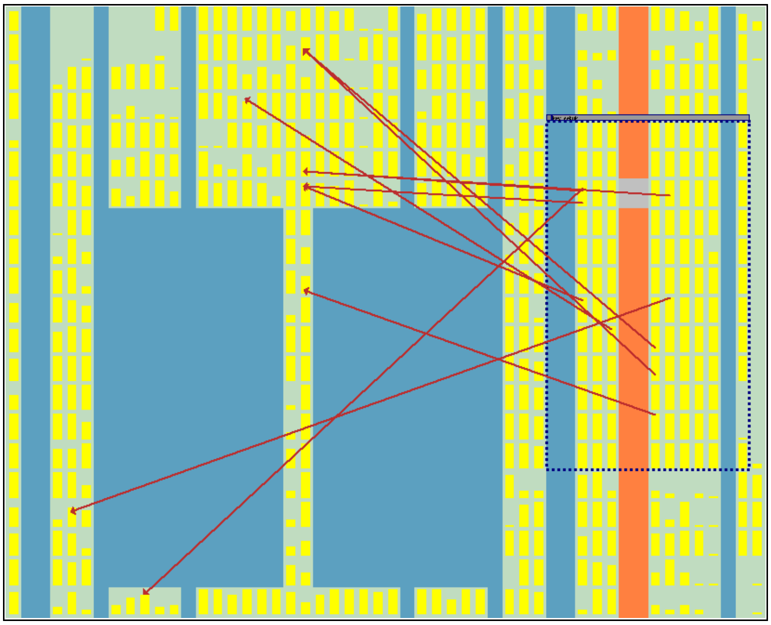
The View Critical Paths feature displays routing paths in the floorplan and ranks their importance, as shown in Figure 7–5. The criticality of a path is determined by either delay or slack. You can view a percentage of critical paths or specify how many paths you wish to see. You can also choose to see paths for all clock domains or a specific clock domain. The following paths can be displayed:

- $t_{PD}$ - The time required for a signal from an input pin to propagate through combinational logic and appear at an external output pin.
- $t_{SU}$ - The length of time for which data that feeds a register via its data or enable input(s) must be present at an input pin before the clock signal that clocks the register is asserted at the clock pin.
- $t_{CO}$ - The maximum time required to obtain a valid output at an output pin that is fed by a register after a clock signal transition on an input pin that clocks the register. This time always represents an external pin-to-pin delay.

- $t_H$ - The minimum length of time for which data that feeds a register through data or enable input(s) must be retained at an input pin after the clock signal that clocks the register is asserted at the clock pin.
- Register-to-Register ( $f_{MAX}$) - The maximum clock frequency that can be achieved without violating internal setup ($t_{SU}$) and hold ($t_H$) time requirements.

To view critical paths in the floorplan, click the **Show Critical Paths** icon or chose **Routing** > **Show Critical Paths** (View menu). To set the criteria for the critical path you want to view, select the **Critical Paths Settings** icon or choose **Routing** > **Critical Paths Settings** (View menu). See Figure 7–5.

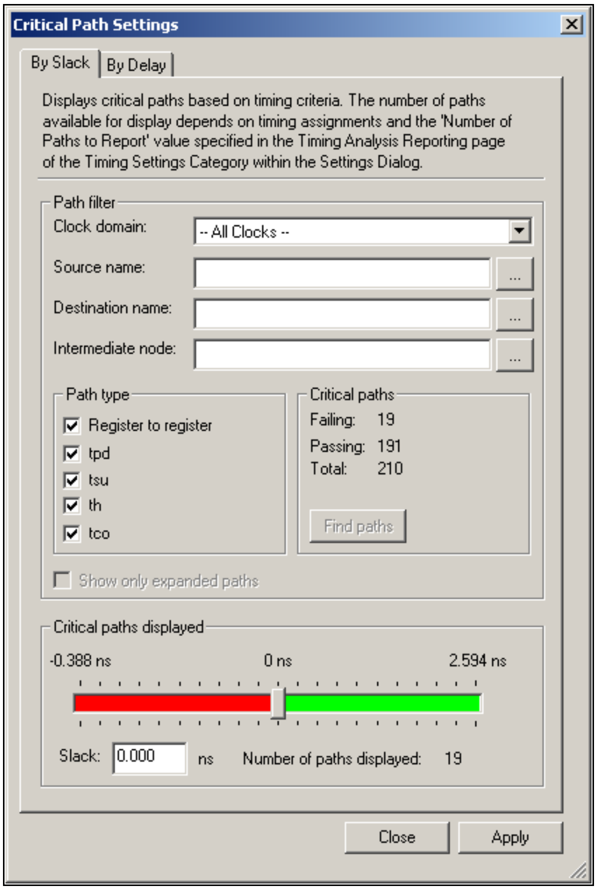
*Figure 7–5. Critical Paths*



When viewing critical paths by slack, the settings are specified with the **By Slack** tab of the **Critical Path Settings** dialog box shown in Figure 7–6. You determine which path to view and specify the slack threshold beyond which you would like the path displayed in the floorplan. For example, you can view all paths with a slack of -1 ns or worse.

☞ Timing settings must be made for paths to be displayed in the floorplan.

*Figure 7–6. Critical Paths Settings, by Slack*



Critical Path Settings

By Slack | By Delay

Displays critical paths based on timing criteria. The number of paths available for display depends on timing assignments and the 'Number of Paths to Report' value specified in the Timing Analysis Reporting page of the Timing Settings Category within the Settings Dialog.

Path filter

Clock domain: -- All Clocks --

Source name:

Destination name:

Intermediate node:

Path type

[✓] Register to register

[✓] tpd

[✓] tsu

[✓] th

[✓] tco

Critical paths

| | |
|---|---|
| Failing: | 19 |
| Passing: | 191 |
| Total: | 210 |

Find paths

[ ] Show only expanded paths

Critical paths displayed

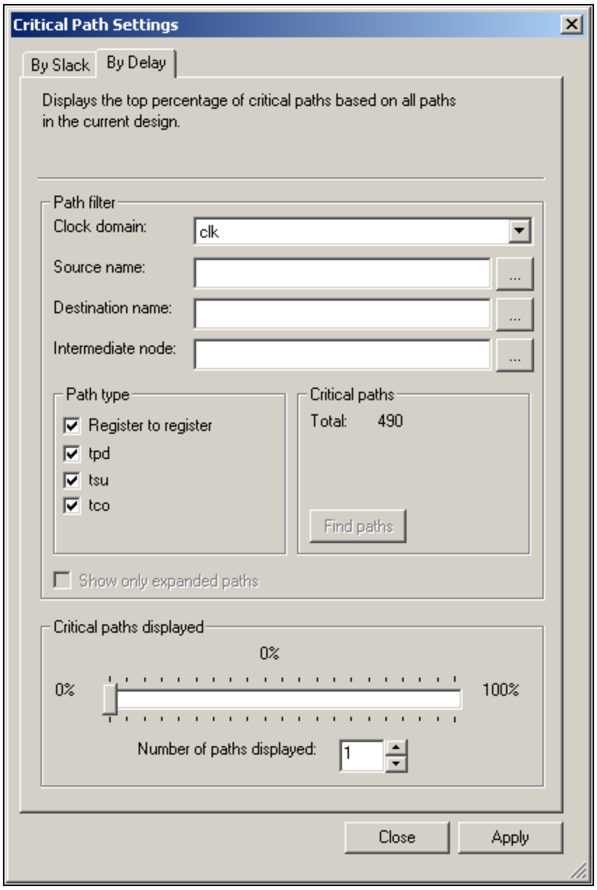
-0.388 ns   0 ns   2.594 ns

Slack: 0.000 ns   Number of paths displayed: 19

Close   Apply

When viewing critical paths by delay, the settings are specified with the **By Delay** tab of the **Critical Path Settings** dialog box shown in Figure 7–7. This view displays the critical paths with the longest delay.

*Figure 7–7. Critical Paths Settings, by Delay*


Critical Path Settings

By Slack | By Delay

Displays the top percentage of critical paths based on all paths in the current design.

Path filter
Clock domain: clk
Source name:
Destination name:
Intermediate node:

Path type
- [✓] Register to register
- [✓] tpd
- [✓] tsu
- [✓] tco

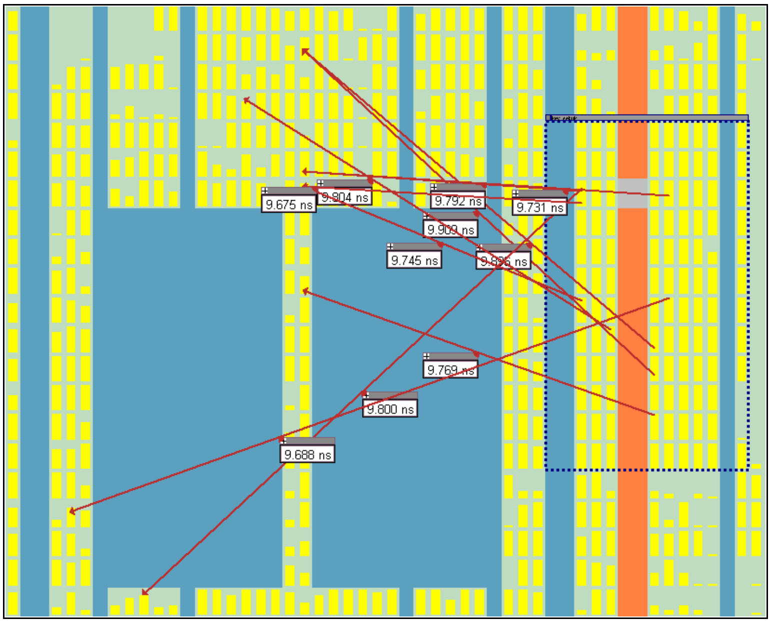
Critical paths
Total: 490

- [ ] Show only expanded paths

Critical paths displayed
0%
Number of paths displayed: 1

The critical path feature is extremely useful in determining the criticality of nodes based on placement. There are a number of options to view the details of critical path. To see the delay of the critical path, click the **Show Routing Delays** icon or choose **Routing** > **Show Routing Delays** (View menu). See Figure 7–8.

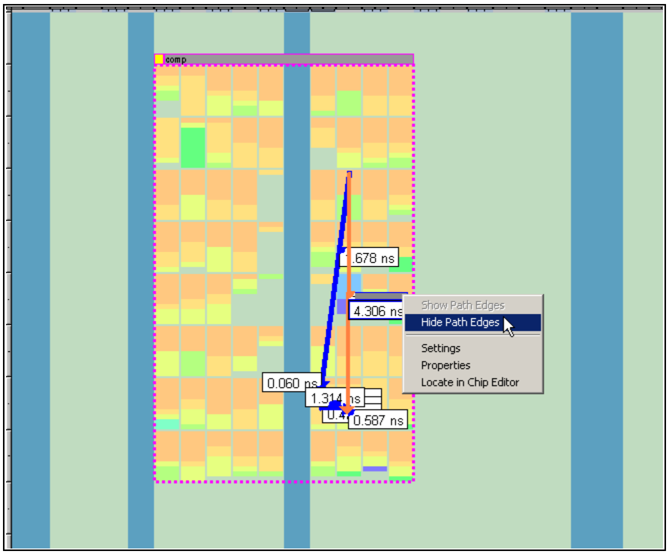
*Figure 7–8. Routing Delays for Critical Paths*



The default view shows the path. You can also view all the combinational nodes to see the worst-case path between the source and destination nodes. To view the full path, select the path by clicking on the delay label, right click, and select **Show Path Edges**. Figure 7–9 shows a critical path through combinational nodes. To hide the combinational nodes, select the path, right click, and select **Hide Path Edges**.

The routing delays must be shown in order to be able to select a path.

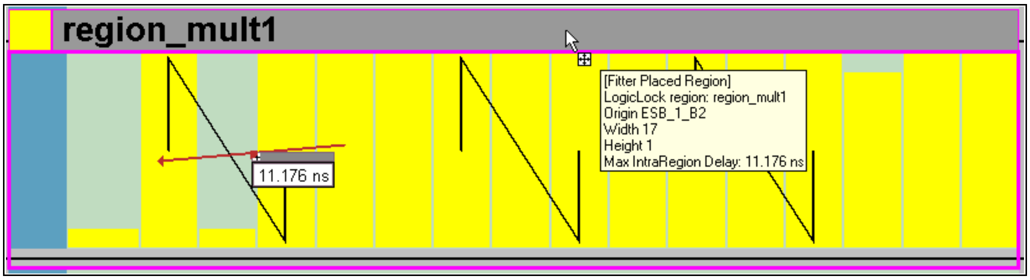
*Figure 7–9. Worst-Case Combinational Paths of Critical Paths*



You can also assign the path to a LogicLock region in the **Paths** dialog box; select the path, right click, and select **Properties**.

You can determine the maximum routing delay between two nodes within a LogicLock region. To use this feature, click the **Show Intra-region Delay** icon or go to **Routing**> **Show Intra-region Delay** (View menu). Place your cursor over a fitter-placed LogicLock region to see the maximum delay. Figure 7–10 shows the maximum routing delay of a LogicLock region.

*Figure 7–10. Maximum Intra-Region Delay*



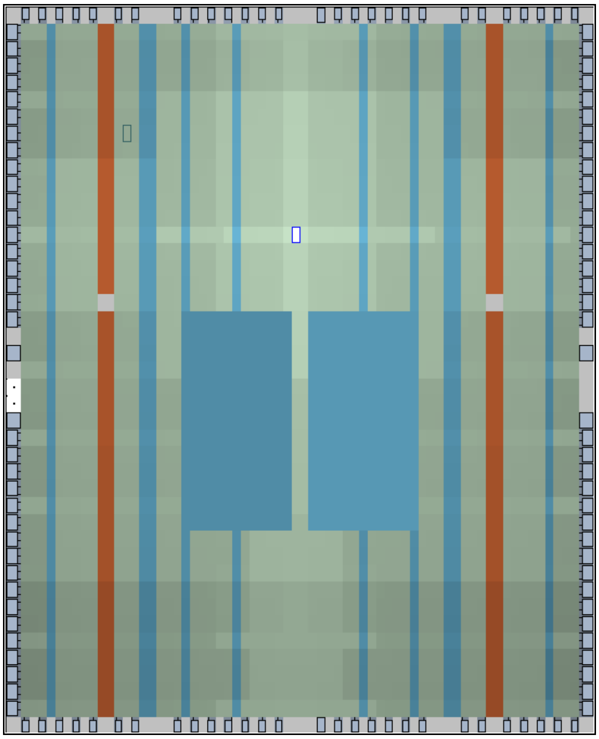
[Fitter Placed Region]
LogicLock region: region_mult1
Origin ESB_1_B2
Width 17
Height 1
Max IntraRegion Delay: 11.176 ns

For more information on making path assignments with the **Paths** dialog box, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook.*

## Physical Timing Estimates

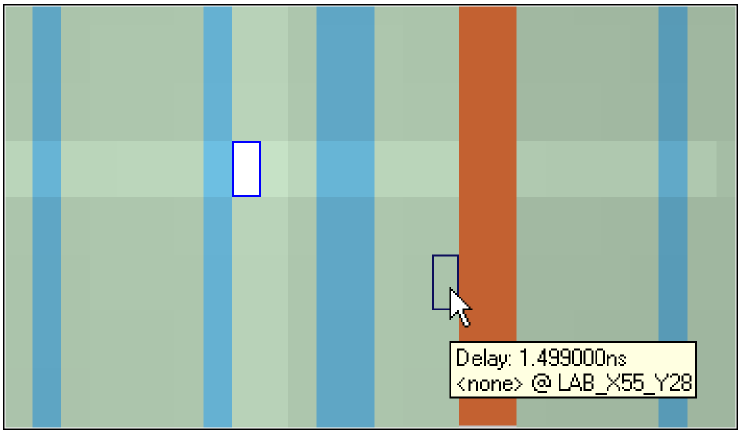
In the Timing Closure Floorplan Editor, you can select a resource and see the approximate delay to any other resource on the chip. Once a resource is selected, the delay is represented by the color of potential destination resources. The darker the resource, the longer the delay, as shown in Figure 7–11.

***Figure 7–11. Physical Timing Estimates for Large Floorplan***



You can also get an approximation of the delay between two points by selecting a source and holding your cursor over a potential destination resource, as shown in Figure 7–12.

*Figure 7–12. Delay for Physical Timing Estimate*



The delays represent an estimate based on probable best-case routing. It is possible the delay is greater than what is shown, depending on the availability of routing resources. In general, there is a strong correlation between the probable and actual delay.

To view the physical timing estimates, click the **Show Physical Timing Estimate** icon or choose **Routing** > **Show Physical Timing Estimates** (View menu).

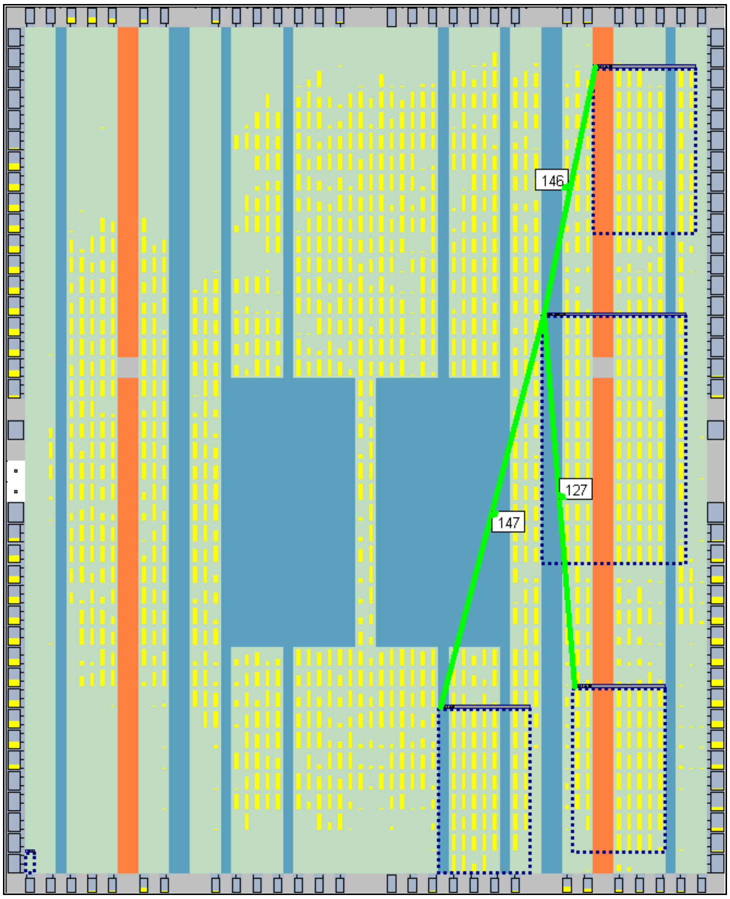
The physical timing estimate information can be used when manually placing logic in a device. This allows you to place critical nodes and modules closer together and non-critical or unrelated nodes and modules further apart. This reduces the routing congestion between critical and non-critical entities and modules allowing the Quartus II Fitter to select the timing requirements.

## LogicLock Region Connectivity

You can also see how logic in LogicLock regions interface by viewing the connectivity between assigned LogicLock regions. This capability is extremely valuable when entities are assigned to LogicLock regions. It is also possible to see the fan-in and fan-out of selected LogicLock regions.

Figure 7–13 shows standard LogicLock region connections. To view the connections in the timing closure floorplan, click the **Show LogicLock Regions Connectivity** icon in the toolbar or choose **Routing** > **Show LogicLock Regions Connectivity** (View menu).

*Figure 7–13. LogicLock Region Connections with Connection Count*

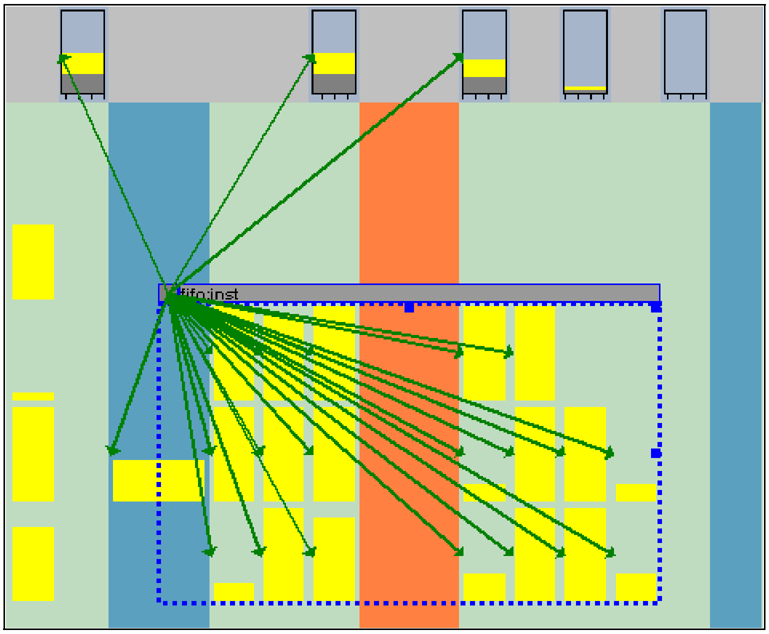


The connection line thickness indicates how many connections exist between regions. To view the number of connections between regions, click the **Show Connection Count** icon or choose **Routing** > **Show Connection Count** (View menu).

LogicLock region connectivity is applicable only when the user assignments are viewed in the floorplan. When floating LogicLock regions are used, the origin of the user-assigned region is not necessarily the same as the fitter-placed region. This allows you to unlock a region and then lock it down again at a later time. You can change the origin of your floating LogicLock regions to that of the last compilation origin in the **LogicLock Regions** window (Assignments Menu), or by selecting **Back-Annotate Origin and Lock** under **Location** in the **LogicLock Regions Properties** dialog box.

To see the fan-in or fan-out of a LogicLock region, select the user-assigned LogicLock region while the fan-in or the fan-out option is turned on. To set the fan-in option, click the **Show Node Fan-In** icon or choose **Routing** > **Show Node Fan-In** (View menu). To set the **fan-out** option, select the **Show Node Fan-Out** icon or choose **Routing** > **Show Node Fan-Out** (View menu). Only the nodes that have user assignments are seen when viewing fan-in or fan-out of LogicLock regions. Figure 7–14 shows the fan-out of a selected LogicLock region.

*Figure 7–14. Fan-In or Fan-Out*



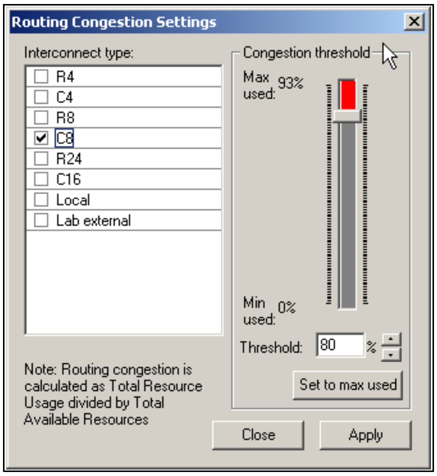
## Viewing Routing Congestion

The View Routing Congestion feature allows you to determine the percentage of routing resources used after a compilation. This feature identifies where there is a lack of routing resources.

The congestion is visually represented by the color and shading of logic resources. The darker shading represents a greater routing resource utilization. Logic resources that are red have routing resource utilization greater than the specified threshold.

To view routing congestion in the floorplan, click the **Show Routing Congestion** icon, or choose **Routing** > **Show Routing Congestion** (View menu). To set the criteria for the critical path you wish to view, click the **View Routing Congestion Settings** icon or choose **Routing** > **Routing Congestion Settings** (View menu). Figure 7–15 shows the **Routing Congestion Settings** dialog box.

*Figure 7–15. Routing Congestion Settings Window*


Routing Congestion Settings

Interconnect type:
- [ ] R4
- [ ] C4
- [ ] R8
- [x] C8
- [ ] R24
- [ ] C16
- [ ] Local
- [ ] Lab external

Congestion threshold

Max used: 93%

Min used: 0%

Threshold: 80 %

Note: Routing congestion is calculated as Total Resource Usage divided by Total Available Resources

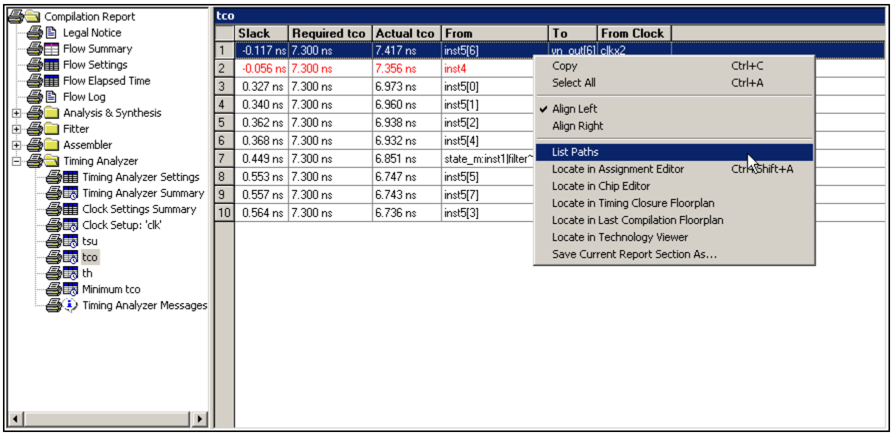
You can choose the routing resource you want to examine and set the congestion threshold. Routing congestion is calculated based on the total resource usage divided by the total available resources.

If you are using the routing congestion viewer to determine where there is a lack of routing resources, examine each routing resource individually to see which ones use close to 100% of available resources.

## I/O Timing Analysis Report File

Use the **Timing Analyzer** folder in the **Compilation Report** (Processing menu) to determine whether I/O timing has been met. The $t_{SU}$, $t_H$, and $t_{CO}$ reports list the I/O paths and the slack associated with each. The I/O paths that have not met the required timing are reported with a negative slack and are displayed in red as shown in Figure 7–16.

*Figure 7–16. I/O Requirements*

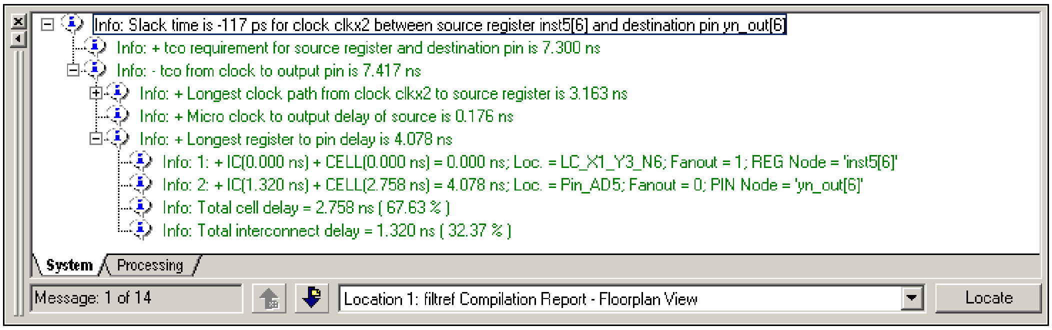


tco

| | Slack | Required tco | Actual tco | From | To | From Clock |
|---|---|---|---|---|---|---|
| 1 | -0.117 ns | 7.300 ns | 7.417 ns | inst5[6] | yn_out[6] | clkx2 |
| 2 | -0.056 ns | 7.300 ns | 7.356 ns | inst4 | | |
| 3 | 0.327 ns | 7.300 ns | 6.973 ns | inst5[0] | | |
| 4 | 0.340 ns | 7.300 ns | 6.960 ns | inst5[1] | | |
| 5 | 0.362 ns | 7.300 ns | 6.938 ns | inst5[2] | | |
| 6 | 0.368 ns | 7.300 ns | 6.932 ns | inst5[4] | | |
| 7 | 0.449 ns | 7.300 ns | 6.851 ns | state_m:inst1\|filter~ | | |
| 8 | 0.553 ns | 7.300 ns | 6.747 ns | inst5[5] | | |
| 9 | 0.557 ns | 7.300 ns | 6.743 ns | inst5[7] | | |
| 10 | 0.564 ns | 7.300 ns | 6.736 ns | inst5[3] | | |

To determine why timing requirements are not met, right-click a particular I/O entry and choose **List Paths**. A message appears in the **System** tab of the **Message** window. You can expand a selection by clicking the "**+**" icon at the beginning of the line, as shown in Figure 7–17. This is a good method of determining where along the path the greatest delay is located.

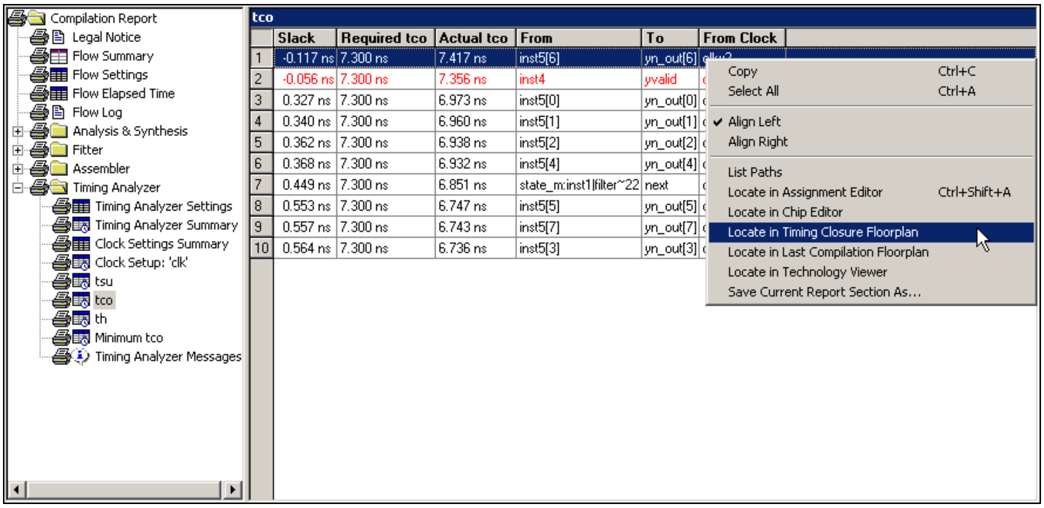
*Figure 7–17. I/O Slack Report*



```
Info: Slack time is -117 ps for clock clkx2 between source register inst5[6] and destination pin yn_out[6]
  Info: + tco requirement for source register and destination pin is 7.300 ns
  Info: - tco from clock to output pin is 7.417 ns
    Info: + Longest clock path from clock clkx2 to source register is 3.163 ns
    Info: + Micro clock to output delay of source is 0.176 ns
    Info: + Longest register to pin delay is 4.078 ns
      Info: 1: + IC(0.000 ns) + CELL(0.000 ns) = 0.000 ns; Loc. = LC_X1_Y3_N6; Fanout = 1; REG Node = 'inst5[6]'
      Info: 2: + IC(1.320 ns) + CELL(2.758 ns) = 4.078 ns; Loc. = Pin_AD5; Fanout = 0; PIN Node = 'yn_out[6]'
      Info: Total cell delay = 2.758 ns ( 67.63 % )
      Info: Total interconnect delay = 1.320 ns ( 32.37 % )
```

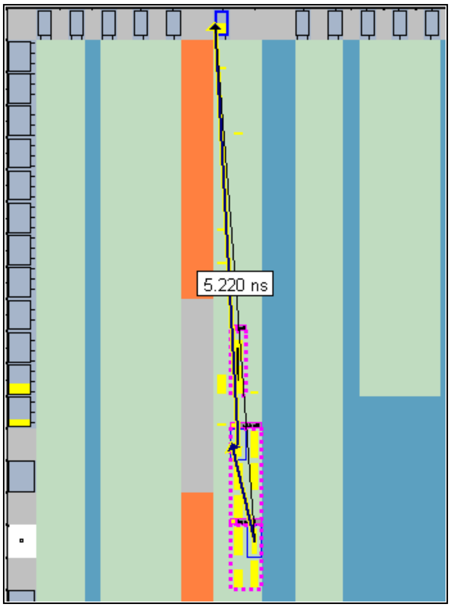
To visually analyze I/O timing, right-click on an I/O entry in the report and select **Locate in Timing Closure Floorplan** as shown in Figures 7–18 and 7–19. The Timing Closure Floorplan Editor is displayed, highlighting the I/O path. Note that you can set the level of detail in the floorplan in the View menu.

*Figure 7–18. Locate Failing Path in Timing Closure Floorplan Editor*



| | Slack | Required tco | Actual tco | From | To | From Clock |
|---|---|---|---|---|---|---|
| 1 | -0.117 ns | 7.300 ns | 7.417 ns | inst5[6] | yn_out[6] | clkx2 |
| 2 | -0.056 ns | 7.300 ns | 7.356 ns | inst4 | yvalid | c |
| 3 | 0.327 ns | 7.300 ns | 6.973 ns | inst5[0] | yn_out[0] | c |
| 4 | 0.340 ns | 7.300 ns | 6.960 ns | inst5[1] | yn_out[1] | c |
| 5 | 0.362 ns | 7.300 ns | 6.938 ns | inst5[2] | yn_out[2] | c |
| 6 | 0.368 ns | 7.300 ns | 6.932 ns | inst5[4] | yn_out[4] | c |
| 7 | 0.449 ns | 7.300 ns | 6.851 ns | state_m:inst1\|filter~22 | next | c |
| 8 | 0.553 ns | 7.300 ns | 6.747 ns | inst5[5] | yn_out[5] | c |
| 9 | 0.557 ns | 7.300 ns | 6.743 ns | inst5[7] | yn_out[7] | c |
| 10 | 0.564 ns | 7.300 ns | 6.736 ns | inst5[3] | yn_out[3] | c |

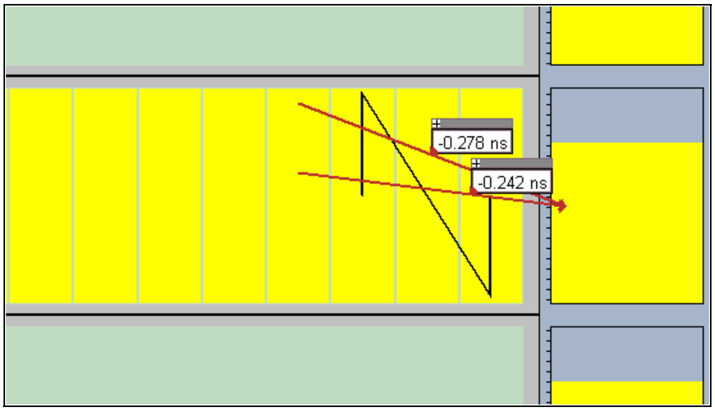
*Figure 7–19. Failing Path in Timing Closure Floorplan Editor, Field View*



In Figure 7–20 the arrows indicate the critical path (i.e., a register) from the beginning point to the end point (i.e., another register). The times shown are the slack figures for each path. Negative slack indicates paths that failed to meet their timing requirements.

To see all the intermediate nodes (i.e., combinational logic cells) on a path and the delay for each level of logic, right-click the title bar above a path's slack number and choose **Expand** (right button pop-up menu). To view all these paths in the **Timing Closure Floorplan Editor** choose **Routing** > **Show Critical Paths** (View menu).

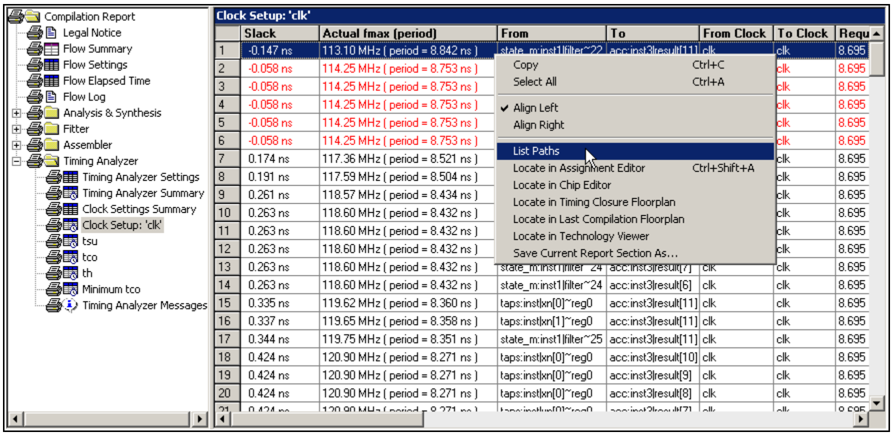
*Figure 7–20. Critical I/O Paths in the Timing Closure Floorplan*



## $f_{MAX}$ Timing Analysis Report File

To determine whether your system performance or $f_{MAX}$ timing requirements are met, the Quartus II software generates a timing analysis report that provides detailed timing information on every clock in your design. This report is accessed by opening the **Timing Analyzer** folder in the **Compilation Report** (Processing menu). The **Clock Setup** folder of the **Compilation Report** provides figures for slack and register-to-register $f_{MAX}$. The paths that are not meeting timing requirements are shown in red. See Figure 7–21.

*Figure 7–21. $f_{MAX}$ Requirements*

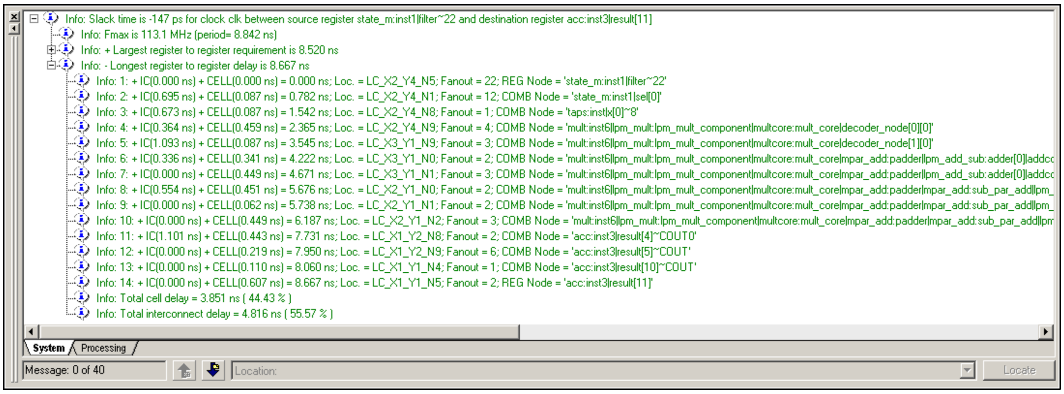


Clock Setup: 'clk'

| | Slack | Actual fmax (period) | From | To | From Clock | To Clock | Requ |
|---|---|---|---|---|---|---|---|
| 1 | -0.147 ns | 113.10 MHz ( period = 8.842 ns ) | state_m:inst1\|filter~22 | acc:inst3\|result[11] | clk | clk | 8.695 |
| 2 | -0.058 ns | 114.25 MHz ( period = 8.753 ns ) | | | | clk | 8.695 |
| 3 | -0.058 ns | 114.25 MHz ( period = 8.753 ns ) | | | | clk | 8.695 |
| 4 | -0.058 ns | 114.25 MHz ( period = 8.753 ns ) | | | | clk | 8.695 |
| 5 | -0.058 ns | 114.25 MHz ( period = 8.753 ns ) | | | | clk | 8.695 |
| 6 | -0.058 ns | 114.25 MHz ( period = 8.753 ns ) | | | | clk | 8.695 |
| 7 | 0.174 ns | 117.36 MHz ( period = 8.521 ns ) | | | | clk | 8.695 |
| 8 | 0.191 ns | 117.59 MHz ( period = 8.504 ns ) | | | | clk | 8.695 |
| 9 | 0.261 ns | 118.57 MHz ( period = 8.434 ns ) | | | | clk | 8.695 |
| 10 | 0.263 ns | 118.60 MHz ( period = 8.432 ns ) | | | | clk | 8.695 |
| 11 | 0.263 ns | 118.60 MHz ( period = 8.432 ns ) | | | | clk | 8.695 |
| 12 | 0.263 ns | 118.60 MHz ( period = 8.432 ns ) | | | | clk | 8.695 |
| 13 | 0.263 ns | 118.60 MHz ( period = 8.432 ns ) | state_m:inst1\|filter~24 | acc:inst3\|result[7] | clk | clk | 8.695 |
| 14 | 0.263 ns | 118.60 MHz ( period = 8.432 ns ) | state_m:inst1\|filter~24 | acc:inst3\|result[6] | clk | clk | 8.695 |
| 15 | 0.335 ns | 119.62 MHz ( period = 8.360 ns ) | taps:inst\|xn[0]~reg0 | acc:inst3\|result[11] | clk | clk | 8.695 |
| 16 | 0.337 ns | 119.65 MHz ( period = 8.358 ns ) | taps:inst\|xn[1]~reg0 | acc:inst3\|result[11] | clk | clk | 8.695 |
| 17 | 0.344 ns | 119.75 MHz ( period = 8.351 ns ) | state_m:inst1\|filter~25 | acc:inst3\|result[11] | clk | clk | 8.695 |
| 18 | 0.424 ns | 120.90 MHz ( period = 8.271 ns ) | taps:inst\|xn[0]~reg0 | acc:inst3\|result[10] | clk | clk | 8.695 |
| 19 | 0.424 ns | 120.90 MHz ( period = 8.271 ns ) | taps:inst\|xn[0]~reg0 | acc:inst3\|result[9] | clk | clk | 8.695 |
| 20 | 0.424 ns | 120.90 MHz ( period = 8.271 ns ) | taps:inst\|xn[0]~reg0 | acc:inst3\|result[8] | clk | clk | 8.695 |

To analyze why timing was not met, right-click on a particular path reported in the **System** tab of the **Message** window (Figure 7–22) and select **List Paths** (right button pop-up menu) to determine the location of the greatest delay. You can expand a selection by clicking the "+" icon at the beginning of the line.

*Figure 7–22. $f_{MAX}$ Slack Report*

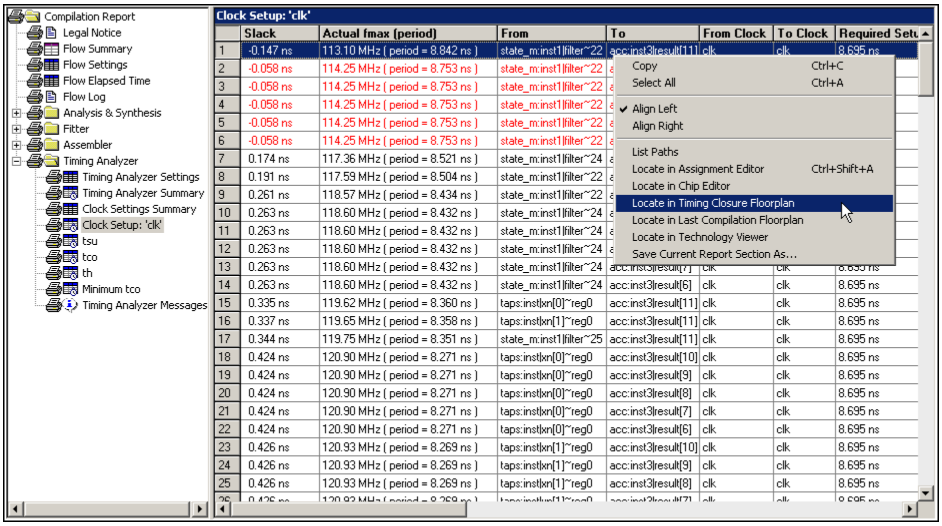


```
Info: Slack time is -147 ps for clock clk between source register state_m:inst1|filter~22 and destination register acc:inst3|result[11]
  Info: Fmax is 113.1 MHz (period= 8.842 ns)
  Info: + Largest register to register requirement is 8.520 ns
  Info: - Longest register to register delay is 8.667 ns
    Info: 1: + IC(0.000 ns) + CELL(0.000 ns) = 0.000 ns; Loc. = LC_X2_Y4_N5; Fanout = 22; REG Node = 'state_m:inst1|filter~22'
    Info: 2: + IC(0.695 ns) + CELL(0.087 ns) = 0.782 ns; Loc. = LC_X2_Y4_N1; Fanout = 12; COMB Node = 'state_m:inst1|sel[0]'
    Info: 3: + IC(0.673 ns) + CELL(0.087 ns) = 1.542 ns; Loc. = LC_X2_Y4_N8; Fanout = 1; COMB Node = 'taps:inst|x[0]~8'
    Info: 4: + IC(0.364 ns) + CELL(0.459 ns) = 2.365 ns; Loc. = LC_X2_Y4_N9; Fanout = 4; COMB Node = 'mult:inst6|lpm_mult:lpm_mult_component|multcore:mult_core|decoder_node[0][0]'
    Info: 5: + IC(1.093 ns) + CELL(0.087 ns) = 3.545 ns; Loc. = LC_X3_Y1_N9; Fanout = 3; COMB Node = 'mult:inst6|lpm_mult:lpm_mult_component|multcore:mult_core|decoder_node[1][0]'
    Info: 6: + IC(0.336 ns) + CELL(0.341 ns) = 4.222 ns; Loc. = LC_X3_Y1_N0; Fanout = 2; COMB Node = 'mult:inst6|lpm_mult:lpm_mult_component|multcore:mult_core|mpar_add:padder|lpm_add_sub:adder[0]|addcc
    Info: 7: + IC(0.000 ns) + CELL(0.449 ns) = 4.671 ns; Loc. = LC_X3_Y1_N1; Fanout = 3; COMB Node = 'mult:inst6|lpm_mult:lpm_mult_component|multcore:mult_core|mpar_add:padder|lpm_add_sub:adder[0]|addcc
    Info: 8: + IC(0.554 ns) + CELL(0.451 ns) = 5.676 ns; Loc. = LC_X2_Y1_N0; Fanout = 2; COMB Node = 'mult:inst6|lpm_mult:lpm_mult_component|multcore:mult_core|mpar_add:padder|mpar_add:sub_par_add|lpm_
    Info: 9: + IC(0.000 ns) + CELL(0.062 ns) = 5.738 ns; Loc. = LC_X2_Y1_N1; Fanout = 2; COMB Node = 'mult:inst6|lpm_mult:lpm_mult_component|multcore:mult_core|mpar_add:padder|mpar_add:sub_par_add|lpm_
    Info: 10: + IC(0.000 ns) + CELL(0.449 ns) = 6.187 ns; Loc. = LC_X2_Y1_N2; Fanout = 3; COMB Node = 'mult:inst6|lpm_mult:lpm_mult_component|multcore:mult_core|mpar_add:padder|mpar_add:sub_par_add|lpm
    Info: 11: + IC(1.101 ns) + CELL(0.443 ns) = 7.731 ns; Loc. = LC_X1_Y2_N8; Fanout = 2; COMB Node = 'acc:inst3|result[4]~COUT0'
    Info: 12: + IC(0.000 ns) + CELL(0.219 ns) = 7.950 ns; Loc. = LC_X1_Y2_N9; Fanout = 6; COMB Node = 'acc:inst3|result[5]~COUT'
    Info: 13: + IC(0.000 ns) + CELL(0.110 ns) = 8.060 ns; Loc. = LC_X1_Y1_N4; Fanout = 1; COMB Node = 'acc:inst3|result[10]~COUT'
    Info: 14: + IC(0.000 ns) + CELL(0.607 ns) = 8.667 ns; Loc. = LC_X1_Y1_N5; Fanout = 2; REG Node = 'acc:inst3|result[11]'
    Info: Total cell delay = 3.851 ns ( 44.43 % )
    Info: Total interconnect delay = 4.816 ns ( 55.57 % )
```

Visually analyze $f_{MAX}$ paths by right-clicking on a path in the report and selecting **Locate in Timing Closure Floorplan** to display the Timing Closure Floorplan Editor, which highlights the path. See Figure 7–23. Figure 7–24 shows the Timing Closure Floorplan Editor displaying a failing path.

Double-clicking the section **Info: - Longest register to register delay is** <*slack value*> **ns** in the list path text locates the path in the Timing Closure Floorplan.

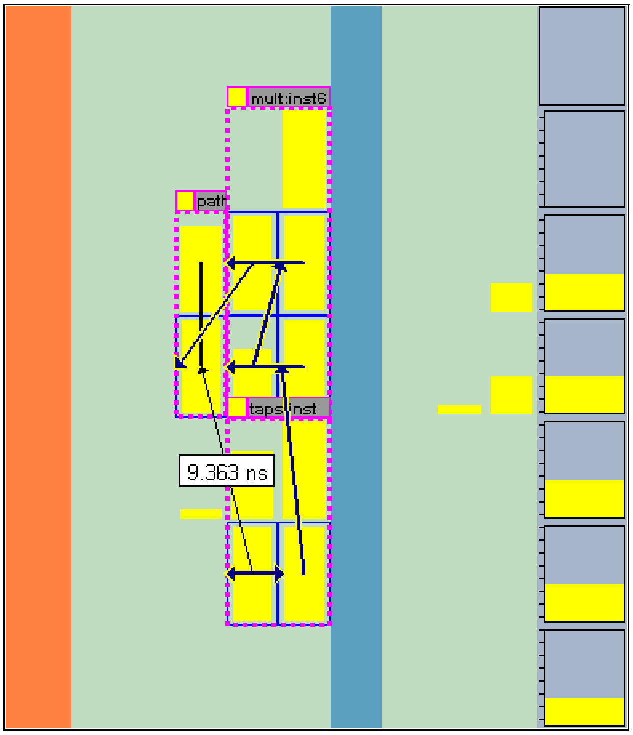
*Figure 7–23. Locate Failing Path in Timing Closure Floorplan*



Clock Setup: 'clk'

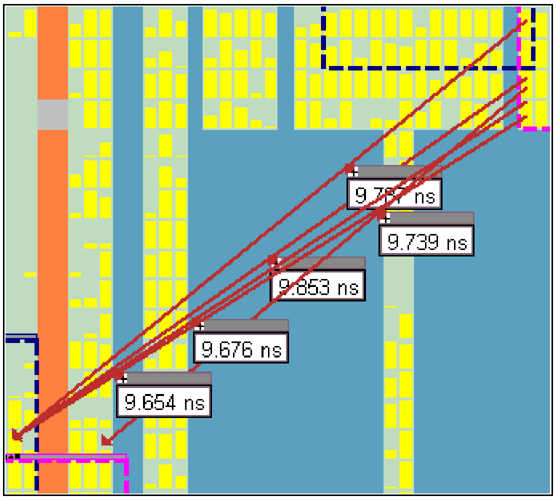
| | Slack | Actual fmax (period) | From | To | From Clock | To Clock | Required Setu |
|---|---|---|---|---|---|---|---|
| 1 | -0.147 ns | 113.10 MHz ( period = 8.842 ns ) | state_m:inst1\|filter~22 | acc:inst3\|result[11] | clk | clk | 8.695 ns |
| 2 | -0.058 ns | 114.25 MHz ( period = 8.753 ns ) | state_m:inst1\|filter~22 | | | | |
| 3 | -0.058 ns | 114.25 MHz ( period = 8.753 ns ) | state_m:inst1\|filter~22 | | | | |
| 4 | -0.058 ns | 114.25 MHz ( period = 8.753 ns ) | state_m:inst1\|filter~22 | | | | |
| 5 | -0.058 ns | 114.25 MHz ( period = 8.753 ns ) | state_m:inst1\|filter~22 | | | | |
| 6 | -0.058 ns | 114.25 MHz ( period = 8.753 ns ) | state_m:inst1\|filter~22 | | | | |
| 7 | 0.174 ns | 117.36 MHz ( period = 8.521 ns ) | state_m:inst1\|filter~24 | | | | |
| 8 | 0.191 ns | 117.59 MHz ( period = 8.504 ns ) | state_m:inst1\|filter~22 | | | | |
| 9 | 0.261 ns | 118.57 MHz ( period = 8.434 ns ) | state_m:inst1\|filter~22 | | | | |
| 10 | 0.263 ns | 118.60 MHz ( period = 8.432 ns ) | state_m:inst1\|filter~24 | | | | |
| 11 | 0.263 ns | 118.60 MHz ( period = 8.432 ns ) | state_m:inst1\|filter~24 | | | | |
| 12 | 0.263 ns | 118.60 MHz ( period = 8.432 ns ) | state_m:inst1\|filter~24 | | | | |
| 13 | 0.263 ns | 118.60 MHz ( period = 8.432 ns ) | state_m:inst1\|filter~24 | acc:inst3\|result[7] | clk | clk | 8.695 ns |
| 14 | 0.263 ns | 118.60 MHz ( period = 8.432 ns ) | state_m:inst1\|filter~24 | acc:inst3\|result[6] | clk | clk | 8.695 ns |
| 15 | 0.335 ns | 119.62 MHz ( period = 8.360 ns ) | taps:inst\|xn[0]~reg0 | acc:inst3\|result[11] | clk | clk | 8.695 ns |
| 16 | 0.337 ns | 119.65 MHz ( period = 8.358 ns ) | taps:inst\|xn[1]~reg0 | acc:inst3\|result[11] | clk | clk | 8.695 ns |
| 17 | 0.344 ns | 119.75 MHz ( period = 8.351 ns ) | state_m:inst1\|filter~25 | acc:inst3\|result[11] | clk | clk | 8.695 ns |
| 18 | 0.424 ns | 120.90 MHz ( period = 8.271 ns ) | taps:inst\|xn[0]~reg0 | acc:inst3\|result[10] | clk | clk | 8.695 ns |
| 19 | 0.424 ns | 120.90 MHz ( period = 8.271 ns ) | taps:inst\|xn[0]~reg0 | acc:inst3\|result[9] | clk | clk | 8.695 ns |
| 20 | 0.424 ns | 120.90 MHz ( period = 8.271 ns ) | taps:inst\|xn[0]~reg0 | acc:inst3\|result[8] | clk | clk | 8.695 ns |
| 21 | 0.424 ns | 120.90 MHz ( period = 8.271 ns ) | taps:inst\|xn[0]~reg0 | acc:inst3\|result[7] | clk | clk | 8.695 ns |
| 22 | 0.424 ns | 120.90 MHz ( period = 8.271 ns ) | taps:inst\|xn[0]~reg0 | acc:inst3\|result[6] | clk | clk | 8.695 ns |
| 23 | 0.426 ns | 120.93 MHz ( period = 8.269 ns ) | taps:inst\|xn[1]~reg0 | acc:inst3\|result[10] | clk | clk | 8.695 ns |
| 24 | 0.426 ns | 120.93 MHz ( period = 8.269 ns ) | taps:inst\|xn[1]~reg0 | acc:inst3\|result[9] | clk | clk | 8.695 ns |
| 25 | 0.426 ns | 120.93 MHz ( period = 8.269 ns ) | taps:inst\|xn[1]~reg0 | acc:inst3\|result[8] | clk | clk | 8.695 ns |
| 26 | 0.426 ns | 120.93 MHz ( period = 8.269 ns ) | taps:inst\|xn[1]~reg0 | acc:inst3\|result[7] | clk | clk | 8.695 ns |

*Figure 7–24. Failing Path in Timing Closure Floorplan*



You can view all failing paths in the Timing Closure Floorplan Editor using the **Show Critical Paths** feature. Figure 7–25 shows critical $f_{MAX}$ paths in the Timing Closure Floorplan Editor.

*Figure 7–25. Critical Paths in the Timing Closure Floorplan Editor*



The *Design Optimization for Altera Devices* chapter in Volume 2 of the *Quartus II Handbook* shows you how to optimize your design in the Quartus II software. With the options and tools available in the Timing Closure Floorplan and the techniques described in that chapter, the Quartus II software can assist you in achieving timing closure in a more time efficient manner.

## Conclusion

Design analysis for timing closure is a fundamental requirement for optimal performance in highly complex designs. The Quartus II Timing Closure Floorplan Editor assists in closing timing quickly on complex designs, reduces iterations by providing more intelligent and better linkage between analysis and assignment tools, and balances multiple design constraints including multiple clocks, routing resources, and area constraints.

# 8. Netlist Optimizations and Physical Synthesis

qii52007-2.0

## Introduction

The Quartus® II software offers advanced netlist optimization options, including physical synthesis, to optimize your design further than the optimization performed in the course of the standard Quartus II compilation flow. Device support for these optimizations vary; see the appropriate section for details.

The effect of these options depends on the structure of your design, but netlist optimizations can help improve the performance of your design regardless of the synthesis tool used. These options work with your design's atom netlist, which specifies a design as Altera®-specific primitives. An example of an atom netlist file is an EDIF Input File (**.edf**) or a Verilog Quartus Mapping (**.vqm**) file generated by a third-party synthesis tool, or an internal netlist generated within the Quartus II software. Netlist optimizations are applied at different stages of the Quartus II compilation flow, either during synthesis or during fitting.

The synthesis netlist optimizations occur during the synthesis stage of the Quartus II compilation flow. Operating on the output from a third-party synthesis tool, or operating as an intermediate step in the Quartus II standard integrated synthesis, these optimizations make changes to the synthesis netlist. These netlist changes are beneficial in terms of area or speed, depending on your selected optimization technique.

The physical synthesis optimizations take place during the fitter stage of the Quartus II compilation flow. Physical synthesis optimizations make placement-specific changes to the netlist that improve performance results for a specific Altera device.

This chapter explains how the netlist optimizations in the Quartus II software can modify your design's netlist and help improve your quality of results. The following sections "Synthesis Netlist Optimizations" on page 8–2 and "Physical Synthesis Optimizations" on page 8–9 explain how the available optimizations work. This chapter also provides information on preserving your compilation results through back-annotation and writing out a new netlist, and provides guidelines for applying the various options.

☞ When synthesis netlist optimization or physical synthesis options are turned on, the node names for primitives in the design can change. The fact that nodes may be renamed must be considered if you are using a LogicLock™ or verification flow that may require fixed node names, such as SignalTap® II or formal verification. Primitive node names are specified during synthesis and are contained in atom netlists from third-party synthesis tools. When netlist optimizations are applied, node names may change as primitives are created and removed. HDL attributes applied to preserve logic in third-party synthesis tools cannot be honored because those attributes are not written into the atom netlist read by the Quartus II software. If you are synthesizing in the Quartus II software, you can use the Preserve Register (`preserve`) and Keep Combinational Logic (`keep`) attributes to maintain certain nodes in the design. For more information on using these attributes during synthesis in the Quartus II software, see the *Quartus II Integrated Synthesis* chapter in Volume 1 of the *Quartus II Handbook.*

☞ Any nodes or entities that have the logic option **Netlist Optimizations** set to **Never allow** are not affected during netlist optimizations (including physical synthesis). This logic option can be applied with the **Assignment Editor** (Assignments menu) if you want to disable all netlist optimizations for parts of your design.

## Synthesis Netlist Optimizations

You can view and modify the synthesis netlist optimization options in the **Synthesis Netlist Optimizations** page under **Analysis & Synthesis Settings** in the **Settings** dialog box (Assignments Menu).

The sections "WYSIWYG Primitive Resynthesis" on page 8–2 and "Gate-Level Register Retiming" on page 8–4 describe these synthesis netlist optimizations, and how they can help improve the quality of results for your design.

### WYSIWYG Primitive Resynthesis

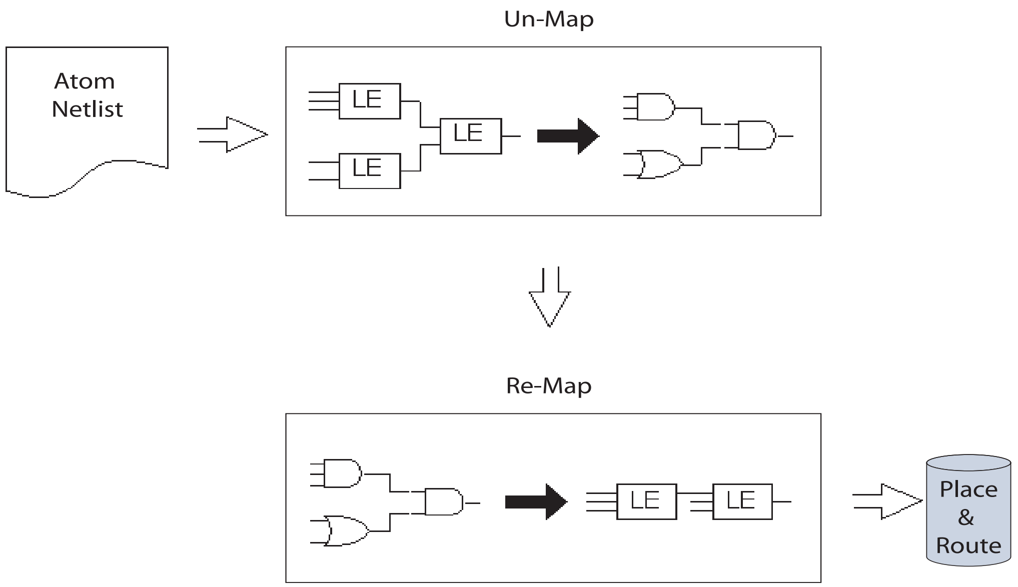
You can use the **Perform WYSIWYG primitive resynthesis (using optimization technique specified in Analysis & Synthesis settings)** synthesis option when you have an atom netlist file that specifies a design as Altera-specific primitives. Atom netlist files can be either an EDIF (**.edf**) or VQM (**.vqm**) file generated by a third-party synthesis tool. This option can be found on the **Synthesis Netlist Optimizations** page under **Analysis & Synthesis Settings** in the **Settings** dialog box (Assignments menu). If you want to perform WYSIWYG resynthesis on only a portion of your design, you can use the **Assignment Editor** (Assignments menu) to assign the **Perform WYSIWYG primitive resynthesis** logic option to a lower-level entity in your design. This option can be used with the Cyclone™ II, MAX® II, Stratix® II, Stratix GX, Stratix, Cyclone, or APEX™ device families.

The **Perform WYSIWYG primitive resynthesis** option directs the Quartus II software to un-map the logic elements (LEs) in an atom netlist to logic gates, and then re-map the gates back to Altera-specific primitives. This feature allows the Quartus II software to use different techniques specific to the device architecture during the re-mapping process. The Quartus II technology mapper optimizes the design for **Speed**, **Area**, or **Balanced**, according to the setting of the **Optimization Technique** option on the **Analysis & Synthesis Settings** page in the **Settings** dialog box (Assignments menu). The Balanced setting is default for most Altera device families; this setting optimizes the timing-critical parts of the design for speed and the rest for area.

See the *Quartus II Integrated Synthesis* chapter in Volume 1 of the *Quartus II Handbook* for details on the Optimization Technique option.

Figure 8–1 shows the Quartus II software flow for this feature.

***Figure 8–1. WYSIWYG Primitive Resynthesis***



This option is not applicable if you are using Quartus II integrated synthesis. With Quartus II synthesis, you do not need to un-map Altera primitives; they are already mapped during the synthesis step using the techniques that are used with the WYSIWYG primitive resynthesis option.

The **Perform WYSIWYG primitive resynthesis** option only un-maps and re-maps logic cell (also referred to as LCELL or LE) primitives and regular I/O primitives (which may contain registers). DDR (double data rate) I/O primitives, memory primitives, digital signal processing (DSP) primitives, and logic cells in carry/cascade chains are not touched. Logic specified in an encrypted VQM or EDIF file, such as third-party intellectual property (IP), is not touched.

Turning on this option can cause drastic changes to the node names in the VQM or EDIF atom netlist from your third-party synthesis tool, because the primitives in the netlist are being broken apart and then remapped within the Quartus II software. Registers can be minimized away and duplicates removed, but registers that are not removed have the same name after remapping.

Any nodes or entities that have the **Netlist Optimizations** logic option set to **Never allow** are not affected during WYSIWYG primitive resynthesis. This logic option can be applied with the **Assignment Editor** (Assignments menu) if you want to disable WYSIWYG resynthesis for parts of your design.

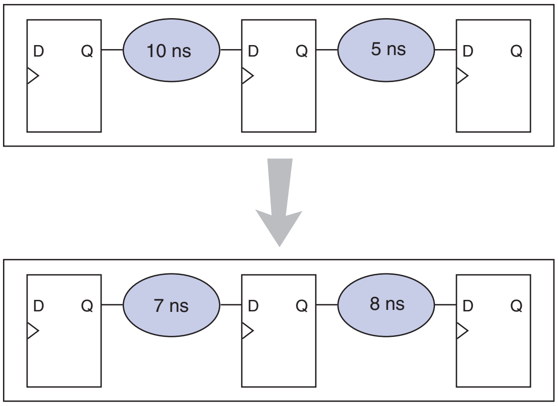
## Gate-Level Register Retiming

The **Perform gate-level register retiming** option enables movement of registers across combinational logic to balance timing, allowing the Quartus II software to trade off the delay between timing-critical paths and non-critical paths. See Figure 8–2 for an example. It can be used with the Cyclone II, MAX II, Stratix II, Stratix, Stratix GX, Cyclone, and APEX device families. The option is found on the **Synthesis Netlist Optimizations** page under **Analysis & Synthesis Settings** in the **Settings** dialog box (Assignments menu).

The functionality of your design is not changed when the **Perform gate-level register retiming** option is turned on. However, if any registers in your design have the **Power-Up Don't Care** logic option assigned, the values of registers during power-up may change due to this register and logic movement. The **Power-Up Don't Care** logic option is turned on globally by default. You can change the default setting for the option on the **Analysis & Synthesis Settings** page in the **Settings** dialog box (Assignments menu) by clicking **More Settings**. You can also set the logic option for individual registers or entities using the Assignment Editor. Registers that are explicitly assigned power-up values are not combined with registers that have been explicitly assigned other values.

Figure 8–2 shows an example of gate-level register retiming where the 10 ns critical delay is reduced by moving the register relative to the combinational logic.

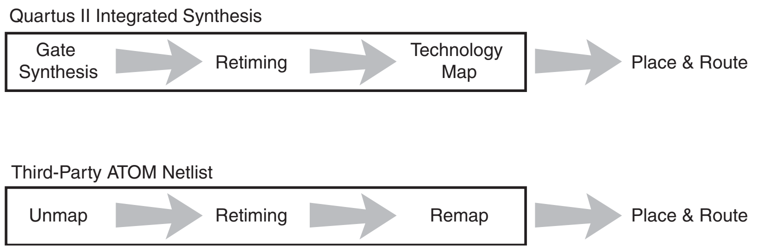
*Figure 8–2. Gate-Level Register Retiming Diagram*



Register retiming makes changes at the gate level. If you are using an atom netlist from a third-party synthesis tool, you must also use the **Perform WYSIWYG primitive resynthesis** option to un-map atom primitives to gates (so that register retiming can be performed) and then to re-map gates to Altera primitives. If your design uses Quartus II integrated synthesis, retiming occurs during synthesis before the design is mapped to Altera primitives. Megafunctions instantiated in a design are always synthesized using the Quartus II software.

The design flows for the case of integrated Quartus II synthesis and a third-party atom netlist are shown in Figure 8–3.

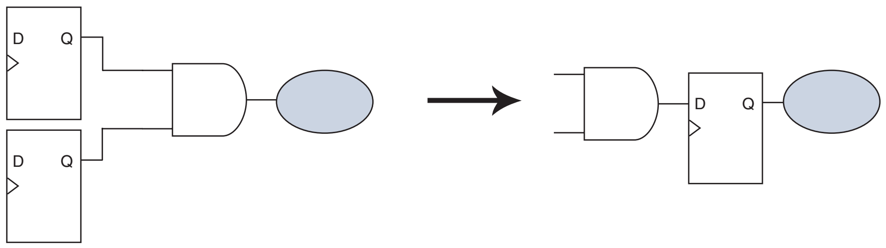
*Figure 8–3. Gate-Level Synthesis*



The gate-level register retiming options only moves registers across combinational gates. Registers are not moved across LCELL primitives instantiated by the user, memory blocks, DSP blocks, or carry/cascade chains that you have instantiated. Carry/cascade chains are always left intact when performing register retiming.

One of the benefits of register retiming is the ability to move registers from the inputs of a combinational logic block to the output, potentially combining the registers. In this case, some registers are removed, and one is created at the output. This case is shown in Figure 8–4.

*Figure 8–4. Combining Registers with Register Retiming*



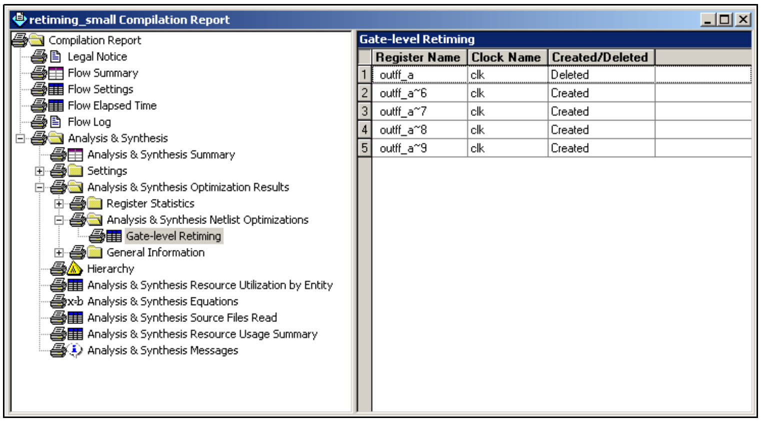
Retiming can only move and combine registers in this type of situation if the following conditions are met:

- All registers have the same clock signal
- All registers have the same clock enable signal
- All registers have asynchronous control signals that are active under the same conditions
- Only one register has an asynchronous load other than VCC or GND

Retiming can always create multiple registers at the input of a combinational block from a register at the output of a combinational block. In this case, the new registers have the same clock and clock enable. The asynchronous control signals and power-up level are derived from previous registers to provide equivalent functionality.

The **Gate-level Retiming** report provides a list of registers that were created and removed during register retiming. This report can be found in the **Analysis & Synthesis Netlist Optimizations** section of the **Analysis & Synthesis Optimization Results** folder under **Analysis & Synthesis** in the **Compilation Report** (Processing menu). See Figure 8–5. Note that the node names for these registers change during the retiming process.

***Figure 8–5. Gate-Level Retiming Report***


retiming_small Compilation Report
Compilation Report
Legal Notice
Flow Summary
Flow Settings
Flow Elapsed Time
Flow Log
Analysis & Synthesis
Analysis & Synthesis Summary
Settings
Analysis & Synthesis Optimization Results
Register Statistics
Analysis & Synthesis Netlist Optimizations
Gate-level Retiming
General Information
Hierarchy
Analysis & Synthesis Resource Utilization by Entity
Analysis & Synthesis Equations
Analysis & Synthesis Source Files Read
Analysis & Synthesis Resource Usage Summary
Analysis & Synthesis Messages
Gate-level Retiming

| | Register Name | Clock Name | Created/Deleted |
|---|---|---|---|
| 1 | outff_a | clk | Deleted |
| 2 | outff_a~6 | clk | Created |
| 3 | outff_a~7 | clk | Created |
| 4 | outff_a~8 | clk | Created |
| 5 | outff_a~9 | clk | Created |

You can set the **Netlist Optimizations** logic option to **Never Allow** for registers to prevent movement during register retiming. This option can be applied either to individual registers or entities in the design and is applied through the **Assignment Editor** (Assignments menu).

The following registers are not moved during gate-level register retiming:

- Registers that have any timing constraint other than global $f_{MAX}$, $t_{SU}$, or $t_{CO}$. For example, any node affected by a Multicycle or Cut Timing assignment is not moved.
- Registers that feed asynchronous control signals on another register
- Registers feeding the clock of another register
- Registers feeding a register in another clock domain
- Registers that are fed by a register in another clock domain
- Registers connected to serializer/deserializer (SERDES)

- Registers that have the **Netlist Optimizations** logic option set to **Never Allow**
- Registers feeding output pins (without logic between the register and the pin)
- Registers fed by an input pin (without logic between register and input pin)
- Both registers in a connection from input pin-register-register connection if both registers have the same clock and the first register does not fan out to anywhere else (since these are considered synchronization registers).

If you want to consider registers with any of these conditions for register retiming, you can override these rules by setting the **Netlist Optimizations** logic option to **Always Allow** on a given set of registers.

### *Allow Register Retiming to Trade-Off $t_{SU}$/$t_{CO}$ with $f_{MAX}$*

The **Allow register retiming to trade off $t_{SU}$/ $t_{CO}$ with $f_{MAX}$** option on the **Synthesis Netlist Optimizations** page under **Analysis & Synthesis Settings** in the **Settings** dialog box (Assignments menu) determines whether the Quartus II compiler should attempt to increase $f_{MAX}$ at the expense of $t_{SU}$ or $t_{CO}$ times. This option affects the optimizations performed due to the gate-level register retiming option.

When both the **Perform gate-level register retiming** and the **Allow register retiming to trade off $t_{SU}$/$t_{CO}$ with $f_{MAX}$** options are turned on, retiming can affect registers that feed and are fed by I/O pins. If the latter option is not turned on, the retiming option does not touch any registers that connect to I/O pins through one or more levels of combinational logic.

## Preserving Your Synthesis Netlist Optimization Results

Given the same source code and settings on a given system, the Quartus II software generates the same results on every compilation. Therefore, it is typically not necessary to take any steps to preserve your results from compilation to compilation. When changes are made to the source code or to the settings, you usually get the best results by allowing the software to compile without using any previous compilation results or location assignments. In addition, in some cases you may skip the synthesis stage of the compile by avoiding running **Analysis & Synthesis**, or **quartus_map**, and instead just running the Fitter or another desired Quartus II executable.

However, if you wish, you may preserve the netlist resulting from netlist optimizations. Preserving the netlist can be required if you use the LogicLock flow to preserve placement and/or import one design into another. If you are using any Quartus II synthesis netlist optimization options, you can save your optimized results by turning on the **Save a node-level netlist into a persistent source file (Verilog Quartus Mapping File)** option on the **Compilation Process** page in the **Settings** dialog box (Assignments menu). This option saves your final results as an atom-based netlist in Verilog Quartus Mapping File (**.vqm**) format. By default, the Quartus II software places the VQM file in the **atom_netlists** directory under the current project directory. If you'd like to create a different VQM file using different Quartus II settings, you may do so by changing the file name setting on the **Compilation Process** page in the **Settings** dialog box (Assignments menu).

If you are using the synthesis netlist optimizations (and not any physical synthesis optimizations), generating a VQM file is optional. You may lock down the location of all LEs and other device resources in the design using the **Back-Annotate Assignments** command (Assignments menu) with or without a Quartus II-generated VQM file. Altera recommends against using back-annotated location assignments unless the design has been finalized. Making any changes to the design invalidates your back-annotated location assignments. If you need to make changes later on, use the new source HDL code as your input files, and remove the back-annotated assignments corresponding to the old code or netlist.

If you create a VQM file and wish to recompile the design, use the new VQM file as the input source file and turn off the synthesis netlist optimizations for the new compilation.

## Physical Synthesis Optimizations

Traditionally, the Quartus II design flow has involved separate steps of synthesis and fitting. The synthesis step optimizes the logical structure of a circuit for area, speed, or both. The fitter then places and routes the logic elements to ensure critical portions of logic are close together and use the fastest possible routing resources. While this push-button flow produces excellent results, the synthesis stage is unable to anticipate the routing delays seen in the fitter. Since routing delays are a significant part of the typical critical path delay, performing synthesis operations with physical delay knowledge allows the tool to target its timing-driven optimizations at these parts of the design. This tight integration of the fitting and synthesis processes is known as physical synthesis.

The following sections describe the physical synthesis optimizations available in the Quartus II software, and how they can help improve your performance results. Physical synthesis optimization options can be used with the MAX II, Stratix II, Stratix, Stratix GX, or Cyclone device families.

You can view and modify the physical synthesis optimization options on the **Physical Synthesis Optimizations** page in the **Fitter Settings** section of the **Settings** dialog box (Assignments Menu).

The physical synthesis optimizations are split into two groups, those that affect only combinational logic and not registers, and those that can affect registers. The options are split to allow designers to keep their registers intact for formal verification or other reasons.

The following physical synthesis optimizations are available:

- Physical synthesis for combinational logic
- Physical synthesis for registers:
    - Register duplication
    - Register retiming

You can control the effect of physical synthesis with the **Physical synthesis effort** option. The default selection is **Normal**. The **Extra** effort setting uses extra compile time to try for extra circuit performance, while the **Fast** effort setting uses less compilation time than **Normal** but may not achieve the same gains.

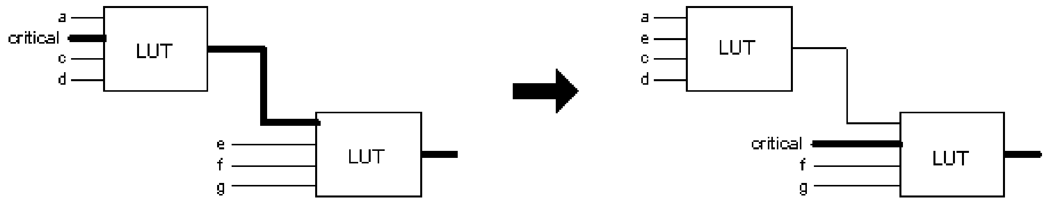
The **Physical Synthesis** report in the **Fitter Netlist Optimizations** section under **Fitter** in the **Compilation Report** (Processing menu) provides a list of atoms that were modified, created, or deleted during physical synthesis. See the"Physical Synthesis Report" on page 8–13.

Nodes or entities that have the **Netlist Optimizations** logic option set to **Never Allow** are not affected by the **Physical Synthesis** algorithms. This logic option can be applied with the **Assignment Editor** (Assignments menu) if you want to disable physical synthesis optimizations for parts of your design.

## Physical Synthesis for Combinational Logic

The **Perform physical synthesis for combinational logic** option on the **Physical Synthesis Optimizations** page in the **Fitter** section of the **Settings** dialog box (Assignments menu) allows the Quartus II fitter to resynthesize the design to reduce delay along the critical path. Physical Synthesis can achieve this type of optimization by swapping the look-up table (LUT) ports within LEs so that the critical path has fewer layers through which to travel. See Figure 8–6 for an example. This option also allows the duplication of LUTs to enable further optimizations on the critical path.

*Figure 8–6. Physical Synthesis for Combinational Logic*



In first case, the critical input feeds through the first LUT to the second LUT. The Quartus II software swaps the critical input to the first LUT with an input feeding the second LUT. This reduces the number of LUTs contained in the critical path. The synthesis information for each LUT is altered to maintain design functionality.

The **Physical Synthesis for combinational logic** option only affects combinational logic in the form of LUTs. The registers contained in the affected logic cells are not modified. Inputs into memory blocks, DSP blocks, and I/O elements are not swapped.

The Quartus II software does not perform combinational optimization on logic cells that have the following properties:

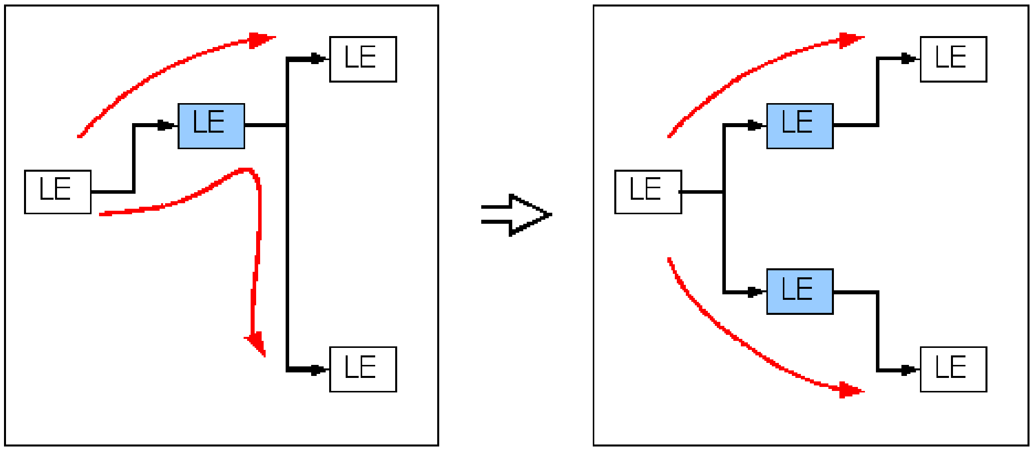
- Are part of a carry/cascade chain
- Drive global signals
- Are constrained to a single logic array block (LAB) location
- Have the **Netlist Optimizations** option set to **Never Allow**

If you want to consider logic cells with any of these conditions for physical synthesis, you can override these rules by setting the **Netlist Optimizations** logic option to **Always Allow** on a given set of nodes.

## Physical Synthesis for Registers - Register Duplication

The **Perform register duplication** fitter option on the **Physical synthesis Optimizations** page in the **Fitter Settings** section of the **Settings** dialog box allows the Quartus II fitter to duplicate registers based on fitter placement information. Combinational logic can also be duplicated when this option is enabled. A logic cell that fans out to multiple locations can be duplicated to reduce the delay of one path without degrading the delay of another. The new logic cell may be placed closer to critical logic without affecting the other fan-out paths of the original logic cell. Figure 8–7 shows an example of register duplication.

*Figure 8–7. Register Duplication*



The Quartus II software does not perform register duplication on logic cells that have the following properties:

- Are part of a carry/cascade chain
- Contain registers that drive asynchronous control signals on another register
- Contain registers that drive the clock of another register
- Contain registers that drive global signals
- Contain registers that are constrained to a single LAB location
- Contain registers that are driven by input pins without a $t_{SU}$ constraint
- Contain registers that are driven by a register in another clock domain
- Are considered virtual I/O pins
- Have the **Netlist Optimizations** option set to **Never Allow**

For more information on virtual I/O pins, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook.*

If you want to consider logic cells with any of these conditions for physical synthesis, you can override these rules by setting the **Netlist Optimizations** logic option to **Always Allow** on a given set of nodes.

## Physical Synthesis for Registers - Register Retiming

The **Perform register retiming** fitter option in the **Physical Synthesis Optimizations** page in the **Fitter Settings** section of the **Settings** dialog box allows the Quartus II fitter to move registers across combinational logic to balance timing. This option enables algorithms similar to the Perform gate-level register retiming option (see ""Gate-Level Register Retiming" on page 8–4). This option applies to the atom level (registers and combinational logic have already been placed into logic cells), and it compliments the synthesis gate-level option.

The Quartus II software does not perform register retiming on logic cells that have the following properties:

- Are part of a cascade chain
- Contain registers that drive asynchronous control signals on another register
- Contain registers that drive the clock of another register
- Contain registers that drive a register in another clock domain
- Contain registers that are driven by a register in another clock domain
- Contain registers that are constrained to a single LAB location
- Contain registers that are connected to serializer/deserializer (SERDES)
- Are considered virtual I/O pins
- Registers that have the **Netlist Optimizations** logic option set to **Never Allow**

For more information on virtual I/O pins, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook.*

If you want to consider logic cells with any of these conditions for physical synthesis, you can override these rules by setting the **Netlist Optimizations** logic option to **Always Allow** on a given set of registers.

## Physical Synthesis Report

All the Physical Synthesis optimizations write results to the **Physical Synthesis** report in the **Fitter Netlist Optimizations** section under **Fitter** in the **Compilation Report** (Processing menu). This report provides a list of atoms that were modified, created, and deleted during physical synthesis. Note that the node names for these atoms change during the physical synthesis process.

## Preserving Your Physical Synthesis Results

Given the same source code and settings on a given system, the Quartus II software generates the same results on every compilation. Therefore, it is typically not necessary to take any steps to preserve your results from compilation to compilation. When changes are made to the source code or to the settings, you usually get the best results by allowing the software to compile without using any previous compilation results or location assignments. However, if you do wish to preserve the compilation results, make sure to follow the guidelines outlined in this section.

If you are using any Quartus II physical synthesis optimization options, you can save your optimized results using the **Save a node-level netlist into a persistent source file (Verilog Quartus Mapping File)** option on the **Compilation Process** page in the **Settings** dialog box (Assignments menu). This option saves your final results as an atom-based netlist in VQM file format. By default, the Quartus II software places the VQM File in the **atom_netlists** directory under the current project directory. If you want to create a different VQM file using different Quartus II settings, you may do so by changing the file name setting on the **Compilation Process** page in the **Settings** dialog box (Assignments menu).

If you are using the physical synthesis optimizations and you wish to lock down the location of all LEs and other device resources in the design using the **Back-Annotate Assignments** command (Assignments menu), a VQM netlist is required to preserve the changes that were made to your original netlist. Since the physical synthesis optimizations depend on the placement of the nodes in the design, back-annotating the placement changes the results from physical synthesis. Changing the results means that node names are different, and your back-annotated locations are no longer valid.

Altera recommends against using a Quartus II-generated VQM or back-annotated location assignments with Physical Synthesis Optimizations unless the design has been finalized. Making any changes to the design invalidates your physical synthesis results and back-annotated location assignments. If you need to make changes later, use the new source HDL code as your input files, and remove the back-annotated assignments corresponding to the Quartus II-generated VQM.

To back-annotate logic locations for a design that was compiled with physical synthesis optimizations, first create a VQM. When recompiling the design with the hard logic location assignments, use the new VQM file as the input source file and turn off the physical synthesis optimizations for the new compilation.

If importing a VQM and back-annotated locations into another project that has any **Netlist Optimizations** turned on, it is important to apply the **Netlist Optimizations** = **Never Allow** constraint, to make sure node names don't change, otherwise the back-annotated location or LogicLock assignments are not valid.

## Applying Netlist Optimization Options

Netlist optimizations options can have various effects on different designs. Designs that are well coded or have already been restructured to balance critical path delays may not see a noticeable difference in performance.

To obtain optimal results when using netlist optimization options, you may need to vary the options applied to find the best results. By default, all options are off. Turning on additional options leads to the largest effect on the node names in the design. Take this into consideration if you are using a LogicLock or verification flow such as SignalTap II or formal verification that requires fixed or known node names. In general, applying all of the Physical Synthesis options at the Extra effort level produces the best results for those options, but adds significantly to the compilation time. You can use the Physical synthesis effort option to decrease the compilation time.

The Synthesis Netlist Optimizations typically do not add much compilation time, relative to the overall design compilation time.

When using a third-party atom netlist (VQM or EDIF), the **WYSIWYG Primitive Resynthesis** option must be turned on in order to use the **Gate-level Register Retiming** option.

A design space explorer (DSE) Tcl/Tk script is provided with the Quartus II software to automate the application of various sets of netlist optimization options.

For more information on using the DSE script to run multiple compilations, see the *Design Space Explorer* chapter in Volume 2 of the *Quartus II Handbook*.

For information on typical performance results using combinations of netlist optimization options and other optimization techniques, see the *Design Optimization for Altera Devices* chapter in Volume 2 of the *Quartus II Handbook*.

## Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script.

For detailed information about specific scripting command options and Tcl API packages, type `quartus_sh --qhelp` at a system command prompt to run the Quartus II Command-Line and Tcl API Help utility.

For more information on Quartus II scripting support, including examples, refer to the *Tcl Scripting* and *Command-Line Scripting* chapters in Volume 2 of the *Quartus II Handbook*.

You can specify many of the options described in this section either on an instance, or at a global level, or both.

Use the following Tcl command to make a global assignment:

```
set_global_assignment -name <QSF variable name> <value> ↵
```

Use the following Tcl command to make an instance assignment:

```
set_instance_assignment -name <QSF variable name> <value> -to <instance name> ↵
```

### Synthesis Netlist Optimizations

Table 8–1 lists the QSF variable name and applicable values for the settings discussed in "Synthesis Netlist Optimizations" on page 8–2. The QSF variable name is used in the Tcl assignment to make the setting along with the appropriate value. The Type column indicates whether the setting is supported as a Global setting, an Instance setting, or both.

*Table 8–1. Synthesis Netlist Optimizations and Associated Settings (Part 1 of 2)*

| Setting Name | QSF Variable Name | Values | Type |
|---|---|---|---|
| Perform WYSIWYG Primitive Resynthesis | ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP | ON, OFF | Global, Instance |
| Optimization Technique | <*Device Family Name*>_OPTIMIZATION_TECHNIQUE | AREA, SPEED, BALANCED | Global, Instance |
| Perform Gate-Level Register Retiming | ADV_NETLIST_OPT_SYNTH_GATE_RETIME | ON, OFF | Global |
| Power-Up Don't Care | ALLOW_POWER_UP_DONT_CARE | ON, OFF | Global |
| Allow Register Retiming to trade off Tsu/Tco with $f_{MAX}$ | ADV_NETLIST_OPT_RETIME_CORE_AND_IO | ON, OFF | Global |

*Table 8–1. Synthesis Netlist Optimizations and Associated Settings (Part 2 of 2)*

| Setting Name | QSF Variable Name | Values | Type |
|---|---|---|---|
| Save a node-level netlist into a persistent source file | LOGICLOCK_INCREMENTAL_COMPILE_ASSIGNMENT | ON, OFF | Global |
| | LOGICLOCK_INCREMENTAL_COMPILE_FILE | <*filename*> | |
| Allow Netlist Optimizations | ADV_NETLIST_OPT_ALLOWED | "ALWAYS ALLOW", DEFAULT, "NEVER ALLOW" | Instance |

## Physical Synthesis Optimizations

Table 8–2 lists the QSF variable name and applicable values for the settings discussed in "Physical Synthesis Optimizations" on page 8–9. The QSF variable name is used in the Tcl assignment to make the setting, along with the appropriate value. The Type column indicates whether the setting is supported as a Global setting, an Instance setting, or both.

*Table 8–2. Physical Synthesis Optimizations and Associated Settings*

| Setting Name | QSF Variable Name | Values | Type |
|---|---|---|---|
| Physical Synthesis for Combinational Logic | PHYSICAL_SYNTHESIS_COMBO_LOGIC | ON, OFF | Global |
| Perform Register Duplication | PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION | ON, OFF | Global |
| Perform Register Retiming | PHYSICAL_SYNTHESIS_REGISTER_RETIMING | ON, OFF | Global |
| Power-Up Don't Care | ALLOW_POWER_UP_DONT_CARE | ON, OFF | Global |
| Allow Netlist Optimizations | ADV_NETLIST_OPT_ALLOWED | "ALWAYS ALLOW", DEFAULT, "NEVER ALLOW" | Instance |
| Save a node-level netlist into a persistent source file | LOGICLOCK_INCREMENTAL_COMPILE_ASSIGNMENT | ON, OFF | Global |
| | LOGICLOCK_INCREMENTAL_COMPILE_FILE | <*filename*> | |

### Back-Annotating Assignments

Use the `logiclock_back_annotate` Tcl command to back-annotate resources in your design. This command can back-annotate resources in LogicLock regions, and resources in designs without LogicLock regions.

For more information on back-annotating assignments, refer to "Preserving Your Synthesis Netlist Optimization Results" on page 8–8 or "Preserving Your Physical Synthesis Results" on page 8–14.

The following Tcl command back-annotates all registers in your design.

```
logiclock_back_annotate -resource_filter "REGISTER"
```

The `logiclock_back_annotate` command is in the `backannotate` package.

## Conclusion

Synthesis Netlist Optimizations and Physical Synthesis Optimizations work in different ways to restructure and optimize your design netlist. Taking advantage of these Quartus II Netlist Optimizations can help improve your quality of results.

# 9. Design Space Explorer

qii52008-2.0

## Introduction

The Quartus® II software includes many advanced optimization algorithms to help you achieve timing closure. The various settings and parameters control the behavior of the algorithms. These options provide complete control over the Quartus II software optimization techniques.

Because each FPGA design is unique, there is no standard set of options that always results in the best performance. Each design requires a unique set of options to achieve optimal performance. This section describes the Design Space Explorer (DSE), a utility that automates the process of finding the best set of options for your design. DSE explores the design space of your design by applying various optimization techniques and analyzing the results.

### DSE Concepts

This section provides an explanation of concepts and terminology used by DSE.

#### *Exploration Space & Exploration Point*

Before a design is explored by DSE, an exploration space is created. An exploration space is a composition of various Synthesis and Fitter settings that are available in the Quartus II software. A single group of settings in the exploration space is referred to as a *point*. DSE traverses the points in an exploration space to determine the optimal settings for your design.

#### *Seed & Seed Sweeping*

The Quartus II Fitter utilizes seeds that specify the starting value which randomly determines the initial placement for the current design. The value of the seed can be any non-negative integer value. Changing the starting value may or may not produce better fitting. By varying the value of the seed value or seed sweeping, an optimal value can be determined for the current design.

DSE extends the concept of fitter seed sweeping with exploration spaces, providing a method for sweeping through general compilation and fitter parameters to find the best options for your design. You can run DSE in a variety of exploration modes, ranging from an exhaustive try-all-options-and-values mode to one that focuses on one parameter.

### DSE Exploration

DSE compares all exploration space point results with the results of a base compilation. This base compile result is generated from the initial settings that were specified in the original Quartus II project files. As DSE traverses all points in the exploration space, all settings that are not explicitly modified by DSE defaults to the base compile setting. For example, if an exploration space point turns on register retiming and does not modify the register packing setting, the register packing setting defaults to the value specified in the base compile.

☞ The base compilation is the original Quartus II project and is restored after DSE traverses all points in the exploration space.

## DSE General Information

You can use DSE in either graphical user interface (GUI) or command-line mode. In either mode, you should run DSE with the Quartus II shell. To run DSE in user interface mode, type `quartus_sh --dse`↵ at a command prompt. To run DSE in command-line mode, type `quartus_sh --dse --nogui <`*options*`>`↵ at a command prompt. The example below lists available command-line options.

**DSE Command Line Options**
Command-line Mode: `quartus_sh --dse -nogui [<`*options*`>]`

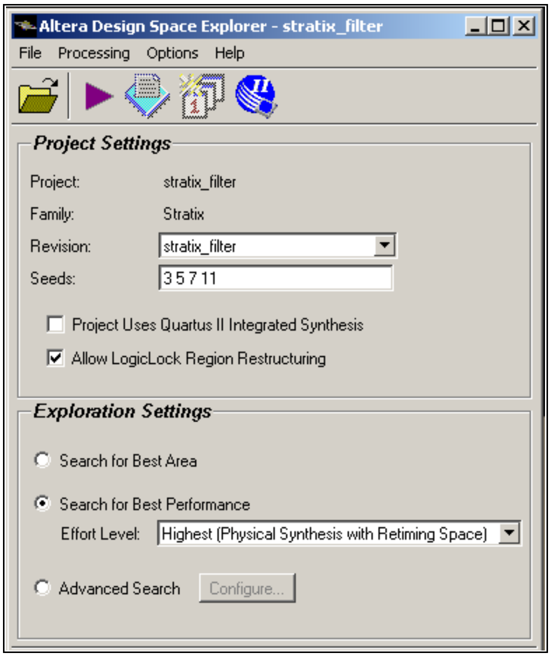
**Options:**
`-project` <*project name*>
`-revision` <*revision name*>
`-seeds` <*seed list*>
`-llr-restructuring`
`-exploration-space` <*space*>
`-optimization-goal` <*goal*>
`-search-method` <*method*>
`-custom-file` <*filename*>
`-stop-after-gain` <*stop-after-gain value*>
`-stop-after-time` <*stop-after-time value*>
`-ignore-failed-base`
`-archive`
`-run-assembler`
`-slaves` <*slave list*>
`-use-lsf`
`-slack-column` <*column name*>
`-help`

The DSE Tcl/Tk script is in the default Quartus II software installation at <*Quartus II install directory*>**\bin\tcl_scripts\dse\dse.tcl** on the PC platform and <*Quartus II install directory*>/<*platform*>/**tcl_scripts** on the Solaris, HP-UX, and Linux platforms.

For more information, type `quartus_sh --help=dse`↵ at the command prompt.

Figure 9–1 shows the DSE user interface. The main user interface is divided into two sections: project settings and exploration settings.

***Figure 9–1. DSE User Interface***



Project: stratix_filter

Family: Stratix

Revision: stratix_filter

Seeds: 3 5 7 11

[ ] Project Uses Quartus II Integrated Synthesis

[✓] Allow LogicLock Region Restructuring

○ Search for Best Area

● Search for Best Performance

Effort Level: Highest (Physical Synthesis with Retiming Space)

○ Advanced Search

## DSE Flow

You can run DSE at any point in the design process. However, Altera recommends that you run DSE very late in your design cycle when you are increasing the performance of the design. The results gained from different combinations of optimization options may not persist over large changes in a design. You can run DSE in signature mode at the midpoint in your design cycle to see the effect of various parameters, such as the register packing logic option.

DSE launches the Quartus II software for every compilation specified in the Exploration Settings option. DSE selectively determines the best settings for your design based upon the **Optimization Goal** selected for the exploration. For example, if the **Optimization Goal** is set to **Optimize for Speed** the Quartus II software tries to achieve all your timing requirements and DSE reports the compile with the smallest slack. Therefore, it is important that you correctly specify all timing requirements in your Quartus II project before performing a design exploration with DSE.

You can change the initial placement configuration used by the Quartus II Fitter by varying the Fitter Seed value. You can enter seeds in the **Seeds** field of the DSE user interface.

☞ When using the Quartus II software, the seed value is set in the **Fitter Settings** page of the **Settings** dialog box (Assignments menu).

Compilation time increases as DSE exploration spaces become more comprehensive. This increase in compilation time comes as a result of running several compilations and comparing the reported slack with the original compilation results.

For typical designs, varying only the seed value results in a 5% $f_{MAX}$ increase. For example, when compiling with three different seeds, one-third of the time $f_{MAX}$ does not improve over the initial compilation, one-third of the time $f_{MAX}$ gets 5% better, and one-third of the time $f_{MAX}$ gets 10% better.

## DSE Support for Altera Device Families

The following device families support all Advanced Exploration Space types:

- Stratix® II
- Stratix
- Stratix GX
- Cyclone™
- MAX® II

The Advanced Exploration Space supports the following device families, as shown in Table 9–1:

- APEX™ 20K
- APEX 20KC
- APEX 20KE
- APEX II
- FLEX® 10K
- FLEX 10KA
- FLEX 10KE

*Table 9–1. Advanced Exploration Space Support for APEX 20K, APEX II, and FLEX 10K Devices*

| | |
|---|---|
| Seed sweep | Area optimization space |
| Extra effort space | Signature fitting effort level |
| Extra effort for Quartus Integrated Synthesis Projects | Custom space |

The following device families support the Synthesis Space type:

- MAX 3000A
- MAX 7000AE
- MAX 7000B
- MAX 7000S

☞ The Synthesis Space type support for the MAX device family is supported only at the command line.

### DSE Exploration

DSE compares all exploration space point results with the results of a base compilation. This base compile result is generated from the initial settings that were specified in the original Quartus II project files. As DSE traverses all points in the exploration space, all settings that are not explicitly modified by DSE defaults to the base compile setting. For example, if an exploration space point turns on register retiming and does not modify the register packing setting, the register packing setting defaults to the value specified in the base compile.

☞ The base compilation is the original Quartus II project and is restored after DSE traverses all points in the exploration space.

## DSE Project Settings

### DSE Project Settings

This section includes information about setting up the working environment for DSE, specifying the project and revision, setting the initial seed, and restructuring LogicLock regions.

The DSE user interface provides two methods to open a Quartus II project for a design exploration. By selecting **Open Project** (File menu) you can browse to your project. The **Open** icon can also be used to open a project for a design exploration.

You can specify the revision to be explored with the **Revision** field in the DSE user interface. The **Revision** field is populated once the Quartus II project has been opened.

☞ If no revisions are created in the Quartus II project, the default revision which is the top-level entity is used. For more information refer to *Quartus II Project Management* chapter in Volume 2 of the *Quartus II Handbook*

The **Seed** field allows you to specify the seed DSE uses in an exploration. The seed value determines the initial placement for your design in a Quartus II compilation.

If your design is written in VHDL or Verilog HDL, turn on the **Project Uses Quartus II Integrated Synthesis** option to allow DSE to explore synthesis options.

The **Allow LogicLock Region Restructuring** option allows DSE to modify the state of any LogicLock regions in your design.

The section below describes the options available in the **Exploration Settings** section of the DSE user interface.

Use the **Exploration Settings** field to select the type of exploration to perform: Search **for Best Area, Search for Best Performance**, or **Advanced Searc**h.

Use the section "Exploration Space" on page 9–8 to select the type of exploration to perform: "Search for Best Area or Performance Options" below, or "Performing an Advanced Search in Design Space Explorer" on page 9–7.

### Search for Best Area or Performance Options

The **Search for Best Performance** option uses a predefined exploration space that targets performance improvements for your design. Depending on the device your design targets, you can select up to four predefined exploration spaces: low (seed sweep), medium (extra effort space), high (physical synthesis space), and highest (physical synthesis with retiming space). As you move from "low" to "highest," the number of options explored by DSE and compilation time increases.

### Advanced Search Option

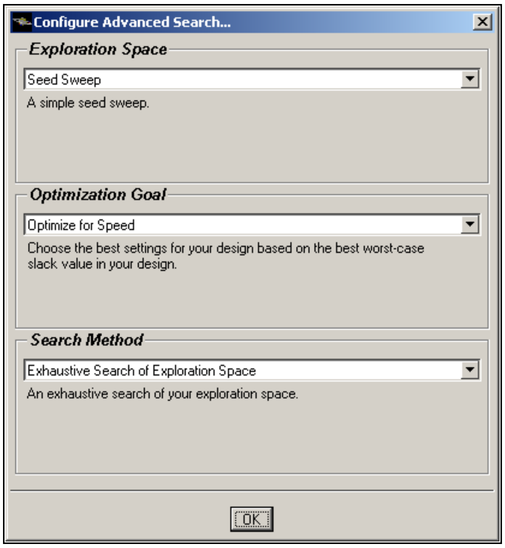
The **Advanced Search** option allows full control over the exploration space, the optimization goal for your design, and the search method used in a design exploration. The section titled "Performing an Advanced Search in Design Space Explorer" on page 9–7 provides detailed information on how to set up and perform an advanced search in DSE.

☞ The advanced search can be used to define equivalent exploration spaces to those found in the **Search for Best Area** and **Search for Best Performance** options.

## Performing an Advanced Search in Design Space Explorer

You must make three exploration settings in the **Advanced Search** dialog box before exploring a design space. These three settings, **Exploration Space**, **Optimization Goal**, and **Search Method**, are described in the following sections. Figure 9–2 shows the **Advanced Search** dialog box.

*Figure 9–2. DSE Advanced Search Dialog Box*



Exploration Space: Seed Sweep
A simple seed sweep.

Optimization Goal: Optimize for Speed
Choose the best settings for your design based on the best worst-case slack value in your design.

Search Method: Exhaustive Search of Exploration Space
An exhaustive search of your exploration space.

## Allow LogicLock Region Restructuring

The Allow LogicLock Region Restructuring option allows DSE to modify LogicLock region properties in your design if any exist. DSE applies the **Soft** property to LogicLock regions to improve timing. Also, DSE may remove LogicLock regions that negatively affect the performance of the design.

## Exploration Space

The exploration space list controls the exploration type that DSE performs on your design. DSE traverses the points in an exploration space, applying the settings to the design and comparing the compilation results to determine the best settings for your design. DSE offers the following predefined exploration spaces:

- Seed sweep
- Extra effort search
- Physical synthesis search
- Retiming search
- Area optimization search
- Custom space
- Signature mode

Not all advanced exploration spaces are available for every device family. See "DSE Support for Altera Device Families" on page 9–5 for advanced exploration space support for various device families.

Compilation time increases proportionally to the breadth of the exploration; the design space increases as more optimization options and parameters are explored.

The **Exploration Space** field is enabled after a project has been opened in DSE.

Turn on **Save exploration space to file** (Option menu) to save an XML file representing the exploration space. The exploration space is written to a file named **<***project name***>.dse** in the project directory. You can modify this file to create a custom exploration space.

For more information on using custom exploration spaces in DSE, see "Creating Custom Spaces for DSE" on page 9–16.

### *Seed Sweep*

The Seed Sweep exploration space leverages the seed sweeping concept and automates the process. Enter the seed values in the **Seeds** field in the DSE user interface. There are no "magic" seeds. Because the variation between seeds is truly random, any integer value is as likely to produce good results. DSE defaults to seeds 3, 5, 7, and 11. The Seed Sweep exploration space does not make changes to your netlist.

The seed field accepts individual seed values, e.g., 2, 3, 4, and 5, or seed ranges, e.g. 2-5.

There is a 1× increase in compilation time for every seed value specified. For example, if you enter five seeds, the compilation time is 5× the initial compilation time.

### *Extra Effort Search*

The Extra Effort Search exploration space adds the **Register Packing** option to the exploration space performed by the **Seed Sweep**. This exploration type also increases the Quartus II Fitter effort during the place-and-route stage. This type of exploration makes no changes to your netlist.

The **Extra Effort Search for Quartus Integrated Synthesis Projects** exploration space includes all the options in **Extra Effort**, and explores various Quartus II integrated synthesis optimization options. The **Extra**

**Effort Search for Quartus Integrated Synthesis Projects** exploration space works only for designs that have been synthesized using the Quartus II integrated synthesis.

For more information on integrated synthesis options, see the *Quartus II Integrated Synthesis* chapter in Volume 1 of the *Quartus II Handbook*.

### *Physical Synthesis Search*

The **Physical Synthesis Search** exploration space adds physical synthesis options such as register retiming and physical synthesis for combinational logic to the options included in the **Extra Effort Search** exploration space. These netlist optimizations move registers in your design. Look-up tables (LUTs) may be modified. The design behavior is not affected by these options.

For more information about physical synthesis, see the *Netlist Optimization & Physical Synthesis* chapter in *Volume 2* of the *Quartus II Handbook*.

The **Physical Synthesis for Quartus Integrated Synthesis Projects** exploration space includes all the options in the **Physical Synthesis** exploration space and explores various Quartus II integrated synthesis optimization options. The **Physical Synthesis for Quartus Integrated Synthesis Projects** exploration space works only for designs that have been synthesized using Quartus II integrated synthesis.

### *Retiming Search*

The **Retiming Search** exploration space includes all the options in the **Physical Synthesis Search** exploration space and explores register retiming. The register retiming may move registers in your design.

The **Retiming Search for Quartus Integrated Synthesis Projects** exploration space includes all the options in **Retiming Search** exploration space, and explores various Quartus II integrated synthesis optimization options. The **Retiming Search for Quartus integrated synthesis Projects** exploration space works only for designs that have been synthesized using the Quartus II integrated synthesis.

### *Area Optimization Search*

The **Area Optimization Search** exploration space explores options that affect logic cell utilization for your design. These options include register packing and Quartus II **Optimization Technique** set to **Area**.

### *Custom Space*

Use the **Custom Space** exploration space to selectively explore the effects of various optimization options on your design. This exploration type gives you complete control over which options are explored and in what mode. In the **Custom Space** mode you can explore all optimization options available in DSE.

For summaries of exploration spaces, refer to Table 9–2.

For more information about using custom exploration spaces with DSE, see "Creating Custom Spaces for DSE" on page 9–16.

***Table 9–2. Summaries of Exploration Spaces*** *Note (1)*

| Search Type | Exploration Spaces | | | | | |
|---|---|---|---|---|---|---|
| | Seed Sweep | Extra Effort | Physical Synthesis | Retiming | Area Optimization | Custom |
| **Analysis & Synthesis Settings** | | | | | | |
| Optimization technique | | | ✓ | ✓ | ✓ | ✓ |
| Perform WYSIWYG resynthesis | | | ✓ | ✓ | ✓ | ✓ |
| Perform gate-level register retiming | | | | ✓ | | ✓ |
| **Fitter Settings** | | | | | | |
| Fitter seed | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Register packing | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Increase PowerFit fitter effort | | ✓ | ✓ | ✓ | | ✓ |
| Perform physical synthesis for combinational logic | | | ✓ | ✓ | | ✓ |
| Perform register retiming | | | | ✓ | | ✓ |

*Note to Table 9–2:*
(1) For exploration spaces that include Quartus Integrated Synthesis, DSE increases the synthesis effort.

### *Signature Mode*

In **Signature** mode, DSE analyzes the $f_{MAX}$, slack, compile time, and area trade-offs of a single parameter. Running the single parameter over multiple seeds, DSE reports the average of these values. With this information you gain a better understanding of how that parameter affects your design. There are four signature mode settings in DSE:

- **Signature**: **Fitting Effort Level**
- **Signature**: **Netlist Optimizations**
- **Signature**: **Fast Fit**
- **Signature**: **Register Packing**

Each setting explores a specific optimization option for your design. For example, in Signature: Register Packing mode, DSE explores the **Auto Packed Registers** logic option with its four settings (**OFF, Normal**, **Minimized Area**, and **Minimize Area with Chains**), and reports the effects of each on your design.

## Optimization Goal

Design metrics are extremely important when exploring the design space of your design whether it be performance, logic utilization, or a combination of both. These metrics allow you to selectively determine which compilation is best, based on the requirements of the design. DSE uses the **Optimization Goal** setting to determine the best compilation results. Here you can specify to DSE which optimization goal you are trying to achieve. Table 9–3 summarizes the four available optimization goals.

*Table 9–3. Optimization Goal Settings*

| Setting | Description |
|---|---|
| Optimize for speed | The exploration space point that contains the best worst-case slack value is selected by DSE as the best run. |
| Optimize for area | The exploration space point that contains the lowest logic cell count is selected by DSE as the best run. |
| Optimize for failing paths | The exploration space point that contains the least amount of failing paths is selected by DSE as the best run. |
| Optimize for negative slack and failing path | The exploration space point that contains the best average negative worst-case slack and lowest number of failing paths is selected by DSE as the best run. |

The optimization goal is independent of the exploration space. An optimization goal that looks for the best performance, bases its best/worst decisions on the exploration space that produces the highest performance and not one with the smallest logic resource utilization.

### Search Method

The **Search Method** setting allows you to control the breadth of the search performed by DSE. DSE provides three search methods: **exhaustive search of exploration space**, **accelerated search of exploration space**, and **distributed search of exploration space**. These three search methods are described in Table 9–4.

*Table 9–4. Search Methods*

| Search Method | Description |
|---|---|
| Exhaustive search of exploration space | Applies all settings available in the exploration space to all seeds specified. This search method yields the optimal settings for your design, but requires the most time. |
| Accelerated search of exploration space | Finds the best exploration space for your design based on the initial seed specified. This sub-space is then applied to all subsequent seeds specified. |
| Distributed search of exploration space | Equivalent to the exhaustive search of exploration space except that this search method uses cluster computing technology to decrease DSE run time. |

## DSE Flow Options

You can control the run time of the design exploration with the options described in this section.

### Continue Exploration Even if Base Compile Fails

DSE continues even if an error occurs during the design compilation. For example, an error occurs in DSE if timing settings are not applied to your design. Turn off this option to make DSE continue with the exploration instead of halting if an error occurs.

### Run Quartus Assembler During Exploration

By default, DSE does not generate programming files for each compilation during exploration. Turn on **Run Quartus Assembler During Exploration** to generate programming files for each compilation.

### Archive All Compiles

Turn on **Archive All Compiles** to create a Quartus Archive File (**.qar**) for each compilation. These archive files are saved to the **dse** directory in the design's working directory.

### Save Exploration Space to File

Turn on **Save Exploration Space to File** to write out a <*project name*>**.dse** file that contains all options explored by DSE. You can use or modify this file to perform a custom exploration.

### Stop Flow After Time

Turn on **Stop Flow After Time** to stop further exploration after a specified number of days, hours, and/or minutes.

☞ Exploration time might exceed the specified value because DSE does not stop in the middle of a compile.

### Stop Flow After Gain

Turn on **Stop Flow After Gain** to stop further exploration after a specified percentage gain.

## DSE Advanced Information

This section covers advanced features that are available in DSE. These features are made available to increase the processing efficiency of design space exploration as well as the further customization of the design space.

### Computer Load Sharing in DSE Using Distributed Exploration Searches

When the **Search Method** is set to Distributed Search of Exploration Space, DSE uses cluster computing technology to decrease exploration search time. DSE uses multiple client computers to compile points in the specified exploration space. Two modes of operation are available when using the **Distributed DSE** option. The first mode uses the Platform LSF grid computing technology to distribute exploration space points to a computing network. In the second mode, DSE acts as a master and distributes exploration space points to client computers. Both modes use an **Exhaustive Search of Exploration Space** search method.

#### *Distributed DSE Using LSF*

The easiest way to use distributed DSE technology is to submit the compilations to a pre-configured LSF cluster at your local site. For more information on LSF software, see **www.platform.com**, or contact your system administrator. Turn on **Use LSF resources** to enable this feature.

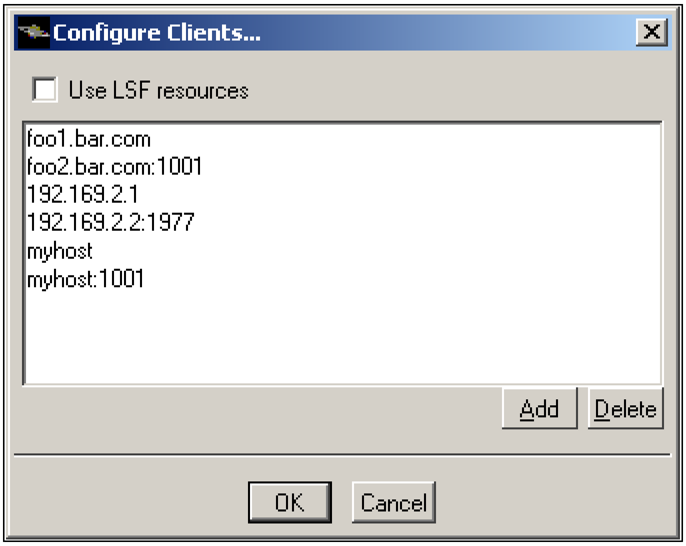
#### *Distributed DSE Using a Quartus II Master Process*

Before DSE can use machines in the local area network to compile points in the exploration space, you need to create Quartus II software slave instances on the machines. In most cases, creating a slave instance on a machine is simple. Enter the following command at a command prompt on a client machine:

```
quartus_sh --qslave ↵
```

Repeating this on several machines creates a cluster of Quartus II software slaves for DSE to use. Once you have created a set of Quartus II software slaves on the network, add the names of each slave machine in **Enter Clients** dialog box. This dialog box appears after selecting **Exhaustive Search of Exploration Space**. Figure 9–3 shows an example of client entries for a distributed search.

*Figure 9–3. Client Entry in DSE*



```
foo1.bar.com
foo2.bar.com:1001
192.169.2.1
192.169.2.2:1977
myhost
myhost:1001
```

At the start of an exploration, DSE assumes the role of a Quartus II software master process and submits points to the slaves on the list to compile. If the list is empty, DSE issues an error and the search stops.

For more information on running and configuring Quartus slaves, type `quartus_sh --help=qslave` ↵ at the command prompt.

The version of the Quartus II software that you use for the Quartus II software slaves must be the same as the version of the Quartus II software you use to run DSE. To see the version of the Quartus II software you are using to run DSE, choose **About DSE** (Help menu). Unexpected results can occur if you mix Quartus II software versions when using the Distributed DSE search feature.

## Creating Custom Spaces for DSE

You can use custom spaces to explore combinations of options that are outside the predefined exploration spaces in the Exploration Space list. An exploration space is defined in an XML file. The following is a description of the tags used to create a custom space for DSE to process.

A custom space is defined by three pairs of major tags, which are:

- `<DESIGNSPACE> and </DESIGNSPACE>`
- `<POINT> and </POINT>`
- `<PARAM> and </PARAM>`

### *DESIGNSPACE Tag*

The `<DESIGNSPACE>` tag defines the start of the exploration space of a custom space. The end tag is `</DESIGNSPACE>`. This tag defines the end of the exploration space. These are both required tags for all custom spaces.

### *POINT Tag*

The POINT tag pair must occur within the DESIGNSPACE tag pair. The `<POINT` *<name>=<stage> enabled="<value>">* tag defines the start of the exploration space point of a custom space. The end tag is *</POINT>*. This tag defines the end of the exploration space point. The POINT also allows you to specify "stage" and whether a particular point is active for a particular DSE exploration.

The "`<stage>`" value in the POINT tag can be one of the following:

- **map**—indicating an Analysis & Synthesis setting change for that particular point
- **fit**—indicating a Fitter setting change for that particular point
- **seed**—indicating a Fitter seed change
- **logiclock**—indicating a LogicLock property change

The *<value>* value in the POINT tag can either be "1," indicating that the exploration space point is active, or "0" for an inactive point. An example of a POINT tag is as follows:

```
<POINT space="map" enabled="1">
...
</POINT>
```

The preceding point indicates a point that has Analysis & Synthesis setting changes and is active.

### *PARAM Tag*

The PARAM tag pair must occur within the POINT tag pair. The `<PARAM name="`*`<parameter>`*`">` tag defines the start of a parameter to be modified for that particular exploration space point. The end tag is `</PARAM>`. This tag defines the end of the parameter. The Analysis & Synthesis settings and the "*<parameter>*" values are shown in Table 9–5. Table 9–6 shows the Fitter settings. An example of a `POINT` tag is shown below:

```
<PARAM name="ADV_NETLIST_OPT_SYNTH_GATE_RETIME"> ON
</PARAM>
```

The point in the example above indicates that the Analysis & Synthesis setting gate-level retiming is turned on for the exploration space point.

***Table 9–5. Analysis & Synthesis Settings*** *Note (1)*

| Analysis & Synthesis Settings | Description | Value |
|---|---|---|
| STRATIX_OPTIMIZATION_TECHNIQUE | Type of optimization technique to use during Analysis & Synthesis stage of a Quartus II software compilation for a Stratix device. | SPEED, AREA, BALANCED |
| CYCLONE_OPTIMIZATION_TECHNIQUE | Type of optimization technique to use during Analysis & Synthesis stage of a Quartus II software compilation for a Cyclone device. | SPEED, AREA, BALANCED |
| ADV_NETLIST_OPT_SYNTH_GATE_RETIME | Gate-level register retiming | OFF, ON |
| ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP | WYSIWYG primitive resynthesis | OFF, ON |
| DSE_SYNTH_EXTRA_EFFORT_MODE | Controls the Quartus II software synthesis effort. | MODE_1, MODE_2, MODE_3 |

***Note to Table 9–5:***
(1) Not all Analysis & Synthesis settings are available for all device families.

***Table 9–6. Fitter Settings (Part 1 of 2)*** *Note (1)*

| Fitter Settings | Description | Value |
|---|---|---|
| AUTO_PACKED_REGISTERS_STRATIX | Register packing for Stratix devices | NORMAL, MINIMIZE_AREA, MINIMIZE_AREA_WITH_CHAINS |
| AUTO_PACKED_REG_CYCLONE | Register packing for Cyclone devices | OFF, MINIMIZE_AREA, MINIMIZE_AREA_WITH_CHAINS |
| INNER_NUM | PowerFit fitter effort level | {integer value} |

*Table 9–6. Fitter Settings (Part 2 of 2) Note (1)*

| Fitter Settings | Description | Value |
|---|---|---|
| PHYSICAL_SYNTHESIS_COMBO_LOGIC | Physical synthesis for combinational logic | OFF, ON |
| PHYSICAL_SYNTHESIS_REGISTER_ DUPLICATION | Physical synthesis for register duplication | OFF, ON |
| PHYSICAL_SYNTHESIS_REGISTER_ RETIMING | Physical synthesis for register retiming | OFF, ON |

*Note to Table 9–6:*
(1) Not all Fitter settings are available for all device families.

The custom space example below shows a simple custom exploration space that performs a seed sweep with various Analysis & Synthesis settings and Fitter settings.

**Simple Custom Space**
```
<DESIGNSPACE>
<POINT space="map">
</POINT>
<POINT space="fit">
</POINT>
<POINT space="map" enabled="1">
  <PARAM name="CYCLONE_OPTIMIZATION_TECHNIQUE">SPEED</PARAM>
  <PARAM name="ADV_NETLIST_OPT_SYNTH_GATE_RETIME">ON</PARAM>
  <PARAM name="ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP">ON</PARAM>
  <PARAM name="STRATIX_OPTIMIZATION_TECHNIQUE">SPEED</PARAM>
 </POINT>
<POINT space="fit" enabled="1">
  <PARAM name="PHYSICAL_SYNTHESIS_REGISTER_RETIMING">ON</PARAM>
  <PARAM name="PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION">
   ON</PARAM>
  <PARAM name="AUTO_PACKED_REG_CYCLONE">OFF</PARAM>
  <PARAM name="AUTO_PACKED_REGISTERS_STRATIX">OFF</PARAM>
  <PARAM name="SEED">3</PARAM>
  <PARAM name="PHYSICAL_SYNTHESIS_COMBO_LOGIC">ON</PARAM>
 </POINT>
</DESIGNSPACE>
```

The example, "Simple Custom Space", defines a custom exploration space that has four points. The first two points in the space are special points: an empty "map" point and an empty "fit" point. DSE expects the first two points in any custom exploration space to be an empty map point and an empty fit point, as seen in this example.

Following the empty map and fit points are one map point and one fit point that change the Quartus II Fitter settings. The map point sets the optimization technique to speed, turns on gate level retiming, and turns on the WYSIWYG resynthesis. For the fit point, register retiming, register duplication, and physical synthesis for combinational logic is turned on; register packing is turned off; and a seed value of three is used.

The example, "Custom Space XML Schema", contains an XML schema that describes the XML format for custom exploration space files. You can use an advanced XML editor or XML verification tool to validate any custom exploration files against this schema.

### Custom Space XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="DESIGNSPACE">
        <xs:annotation>
            <xs:documentation>The root element of a design space
description</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="POINT"/>
            </xs:sequence>
            <xs:attribute name="project" type="xs:string" use="optional"/>
            <xs:attribute name="revision" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="POINT">
        <xs:annotation>
            <xs:documentation>A point in the design space</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="PARAM"/>
            </xs:sequence>
            <xs:attribute name="space" type="xs:string" use="required"/>
            <xs:attribute name="enabled" type="xs:boolean" use="optional" default="1"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="PARAM" type="xs:string" nillable="0">
        <xs:annotation>
            <xs:documentation>A single Quartus II software setting</xs:documentation>
        </xs:annotation>
    </xs:element>
</xs:schema>
```

## Conclusion

DSE automates the process of finding the best set of options for your design. It explores the design space of your design by applying various optimization techniques and analyzing the results to shorten your design's timing closure cycle.

# 10. LogicLock Design Methodology

qii52009-2.1

## Introduction

Available exclusively in the Altera® Quartus® II software, the LogicLock™ block-based design flow enables you to design, optimize, and lock down your design one module at a time. With the LogicLock methodology, you can independently create and implement each logic module into a hierarchical or team-based design. With this method, you can preserve the performance of each module during system integration. Additionally, you can reuse logic modules in other designs, further leveraging resources and shortening design cycles.

The Quartus II software version 4.1 supports the LogicLock block-based design flow for the following devices:

- Stratix® II, Stratix, Stratix GX, MAX II®, and Cyclone™
- APEX® and APEX II
- Excalibur™
- Mercury™ (Mercury devices only support locked and fixed regions)

This chapter assumes that you are familiar with the basic functionality of the Quartus II software. See the "LogicLock Module" in the Quartus II Help for instructions on using the LogicLock feature in a sample design.

For more information and guidelines for hierarchical design flow, see the *Hierarchical Block-Based & Team-Based Design Flows* chapter in Volume 1 of the *Quartus II Handbook*.

### Improving Design Performance

You can use the LogicLock flow for performance optimization and preservation. You can use the LogicLock flow to place modules, entities, or any group of logic into regions in a device's floorplan. Because LogicLock assignments are generally hierarchical, you have more control over the placement and performance of modules and groups of modules.

In addition to hierarchical blocks, you can use the LogicLock feature on individual nodes, e.g., to make a wildcard path-based LogicLock assignment on a critical path. This technique is useful if the critical path spans multiple design blocks.

☞ Although LogicLock constraints can improve performance, they can also degrade performance if they are not applied correctly.

### Preserving Module Performance

The LogicLock design flow allows you to lock the placement and routing of nodes in a region of a device so that the placement of logic in the region remains constant. The Quartus II software then places the LogicLock region into the top-level design with these constraints.

## Designing with the LogicLock Feature

To design with the LogicLock feature, create a LogicLock region in a supported device and then assign logic to the region. The LogicLock region can contain any contiguous, rectangular block of device resources. After you have optimized the logic placed within the boundaries of a region to achieve the required performance, back-annotate the region's contents to lock the logic placement and routing. Then, when you integrate the region with the rest of the design, the performance is preserved.

This section explains the basics of designing with the LogicLock feature, including:

- Creating LogicLock Regions
- Floorplan Editor View
- LogicLock Region Properties
- Hierarchical (Parent/Child) LogicLock Regions
- Assigning LogicLock Region Content
- Tcl Scripts
- Quartus II Block-Based Design Flow
- Additional Quartus II LogicLock Design Features

### Creating LogicLock Regions

There are four ways to create a LogicLock region:

- In the **LogicLock Regions** window (Assignments menu)
- Using the **Create New Region** button in the Timing Closure Floorplan
- Using the **Compilation Hierarchy** window
- Using a Tool Command Language (Tcl) script

#### *LogicLock Regions Window*

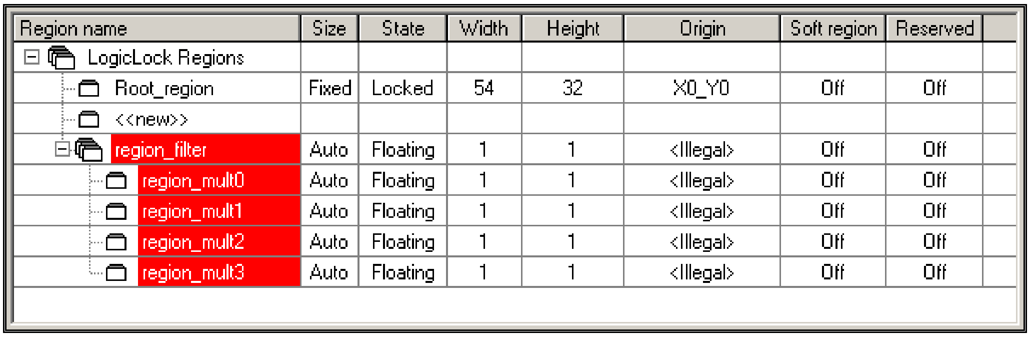
The LogicLock **window** is comprised of the **LogicLock Regions** window and **LogicLock Region Properties** dialog box. Use the **LogicLock Regions** window to create LogicLock regions and assign nodes and entities to them. The dialog box provides a summary of all LogicLock regions in your design. You can modify a LogicLock region's size, state, width, height, and origin as well as whether the region is Soft or Reserved, in this window. When the region is back-annotated, the placement of the nodes within a region are relative to the region's origin, and the region's node placement during subsequent compilations is maintained.

For Stratix, Stratix GX, Stratix II, MAX II, and Cyclone devices, the LogicLock region's origin is located at the bottom-left corner of the region. For all other supported devices, the origin is located at the top-left corner of the region.

The **LogicLock Regions** window displays any LogicLock regions that contain illegal assignments in red as shown in Figure 10–1. If you make illegal assignments, you can use the **Repair Branch** command to reset the assignments for the currently selected region and its descendents to legal default values.

For more information on the **Repair Branch** command, see the "Repair Branch" on page 10–22.

***Figure 10–1. LogicLock Regions Window***



| Region name | Size | State | Width | Height | Origin | Soft region | Reserved |
|---|---|---|---|---|---|---|---|
| LogicLock Regions | | | | | | | |
| Root_region | Fixed | Locked | 54 | 32 | X0_Y0 | Off | Off |
| <<new>> | | | | | | | |
| region_filter | Auto | Floating | 1 | 1 | <Illegal> | Off | Off |
| region_mult0 | Auto | Floating | 1 | 1 | <Illegal> | Off | Off |
| region_mult1 | Auto | Floating | 1 | 1 | <Illegal> | Off | Off |
| region_mult2 | Auto | Floating | 1 | 1 | <Illegal> | Off | Off |
| region_mult3 | Auto | Floating | 1 | 1 | <Illegal> | Off | Off |

You can customize the **LogicLock Regions** window by dragging and dropping the various columns. The columns can also be hidden.

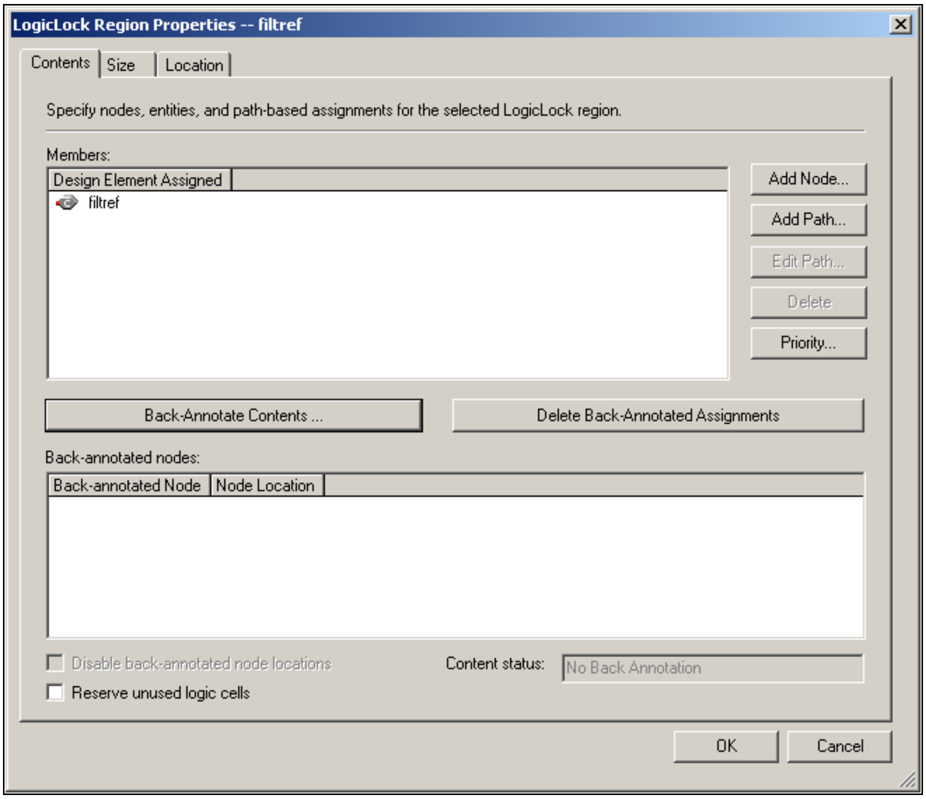
The Soft and Reserved columns are not shown by default.

For designs targeting Stratix, Stratix GX, Stratix II, MAX II, and Cyclone devices, the Quartus II software automatically creates a LogicLock region that encompasses the entire device. This default region is labelled `Root_region`, and it is effectively locked and fixed.

Use the **LogicLock Region Properties** dialog box to obtain detailed information about your LogicLock region, such as which entities and nodes are assigned to your region and what resources are required (see Figure 10–2). The LogicLock Region Properties dialog box shows the properties of the current selected regions.

☞ The **LogicLock Region Properties** dialog box can be opened by double-clicking any region in the **LogicLock Regions** window or right-clicking the region and selecting **Properties**.

***Figure 10–2. LogicLock Region Properties Dialog Box***


LogicLock Region Properties -- filtref
Contents
Size
Location
Add Node...
Add Path...
Edit Path...
Delete
Priority...
Back-Annotate Contents ...
Delete Back-Annotated Assignments
OK
Cancel

Specify nodes, entities, and path-based assignments for the selected LogicLock region.

Members:

| Design Element Assigned |
|---|
| filtref |

Back-annotated nodes:

| Back-annotated Node | Node Location |
|---|---|

☐ Disable back-annotated node locations

☐ Reserve unused logic cells

Content status: No Back Annotation

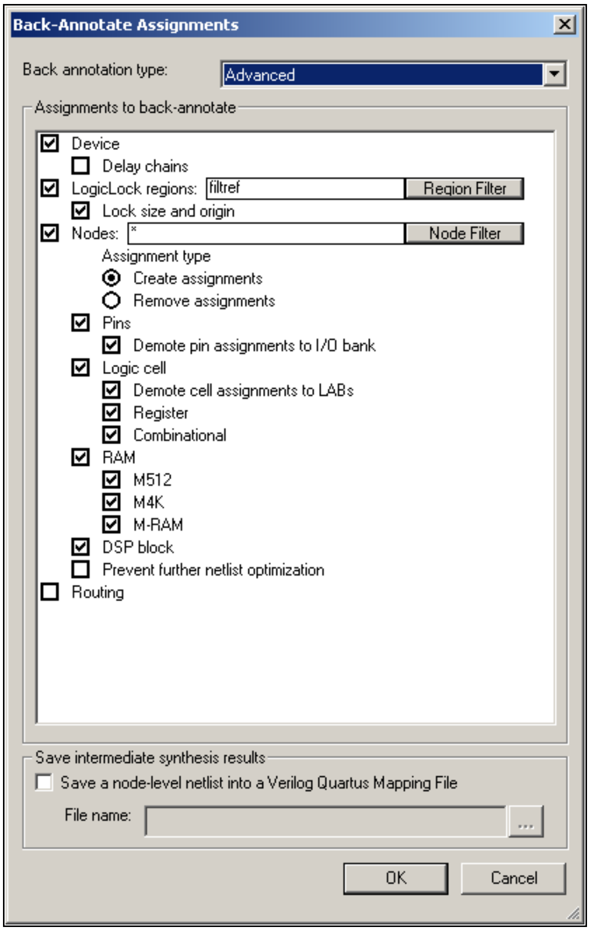
To back-annotate the contents of your LogicLock regions, perform these steps:

1. In the **LogicLock Region Properties** dialog box, click **Back-Annotate Contents**.

2. Select the contents you wish to back-annotate using the **Back-Annotate Assignments** (**Advanced type**) (Assignment menu) dialog box (Figure 10–3)

3. Click OK.

*Figure 10–3. Back-Annotate Assignments Dialog Box (Advanced Type)*



Back-Annotate Assignments

Back annotation type: Advanced

Assignments to back-annotate

- [✓] Device
  - [ ] Delay chains
- [✓] LogicLock regions: filtref
  - [✓] Lock size and origin
- [✓] Nodes: *
  - Assignment type
    - [✓ Create assignments]
    - Remove assignments
  - [✓] Pins
    - [✓] Demote pin assignments to I/O bank
  - [✓] Logic cell
    - [✓] Demote cell assignments to LABs
    - [✓] Register
    - [✓] Combinational
  - [✓] RAM
    - [✓] M512
    - [✓] M4K
    - [✓] M-RAM
  - [✓] DSP block
  - [ ] Prevent further netlist optimization
- [ ] Routing

Save intermediate synthesis results

- [ ] Save a node-level netlist into a Verilog Quartus Mapping File

File name:

You can also back-annotate routing within LogicLock regions for increased region portability. For more information on back-annotating routing information, see "Back-Annotating Routing Information" on page 10–33.

When you back-annotate a region's contents and demote all cell assignments, all of the design element nodes appear under **Back-annotated nodes** with an assignment to a device resource (e.g., logic array block [LAB], M512, M4K, M-RAM, digital signal processing [DSP] block, etc.) under **Node Location**. Each node's location is the placement of the node after the last compilation. If the origin of the region changes, the node's location changes to maintain the same relative placement. This relative placement preserves the performance of the nodes. If cell assignments are demoted, then the nodes are assigned to LABs rather than directly to logic cells.

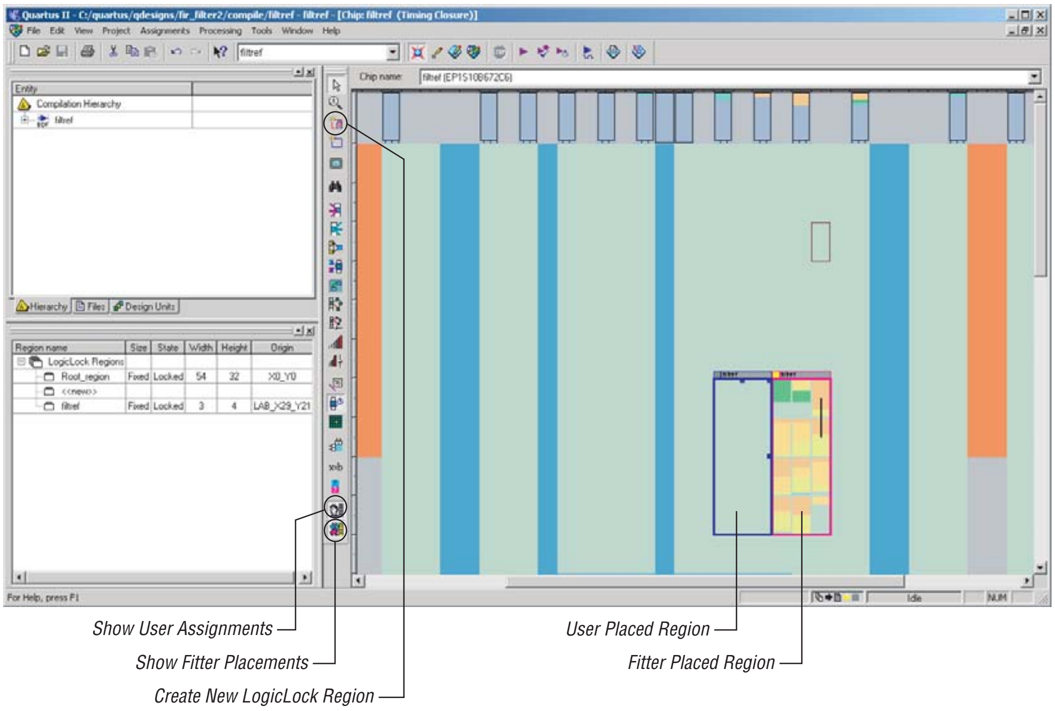
### *Timing Closure Floorplan Editor*

The Timing Closure Floorplan Editor has toolbar buttons with which you can manipulate LogicLock regions, as shown in Figure 10–4. You can use the **Create New Region** button to draw LogicLock regions in the device floorplan.

☞ The Timing Closure Floorplan Editor displays LogicLock regions when **Show User Assignments** or **Show Fitter Placements** is selected. The type of region determines its appearance in the floorplan.

The Timing Closure Floorplan Editor differentiates between user assignments and fitter placements. When the **Show User Assignments** option is turned **on** in the Timing Closure Floorplan, current assignments made to a LogicLock region are visible. When the **Fitter Placement** option is turned **on**, you can see the properties of the LogicLock region after the last compilation. User-assigned LogicLock regions appear in the Floorplan Editor with a dark blue LogicLock border. Fitter-placed regions appear in the Floorplan Editor with a magenta border.

*Figure 10–4. Floorplan Editor Toolbar Buttons*

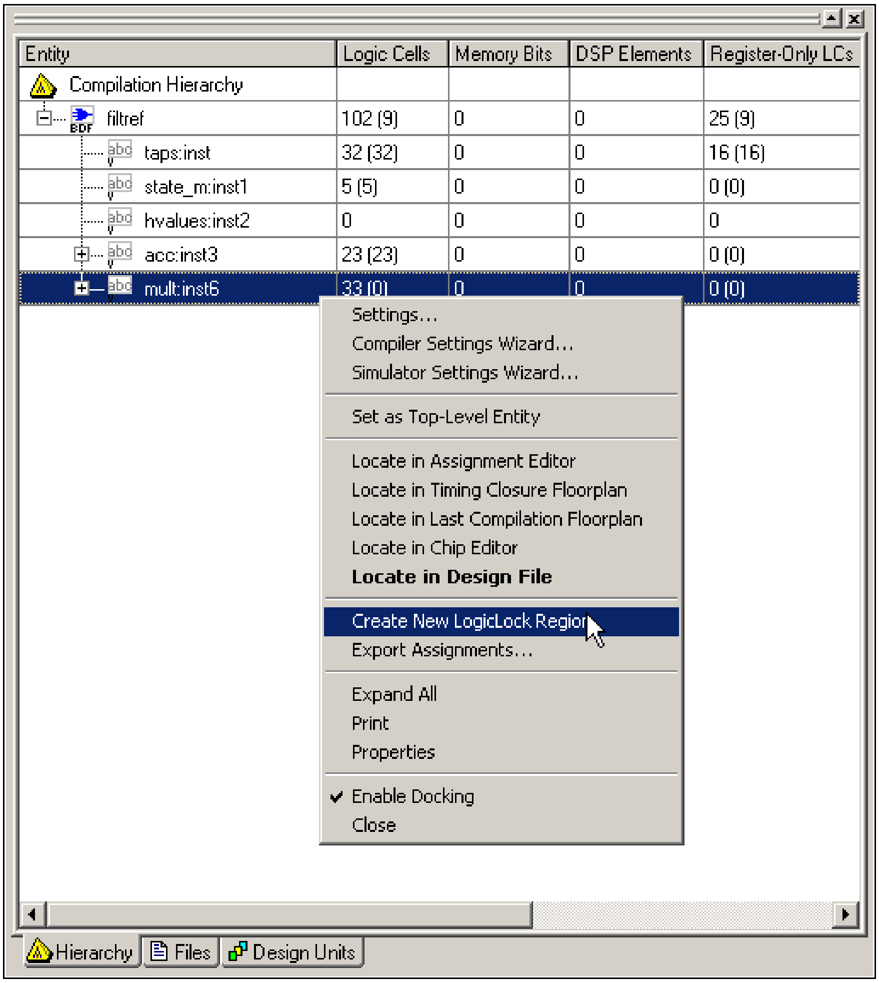

Quartus II - C:/quartus/qdesigns/fir_filter2/compile/filtref - filtref - [Chip: filtref (Timing Closure)]
File Edit View Project Assignments Processing Tools Window Help
filtref
Chip name: filtref (EP1S10B672C6)
Entity
Compilation Hierarchy
filtref
Hierarchy
Files
Design Units
For Help, press F1
Idle
NUM
Show User Assignments
Show Fitter Placements
Create New LogicLock Region
User Placed Region
Fitter Placed Region

| Region name | Size | State | Width | Height | Origin |
|---|---|---|---|---|---|
| LogicLock Regions | | | | | |
| Root_region | Fixed | Locked | 54 | 32 | X0_Y0 |
| <<new>> | | | | | |
| filtref | Fixed | Locked | 3 | 4 | LAB_X29_Y21 |

### *Hierarchy Window*

After you have performed either a full compilation or Analysis & Elaboration on the design, the Quartus II software displays the hierarchy of the design in the Compilation Hierarchy window. With the hierarchy of the design fully expanded, as shown in Figure 10–5, you can conveniently create a LogicLock region by right clicking on any design entity in the design and selecting **Create New LogicLock Region** in the right button pop-up menu.

*Figure 10–5. Hierarchy Window Used to Create LogicLock Regions*



| Entity | Logic Cells | Memory Bits | DSP Elements | Register-Only LCs |
|---|---|---|---|---|
| Compilation Hierarchy | | | | |
| filtref | 102 (9) | 0 | 0 | 25 (9) |
| taps:inst | 32 (32) | 0 | 0 | 16 (16) |
| state_m:inst1 | 5 (5) | 0 | 0 | 0 (0) |
| hvalues:inst2 | 0 | 0 | 0 | 0 |
| acc:inst3 | 23 (23) | 0 | 0 | 0 (0) |
| mult:inst6 | 33 (0) | 0 | 0 | 0 (0) |

Settings...
Compiler Settings Wizard...
Simulator Settings Wizard...
Set as Top-Level Entity
Locate in Assignment Editor
Locate in Timing Closure Floorplan
Locate in Last Compilation Floorplan
Locate in Chip Editor
Locate in Design File
Create New LogicLock Region
Export Assignments...
Expand All
Print
Properties
✓ Enable Docking
Close
Hierarchy
Files
Design Units

### *Tcl Scripts*

You can create LogicLock regions and assign nodes to them with Tcl commands that you can run from the Tcl Console or at the command prompt.

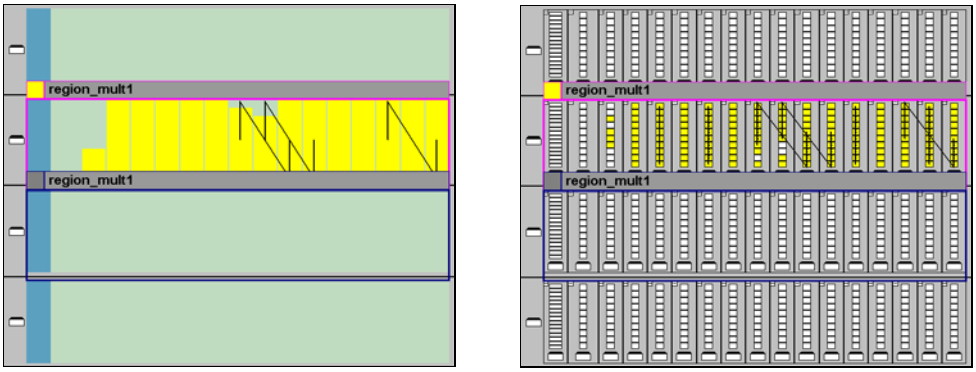
For more information, refer to the *Scripting Support* chapter in Volume 2 of the *Quartus II Handbook*.

## Floorplan Editor View

The **Timing Closure Floorplan** view provides you with current and last compilation assignments on one screen. You can display device resources in either of two views: the Field View and the Interior Cells View, as shown in Figure 10–6. The Field View provides an uncluttered view of the device floorplan in which all device resources such as ESBs and MegaLAB™ blocks are outlined. The interior Cells View provides a detailed view of device resources this includes individual Logic Elements within a MegaLAB and device pins.

*Figure 10–6. Floorplan Editor—Timing Closure*

*(Field View)* *(Interior Cells View)*



## LogicLock Region Properties

A LogicLock region is defined by its size (height and width) and location (where the region is located on the device). You can specify the size and/or location of a region, or the Quartus II software can generate them automatically. The Quartus II software bases the size and location of the region on its contents and the module's timing requirements. Table 10–1 describes the options for creating LogicLock regions.

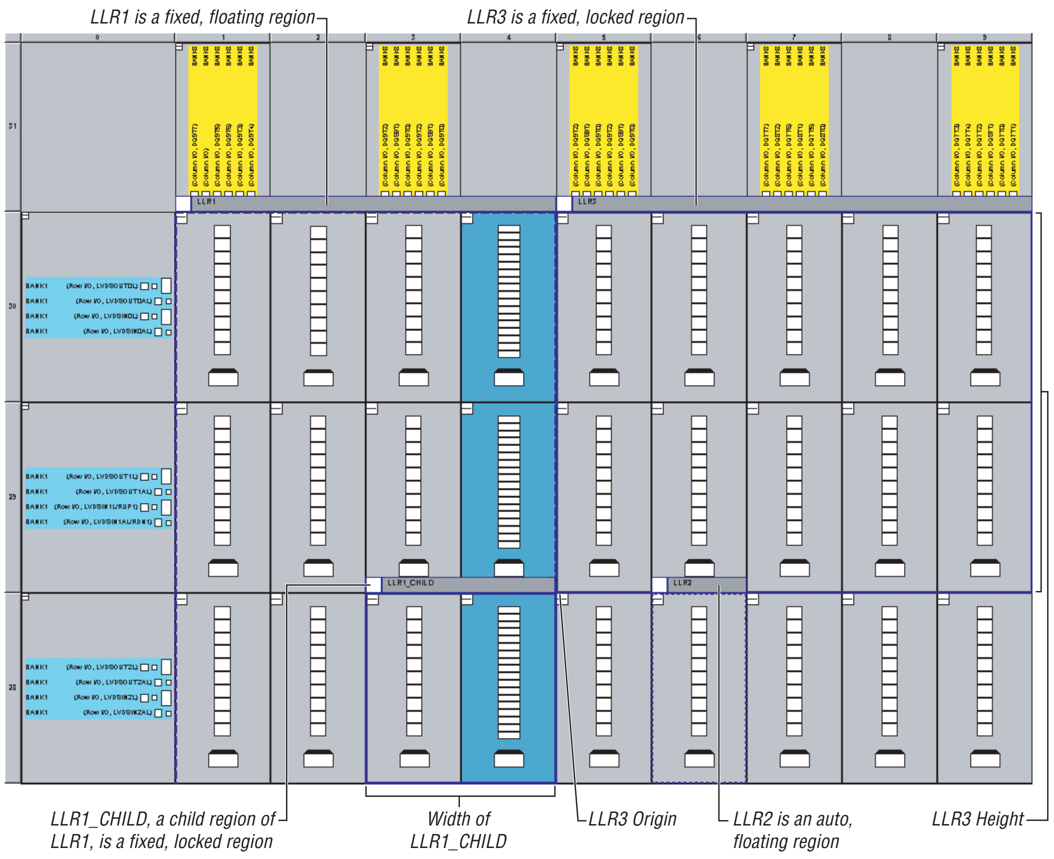
*Table 10–1. Types of LogicLock Regions*

| Properties | Values | Behavior |
|---|---|---|
| State | Floating (default), Locked | Floating regions allow the Quartus II software to determine the region's location on the device. Locked regions represent user-defined locations of a region and are illustrated with a solid boundary in the graphical floorplans. A locked region must have a fixed size. |
| Size | Auto (default), Fixed | Auto-sized regions allow the Quartus II software to determine the appropriate size of a region given its contents. Fixed regions have a user-defined shape and size. |
| Reserved | Off (default), On | The reserved property allows you to define whether you can use the resources within a region for entities that are not assigned to the region. If the reserved property is on, only items assigned to the region can be placed within its boundaries. |
| Enforcement | Hard (default), Soft | Soft regions give more deference to timing constraints, and allow some entities to leave a region if it improves the performance of the overall design. Hard regions do not allow contents to be placed outside of the boundaries of the region. |
| Origin | Any Floorplan Location | The origin defines the top-left corner of the LogicLock region's placement on the floorplan. For Stratix, Stratix II, Stratix GX, MAX II, and Cyclone the origin is located in the lower-left hand corner. The origin is located in the upper-left corner for other families. |

☞ The Quartus II software cannot automatically define a region's size if the location is locked. Therefore, if you want to specify the exact location of the region, you must also specify the size. Mercury devices only support locked and fixed regions.

The floorplan excerpt in Figure 10–7 shows the LogicLock region properties for a design implemented in a Stratix device.

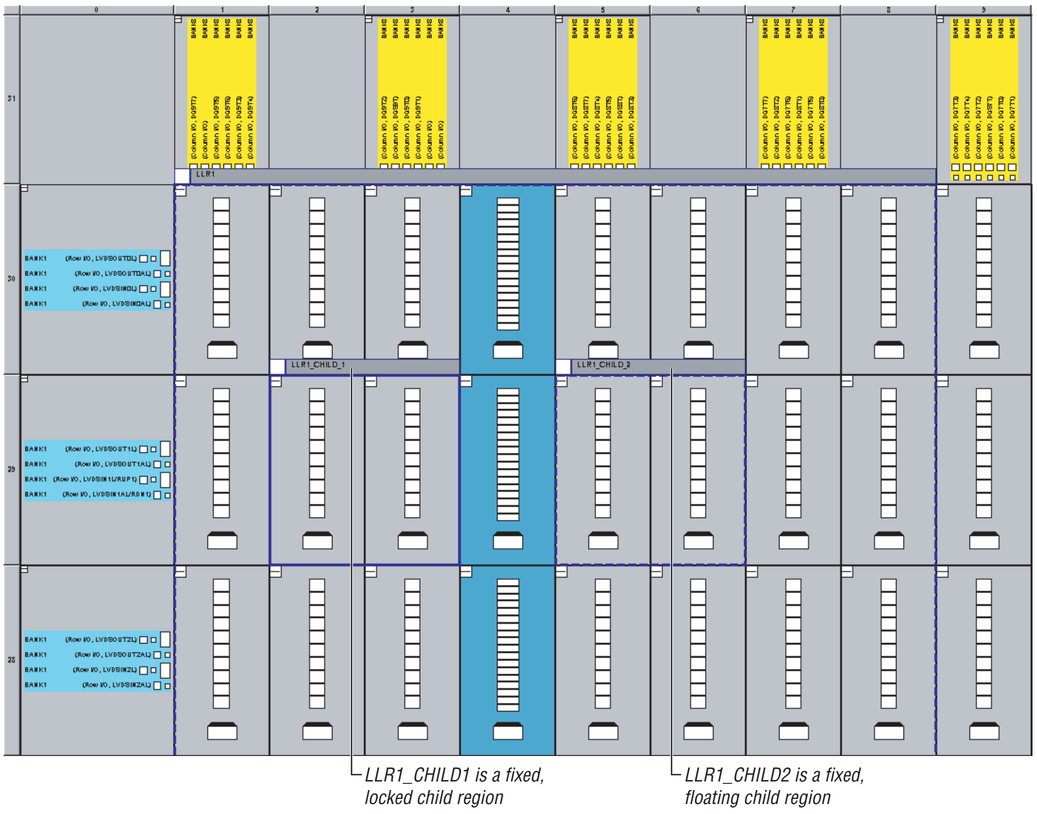
*Figure 10–7. LogicLock Region Properties*


LLR1 is a fixed, floating region
LLR3 is a fixed, locked region
LLR1
LLR3
LLR1_CHILD
LLR2
LLR1_CHILD, a child region of LLR1, is a fixed, locked region
Width of LLR1_CHILD
LLR3 Origin
LLR2 is an auto, floating region
LLR3 Height

## Hierarchical (Parent and/or Child) LogicLock Regions

With the LogicLock design flow, you can define a hierarchy for a group of regions by declaring parent and/or child regions. The Quartus II software places a child region completely within the boundaries of its parent region, allowing you to further constrain module locations. Additionally, Parent and child regions allow you to further improve a module's performance by constraining the nodes in the module's critical path. Figure 10–8 shows an example child region within a parent region, including labels for a locked location and floating location in a Stratix device.

*Figure 10–8. Child Region within a Parent Region*


LLR1
LLR1_CHILD_1
LLR1_CHILD_2
LLR1_CHILD1 is a fixed, locked child region
LLR1_CHILD2 is a fixed, floating child region

☞ The LogicLock region hierarchy does not have to be the same as the design hierarchy.

A child region's location can float within its parent or remain locked relative to its parent's origin, while a locked parent region's location is locked relative to the device. If the child's location is locked and the parent's location is changed, the child's origin changes but maintains the same placement relative to the origin of its parent. Either you or the Quartus II software can determine a child region's size; however, it must fit entirely within the parent region.

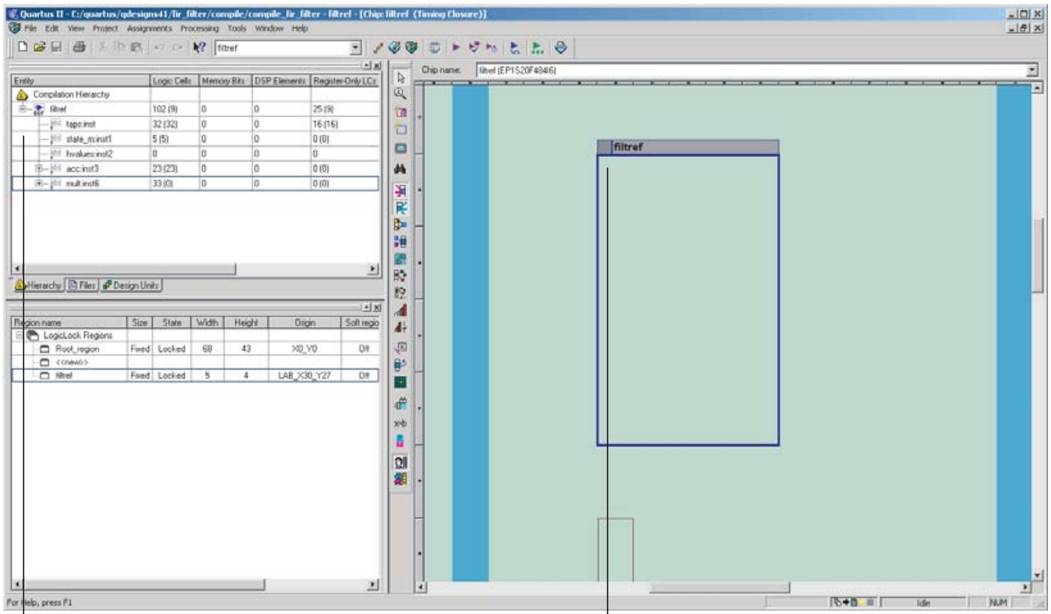
## Assigning LogicLock Region Content

Once you have defined a LogicLock region, you must assign resources to it using the **Timing Closure Floorplan**, the **LogicLock Regions** dialog box, the Assignment Editor, or Tcl scripts with the Quartus II Tcl Console or the `quartus_sh` executable.

### Using Drag & Drop to Place Logic

You can drag selected logic from the Compilation Hierarchy window, Node Finder, or a schematic design file and drop it into the Timing Closure Floorplan or the **LogicLock Regions** dialog box. Figure 10–9 shows logic that has been dragged from the Compilation Hierarchy window dropped into a LogicLock region in the **Timing Closure Floorplan**.

*Figure 10–9. Drag & Drop Logic in the Current Assignments Floorplan*

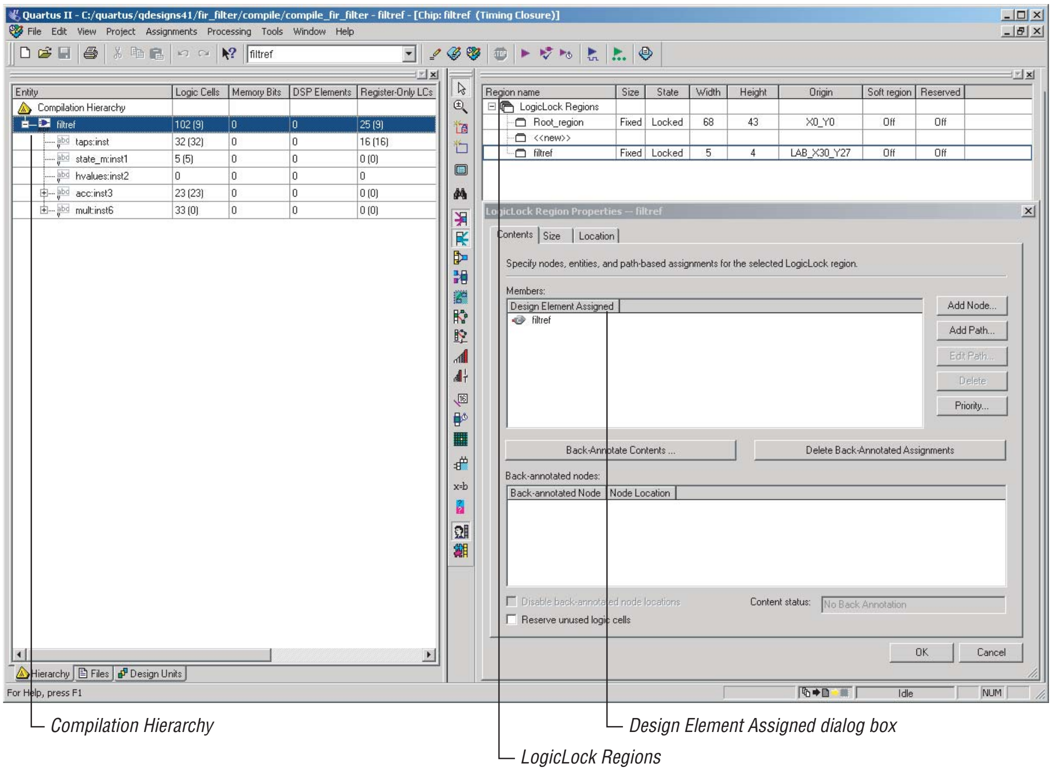

Quartus II - C:/quartus/qdesigns41/fir_filter/compile/compile_fir_filter - filtref - [Chip: filtref (Timing Closure)]
File Edit View Project Assignments Processing Tools Window Help
filtref
Chip name: filtref (EP1S20F484I6)
filtref
Hierarchy Files Design Units
For Help, press F1
Idle
NUM
Compilation Hierarchy
LogicLock Region

| Entity | Logic Cells | Memory Bits | DSP Elements | Register-Only LCs |
|---|---|---|---|---|
| Compilation Hierarchy | | | | |
| filtref | 102 (9) | 0 | 0 | 25 (9) |
| taps:inst | 32 (32) | 0 | 0 | 16 (16) |
| state_m:inst1 | 5 (5) | 0 | 0 | 0 (0) |
| hvalues:inst2 | 0 | 0 | 0 | 0 |
| acc:inst3 | 23 (23) | 0 | 0 | 0 (0) |
| mult:inst6 | 33 (0) | 0 | 0 | 0 (0) |

| Region name | Size | State | Width | Height | Origin | Soft regio |
|---|---|---|---|---|---|---|
| LogicLock Regions | | | | | | |
| Root_region | Fixed | Locked | 68 | 43 | X0_Y0 | Off |
| <<new>> | | | | | | |
| filtref | Fixed | Locked | 5 | 4 | LAB_X30_Y27 | Off |

Figure 10–10 shows logic that has been dragged from the Compilation Hierarchy window and dropped into the **LogicLock Regions** dialog box. Logic can also be dropped into the **Design Element Assigned** dialog box.

*Figure 10–10. Drag & Drop Logic into the LogicLock Regions Dialog Box*


Quartus II - C:/quartus/qdesigns41/fir_filter/compile/compile_fir_filter - filtref - [Chip: filtref (Timing Closure)]
File Edit View Project Assignments Processing Tools Window Help
Hierarchy
Files
Design Units
For Help, press F1
Idle
NUM
Compilation Hierarchy
Design Element Assigned dialog box
LogicLock Regions

| Entity | Logic Cells | Memory Bits | DSP Elements | Register-Only LCs |
|---|---|---|---|---|
| Compilation Hierarchy | | | | |
| filtref | 102 (9) | 0 | 0 | 25 (9) |
| taps:inst | 32 (32) | 0 | 0 | 16 (16) |
| state_m:inst1 | 5 (5) | 0 | 0 | 0 (0) |
| hvalues:inst2 | 0 | 0 | 0 | 0 |
| acc:inst3 | 23 (23) | 0 | 0 | 0 (0) |
| mult:inst6 | 33 (0) | 0 | 0 | 0 (0) |

| Region name | Size | State | Width | Height | Origin | Soft region | Reserved |
|---|---|---|---|---|---|---|---|
| LogicLock Regions | | | | | | | |
| Root_region | Fixed | Locked | 68 | 43 | X0_Y0 | Off | Off |
| <<new>> | | | | | | | |
| filtref | Fixed | Locked | 5 | 4 | LAB_X30_Y27 | Off | Off |

LogicLock Region Properties -- filtref

Contents | Size | Location

Specify nodes, entities, and path-based assignments for the selected LogicLock region.

Members:

| Design Element Assigned |
|---|
| filtref |

Add Node... | Add Path... | Edit Path... | Delete | Priority...

Back-Annotate Contents ... | Delete Back-Annotated Assignments

Back-annotated nodes:

| Back-annotated Node | Node Location |
|---|---|

☐ Disable back-annotated node locations

☐ Reserve unused logic cells

Content status: No Back Annotation

OK | Cancel

☞ You must manually assign pins to a LogicLock region. The Quartus II software does not include pins automatically when you assign an entity to a region. The software only obeys pin assignments to locked regions that border the periphery of the device. For Stratix, Stratix II, MAX II, and Cyclone devices, the locked regions must enclose the I/O pins as resources.

### *Using the Assignment Editor to Place Logic*

You can also use the Assignment Editor to assign entities and nodes to a LogicLock region (see Figure 10–11). To assign content to a LogicLock region with the Assignment Editor, perform the following steps:

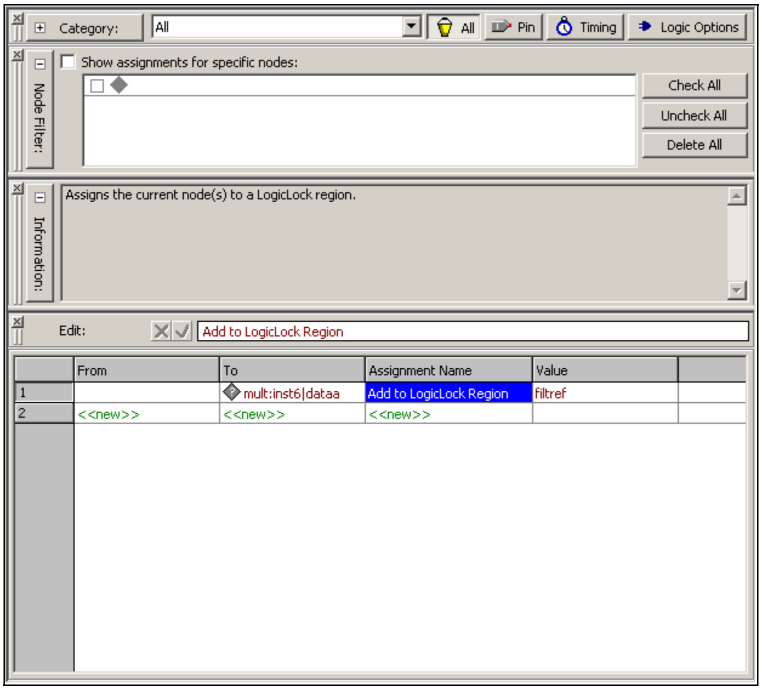
1. Under **Assignment Name**, select **Add to LogicLock Region.**

2. Under **Value,** specify your LogicLock region name.

3. Under **To,** specify either nodes or entities that are to reside in the LogicLock region.

The nodes or entities are then assigned to the selected LogicLock region.

*Figure 10–11. Assignment Editor*



Assigns the current node(s) to a LogicLock region.

Edit: Add to LogicLock Region

| | From | To | Assignment Name | Value |
|---|---|---|---|---|
| 1 | | mult:inst6\|dataa | Add to LogicLock Region | filtref |
| 2 | <<new>> | <<new>> | <<new>> | |

## Tcl Scripts

You can create LogicLock regions and assign nodes to them with Tcl commands that you can run from the Tcl Console or at the command prompt. The Tcl command `set_logiclock` is used to create or change the attributes of LogicLock regions.

For more information on creating and using LogicLock regions and contents, see the *Command Line* and *Tcl API* topics in the Quartus II online Help.

## Quartus II Block-Based Design Flow

When using the LogicLock design flow, it is recommended that you divide the design into modules. Then, perform the following steps in the Quartus II software for each module:

1. Synthesize the module using the Quartus II software or another synthesis tool.

2. Optimize the module in the Quartus II software.

3. Export the module and the LogicLock constraints.

4. Import all modules and LogicLock constraints into the top-level project.

5. Compile and verify the top-level design.

### *Synthesize the Module*

You can synthesize the module in the Quartus II software or any Altera-supported third-party synthesis tool, e.g., the Synplify®, LeonardoSpectrum™, or FPGA Compiler II software. The software synthesizes each module into an atom netlist, which represents the logic in terms of Altera primitives for the target Altera device.

In the atom netlist, the nodes are fixed as Altera primitives; the node names do not change if the atom netlist does not change. If a node name does change, any placement information made to that node is invalid and ignored. Third-party tools generate atom netlists as EDIF Input Files (**.edf**) or Verilog Quartus Mapping Files (**.vqm**).

### *Optimize the Module*

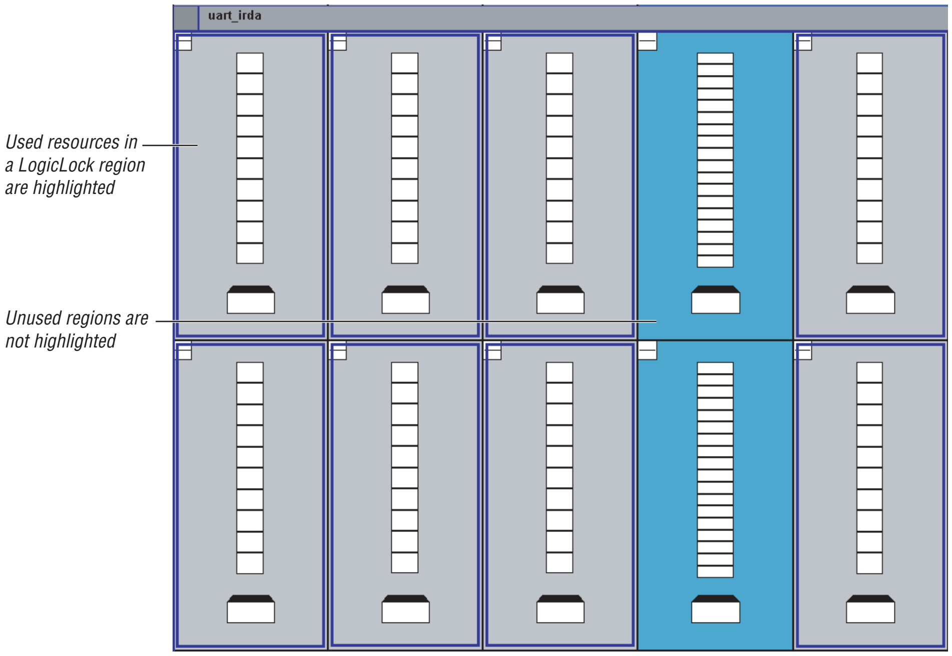
Before optimizing a module in the Quartus II software, create a project with the module as the top-level entity. You must assign the module to a single (or multiple) LogicLock region. See the "Constraint Priority" on page 10–30 for information on the precedence of the LogicLock region and other constraint settings.

After you have optimized the module so that it meets timing requirements, lock down the placement of nodes in a LogicLock region by back-annotating the contents of the region. To make relative location assignments, you must fix the node names. Fixed node names require an atom netlist so that the assignments for each node remain valid. The node placement is fixed relative to the LogicLock region for the module.

For the Quartus II software to achieve optimal placement, you should make timing assignments for all clock signals in the design, e.g., $\mathbf{t}_{SU}$, $\mathbf{t}_{CO}$, and $\mathbf{t}_{PD}$.

To facilitate the LogicLock design flow, the **Timing Closure Floorplan** highlights resources that have back-annotated LogicLock regions. Figure 10–12 shows a back-annotated LogicLock region in the **Timing Closure Floorplan**.

*Figure 10–12. Back-Annotated LogicLock Region*


uart_irda
Used resources in a LogicLock region are highlighted
Unused regions are not highlighted

## *Export the Module*

This section describes how to export a module's constraints to a format that can be imported by a top-level design. To be exported, a module requires design information as an atom netlist (VQM or EDF), placement information stored in a Quartus Settings File (**.qsf**), and routing information stored in a Routing Constraint File (**.rcf**).

### Atom Netlist Design Information

The atom netlist contains design information that fully describes the module's logic in terms of an Altera device architecture. If the design was synthesized using a third-party tool and then brought into the Quartus II software, an atom netlist already exists and the node names are fixed. You do not need to generate another atom netlist. However, if you use any Synthesis Netlist Optimizations, or Physical Synthesis Optimizations, you must generate a Quartus II VQM. because the original atom netlist may have changed as a result of these optimizations.

It is recommended that you turn on the option **Prevent further netlist optimization** option when back-annotating a region with the **Synthesis Netlist Optimizations** and/or **Physical Synthesis Optimization** options turned on. This sets the **Netlist Optimizations to Never Allow** option on all nodes in the region, avoiding the possibility of a node name change when the region is imported into the top-level design.

If you synthesized the design as a VHDL Design File (**.vhd**), Verilog Design File (**.v**), Text Design File (**.tdf**), or a Block Design File (**.bdf**) in the Quartus II software, you must also create an atom netlist to fix the node names. During compilation, the Quartus II software creates a VQM File in the **atom_netlists** subdirectory in the project directory.

If the atom netlist is from a third-party synthesis tools and the design has a black-boxed library of parameterized modules (LPM) functions or Altera megafunctions, you must generate a Quartus II VQM File for the black-boxed modules.

For instructions on creating an atom netlist in the Quartus II software, see *Saving Synthesis Results for an Entity to a Verilog Quartus Mapping File* in Quartus II Help.

When you export LogicLock regions, the Quartus II software defaults to exporting your entire design's LogicLock region assignments. However, you can export a sub-entity of the compilation hierarchy and all of its relevant regions. This can be accomplished by right-clicking the entity in the Compilation Hierarchy and selecting **Export Assignments** from the right button pop-up menu.

### Placement Information

The QSF contains the module's LogicLock constraint information, including clock settings, pin assignments, and relative placement information for back-annotated regions. To maintain performance, you must back-annotate the module.

**Routing Information**
The RCF contains the module's LogicLock routing information. To maintain performance, you must back-annotate the module.

For instructions on exporting a LogicLock region assignment in the Quartus II software, see *Exporting LogicLock Region Assignments and Other Entity Assignments* in Quartus II Help.

### *Import the Module*

You can specify which QSF is used for a specific instance or entity with the **LogicLock Import File Name** option in the Assignment Editor. Therefore, you can specify different LogicLock region constraints for each instance of an entity and import them into the top-level design. You can also specify an RCF file with the **LogicLock Routing Constraints File Name** option in the Assignment Editor.

When importing LogicLock regions into the top-level design, you must specify the QSF and RCF for the modules in the project. If the design instantiates a module multiple times, the Quartus II software applies the LogicLock regions multiple times.

Before importing LogicLock regions, you must perform **Analysis & Elaboration**, or compile the top-level design, so that the Quartus II software is aware of all instances of the lower-level modules.

The following sections describe how to specify a QSF for a module and how to import the LogicLock assignments into the top-level design.

**Specify the QSF and Atom Netlist**
To specify the QSF and atom netlist to import, perform the following steps:

1. Specify an atom netlist for the module that you are importing by either copying the atom netlist to your current working directory or choosing **Add/Remove Project Files** (Project menu) and browsing to the file.

2. Perform **Analysis & Elaboration**.

3. Expand the design hierarchy on the **Compilation Hierarchy** tab of the Project Navigator by clicking the **+** icon next to the top-level entity.

4. Right-click on the entity and choose **Locate** in the **Assignment Editor**.

5. Under **Assignment Name**, choose **LogicLock Import File Name**.

6. Under **Value**, type the name and relative path to the QSF, or click **Browse** and navigate to the QSF in the **Select File** dialog box.

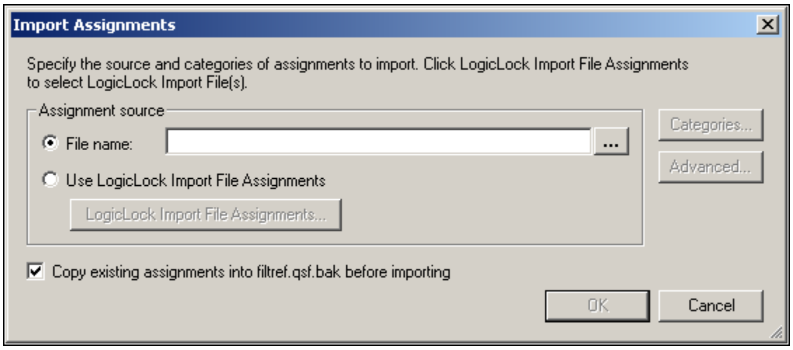
Repeat steps 3 through 5 for all entities that require a specific QSF.

You can follow the same procedure for specifying a QSF when specifying an RCF. Instead of selecting **LogicLock Import File Name**, select **LogicLock Back Routing Constraints File Name.**

**Import the Assignments**

To import the assignments, choose **Import Assignments** (Assignments menu). Figure 10–13 shows the **Import Assignment** dialog box.

*Figure 10–13. Import Assignments Dialog Box*



Import Assignments

Specify the source and categories of assignments to import. Click LogicLock Import File Assignments to select LogicLock Import File(s).

Assignment source

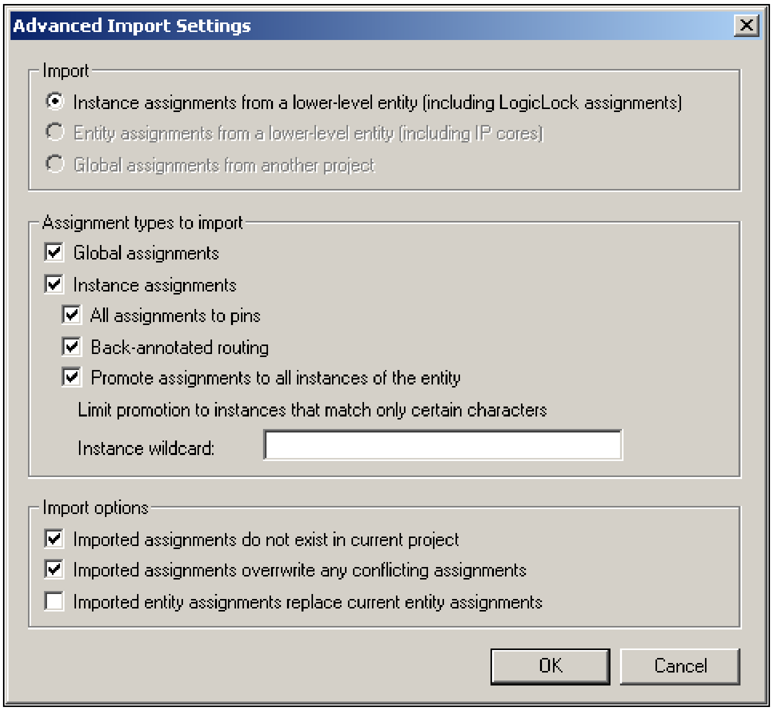
[✓ File name:]

Use LogicLock Import File Assignments

[✓ Copy existing assignments into filtref.qsf.bak before importing]

There are a number of options available in the Advanced Import Assignments dialog box that you can use to control the import of your LogicLock regions, as shown in Figure 10–14.

*Figure 10–14. Advanced Import Settings Dialog Box*


Advanced Import Settings

Import
- [●] Instance assignments from a lower-level entity (including LogicLock assignments)
- [ ] Entity assignments from a lower-level entity (including IP cores)
- [ ] Global assignments from another project

Assignment types to import
- [✓] Global assignments
- [✓] Instance assignments
  - [✓] All assignments to pins
  - [✓] Back-annotated routing
  - [✓] Promote assignments to all instances of the entity
  - Limit promotion to instances that match only certain characters
  - Instance wildcard:

Import options
- [✓] Imported assignments do not exist in current project
- [✓] Imported assignments overwrite any conflicting assignments
- [ ] Imported entity assignments replace current entity assignments

The Quartus II software converts all imported parent or top-level regions (that do not contain back-annotated routing information) to floating regions to prevent spurious no-fit errors. This allows the Quartus II software to move LogicLock regions to areas on the device with free resources. Child regions are locked or floating relative to their parent region's origin as specified in the modules' original LogicLock constraints.

☞ If you want to lock a LogicLock region to a location, you can manually lock down the region in the **LogicLock Regions** dialog box or the **Timing Closure Floorplan**.

Each imported LogicLock region has a name that corresponds to the original LogicLock region name combined with the instance name in the form of <*instance name*>|<*original LogicLock region name*>. For example, if a LogicLock region for a module is named `LLR_0` and the instance name for the module is `Filter:inst1`, then he LogicLock region name in the top-level design is `Filter:inst1|LLR_0`.

### *Compile & Verify the Top-Level Design*

After importing all modules, you can compile and verify the top-level design. The compilation report shows whether system timing requirements have been met.

## Additional Quartus II LogicLock Design Features

To complement the **LogicLock Regions** dialog box and Device Floorplan view, the Quartus II software has additional features to help you design with the LogicLock feature.

### *Tooltips*

When you move the mouse so that the pointer is over a LogicLock region name in the Hierarchy window of the Project Navigator or **LogicLock Regions** dialog box, or over the top bar of the LogicLock region in the **Timing Closure Floorplan**, the Quartus II software displays a tooltip with information about the properties of the LogicLock region.

☞ Placing the mouse over **Fitter Placed LogicLock Regions** displays the maximum routing delay within the LogicLock region. You must first turn on the **Show Critical Paths** (see "Show Critical Paths" on page 10–24) command before the delay information becomes available.

### *Repair Branch*

When you retarget your design to either a larger or smaller device, there is a chance that your LogicLock regions no longer contain valid values for location or size in the new device, resulting in an illegal LogicLock region. The Quartus II software identifies illegal LogicLock regions in the **LogicLock Regions** dialog box by coloring the name of the region containing the error red.

To correct the illegal LogicLock region, use the **Repair Branch** command. Right click the desired LogicLock region's name and select **Repair Branch** (Right button pop-up menu).

If more then one illegal LogicLock region exists, you can repair all regions by right clicking the first line in the **LogicLock** window that contains the text **LogicLock Regions** and selecting **Repair Branch**.

### *Reserve LogicLock Region*

The Quartus II software honors all entity and node assignments to LogicLock regions. Occasionally, entities and nodes do not occupy an entire region, which leaves some of the region's resources unoccupied. To increase the region's resource utilization and performance, the Quartus II software's default behavior fills the unoccupied resources with other nodes and entities that have not been assigned to any other region. You can prevent this behavior by turning on **Reserve unused logic cells** on the **Contents** tab of the **LogicLock Region Properties** dialog box. When this option is turned on, your LogicLock region only contains the entities and nodes that you have specifically assigned to your LogicLock region.

In a team-based design environment, this option is extremely helpful in device floorplanning. When this option is turned on, each team can be assigned a portion of the device floorplan where placement and optimization of each submodule occurs. Device resources can be distributed to every module without affecting the performance of other modules.

### *Prevent Assignment to LogicLock Regions Option*

Turning on the **Prevent Assignment to LogicLock Regions** options exclude any arbitrary entity or node from being a member of any LogicLock region. However, it does not prevent the entity or node from entering into LogicLock regions. The fitter places the entity or node anywhere on the device as if no regions exist. For example, if an entire module is assigned to a LogicLock region, when this option is turned on, you can exclude a specific sub-entity or node from the region.

☞ The **Prevent Assignment to LogicLock Regions** option for a given entity or node is found in the **Assignment Editor** under **Assignment Name**.

### *LogicLock Regions Connectivity*

The Timing Closure Floorplan Editor allows you to see connections between various LogicLock regions that exist within a design. The connection between the regions is drawn as a single line between the LogicLock regions. The thickness of this line is proportional to the number of connections between the regions.

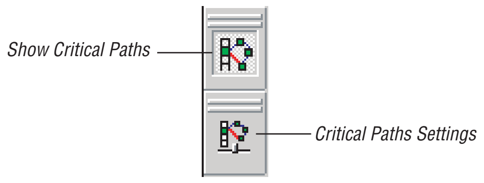
### *Rubber Banding*

When the **Rubber Banding** option is turned on, the Quartus II software shows existing connections between LogicLock regions and nodes during movement of LogicLock regions within the Floorplan Editor.

### *Show Critical Paths*

You can display the critical paths within a LogicLock region by turning the **Show Critical Paths** option On. This option is used in conjunction with the **Critical Paths Settings** option that allows you to display either one or more of the following paths: pin-to-pin, pin-to-register, register-to-pin, or register-to-register, as shown in Figure 10–15.

***Figure 10–15. Show Critical Paths & Critical Paths Settings***



### *Show Connection Count*

You can determine the number of connections between LogicLock regions by turning the **Show Connection Count** option On.

### *Analysis & Synthesis Resource Utilization by Entity*

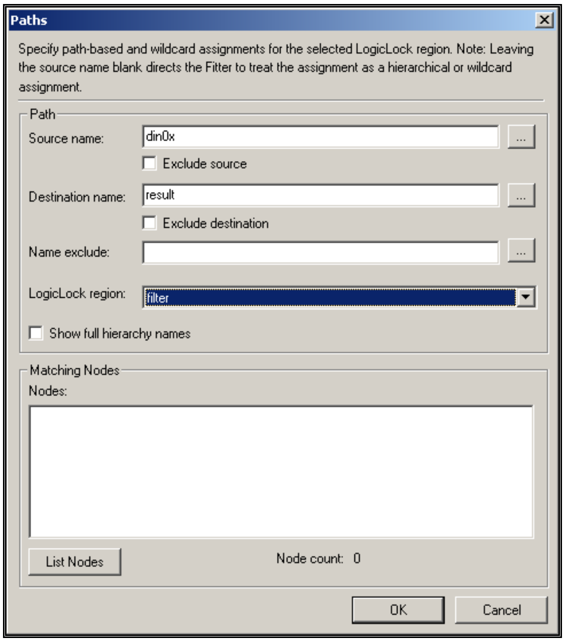
The Compilation Report contains an **Analysis & Synthesis Resource Utilization by Entity** section, which reports accurate resource usage statistics, including entity-level information. This feature is useful when manually creating LogicLock regions.

### *Path-Based Assignments*

You can assign paths to LogicLock regions based on source and destination nodes, allowing; you to easily group critical design nodes into a LogicLock region. The path's source and destination nodes must be a valid register-to-register path, meaning that the source and destination nodes must be registers. Figure 10–16 shows the **Path-Based Assignment** dialog box.

☞ Both "*" and "?" wildcard characters are allowed for both the source and destination nodes. When creating path-based assignments you can have the option of excluding certain nodes with the Name exclude field in the **Path** dialog box.

*Figure 10–16. Path-Based Assignment Dialog Box*


Paths

Specify path-based and wildcard assignments for the selected LogicLock region. Note: Leaving the source name blank directs the Fitter to treat the assignment as a hierarchical or wildcard assignment.

Path

Source name: din0x

☐ Exclude source

Destination name: result

☐ Exclude destination

Name exclude:

LogicLock region: filter

☐ Show full hierarchy names

Matching Nodes

Nodes:

Node count: 0

The **Path-Based Assignment** dialog box is launched from the **Contents Tab** of the **LogicLock Regions** dialog box.

You can also use the Quartus II Timing Analysis Report to create path-based assignments. To create path-based assignments, follow these steps:

1. Expand the Timing Analyzer section in the Compilation Report.
2. Select any of the clocks in the section that is labelled "Clock Setup:<*clock name*>"
3. Locate a path that you would like to assign to a LogicLock region. Drag this path from the Report window and drop it in the <<new>> section of the LogicLock Region window.

This operation creates a path-based assignment from the source register to the destination register as shown in the Timing Analysis Report.

### *Quartus II Revisions Feature*

When you create, modify, or import LogicLock regions into a top-level design, you may need to experiment with different configurations to achieve your desired results. The Quartus II software provides the Revisions feature that allows for a convenient way to organize the same project with different settings until an optimum configuration is found.

Use the **Revisions** dialog box (Project menu) to create and set revisions. Revision can be based on the current design or any previously created revisions. A description can also be entered for each revision created. This is a convenient way to organize the placement constraints created for your LogicLock regions.

### *LogicLock Assignment Precedence*

Conflicts might arise during the assignment of entities and nodes to LogicLock regions. For example, an entire top-level entity might be assigned to one region and a node within this top-level entity assigned to another region. To resolve conflicting assignments, the Quartus II software maintains an order of precedence for LogicLock assignments. The Quartus II software's order of precedence is as follows from highest to lowest:

1. Exact node-level assignments

2. Path-based and wildcard assignments

3. Hierarchical assignments

However, conflicts might also occur within path-based and wildcard assignments. Path-based and wildcard assignment conflicts arise when one path-based or wildcard assignment contradicts another path-based or wildcard assignment. For example, a path-based assignment is made containing a node labeled `X` and assigned to LogicLock region `PATH_REGION`. A second assignment is made using wildcard assignment X* with node X being placed into region `WILDCARD_REGION`. As a result of these two assignments, node `X` is assigned to two regions: `PATH_REGION` and `WILDCARD_REGION`.

To resolve this type of conflict, the Quartus II software keeps the order in which the assignments were made and treats the last assignment created with the highest priority.

☞ Open the **Priority** dialog box by selecting **Priority** on the **Contents** tab of the **LogicLock properties** dialog box. You can change the priority of path-based and wildcard assignments by using the **Up** or **Down** buttons in the **Priority** dialog box. To prioritize assignments between regions, you must select multiple **LogicLock** regions. Once the regions have been selected, you can open the **Priority** dialog box from the **LogicLock Properties** window.

### *LogicLock Regions versus Soft LogicLock Regions*

Normally all nodes assigned to a particular LogicLock region always resides within the boundaries of that region. Soft LogicLock regions can enhance design performance by removing the fixed rectangular boundaries of LogicLock regions. When you assign a LogicLock region as being "Soft," Quartus II software attempts to place as many nodes assigned to the region as close together as possible, and has the added flexibility of moving nodes outside of the soft region to meet your design's performance requirement. This allows the Quartus II Fitter greater flexibility in placing nodes in the device to meet your performance requirements.

When you assign nodes to a soft LogicLock region, they can be placed anywhere in the device, but if the soft region is the child of a region, the nodes will not be assigned outside the boundaries of the parent region. If a non-soft parent does not exist (in a design targeting a Stratix, Stratix GX, Stratix II, MAX II, or Cyclone device), the region floats within the Root_region, i.e., the boundaries of the device. You can turn On the **Soft Region** option on the **Location** tab of the **LogicLock Region Properties** dialog box.

☞ Soft regions can have an arbitrary hierarchy that allows any combination of parent and child to be a soft region. The *Reserved* option is not compatible with soft regions.

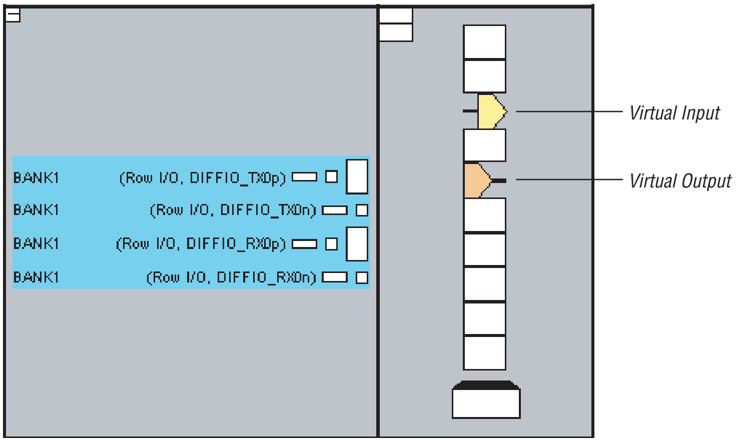
Soft LogicLock regions cannot be back-annotated because the Quartus II software may have placed nodes outside of the LogicLock region resulting in undefinable location assignments relative to the region's origin and size.

Soft LogicLock regions are available for all device families that support floating LogicLock regions.

### Virtual Pins

When you compile a design in the Quartus II software, all I/O ports are directly mapped to pins on the targeted device. This I/O port mapping may create problems for a modular/hierarchical design because lower-level modules may have more I/O ports than pins available on the targeted device, or the I/O ports may not directly feed a device pin, but may drive other internal nodes. The Quartus II software supports virtual pins to accommodate this situation. Virtual pin assignments tell the Quartus II software which I/O ports of the design module become internal nodes in the top-level design. These assignments prevent the number of I/O ports in the lower-level module from exceeding the total number of available device pins. Every I/O port that is designated as a virtual pin gets mapped to an `LCELL` register in the device. Figure 10–17 shows the virtual input and output pins in the **Floorplan Editor**.

*Figure 10–17. Virtual I/O Pins in the Quartus II Floorplan Editor*


BANK1 (Row I/O, DIFFIO_TXDp)
BANK1 (Row I/O, DIFFIO_TXDn)
BANK1 (Row I/O, DIFFIO_RXDp)
BANK1 (Row I/O, DIFFIO_RXDn)
Virtual Input
Virtual Output

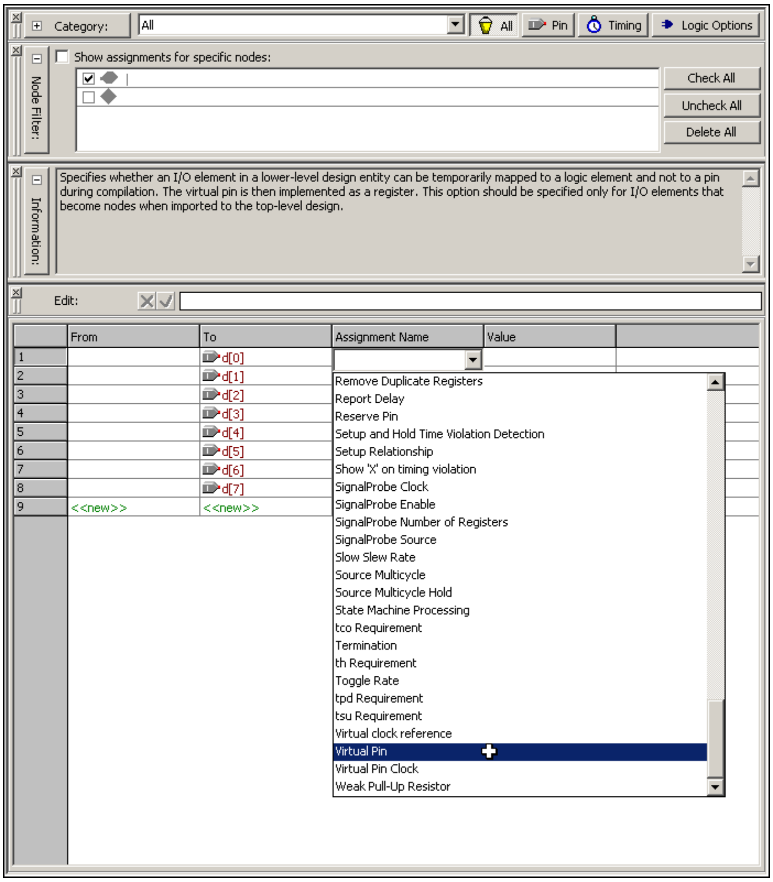
Bidirectional, registered I/O pins, and I/O pins with output enable signals cannot be virtual pins. All virtual pins must map to device I/O pins in the top-level design.

In the top-level design, these virtual pins are connected to an internal node in another module. Making assignments to virtual pins allow you to place them within the same location or region on the device as the corresponding internal node would exist in the top-level module. This feature also has the added benefit of providing accurate timing information during lower-level module optimization.

To accommodate designs with multiple clock domains, you can specify individual clock signals by turning to Virtual Pin Clock option on for each virtual pin.

Virtual pin and virtual pin clock assignments are made through the **Assignment Editor**. Figure 10–18 shows assigning virtual pins using the **Assignment Editor**.

*Figure 10–18. Using the Assignment Editor to Assign Virtual Pin*



Category: All

Specifies whether an I/O element in a lower-level design entity can be temporarily mapped to a logic element and not to a pin during compilation. The virtual pin is then implemented as a register. This option should be specified only for I/O elements that become nodes when imported to the top-level design.

|  | From | To | Assignment Name | Value |
|---|---|---|---|---|
| 1 |  | d[0] |  |  |
| 2 |  | d[1] |  |  |
| 3 |  | d[2] |  |  |
| 4 |  | d[3] |  |  |
| 5 |  | d[4] |  |  |
| 6 |  | d[5] |  |  |
| 7 |  | d[6] |  |  |
| 8 |  | d[7] |  |  |
| 9 | <<new>> | <<new>> |  |  |

Setting **Filter Type** to **Pins: Virtual** allows the Node Finder to display all assigned virtual pins in the design.

## LogicLock Restrictions

This section discusses restrictions that you should consider when using the LogicLock design flow, including:

- Constraint priority
- Placing LogicLock regions
- Placing memory, pins and other device features into LogicLock regions

### Constraint Priority

During the design process, it is often necessary to place restrictions on nodes or entities in the design. Often, these restrictions conflict with the node or entity assignments for a LogicLock region. To avoid conflicts, you should consider the order of precedence given to constraints by the Quartus II software during fitting. The following assignments have priority over LogicLock region assignments:

- Assignments to device resources and location assignments
- Fast input register and fast output register assignments
- Local clock assignments for Stratix devices
- Custom region assignments
- I/O standard assignments

The Quartus II software removes nodes and entities from LogicLock regions if any of these constraints are applied to them.

### Placing LogicLock Regions

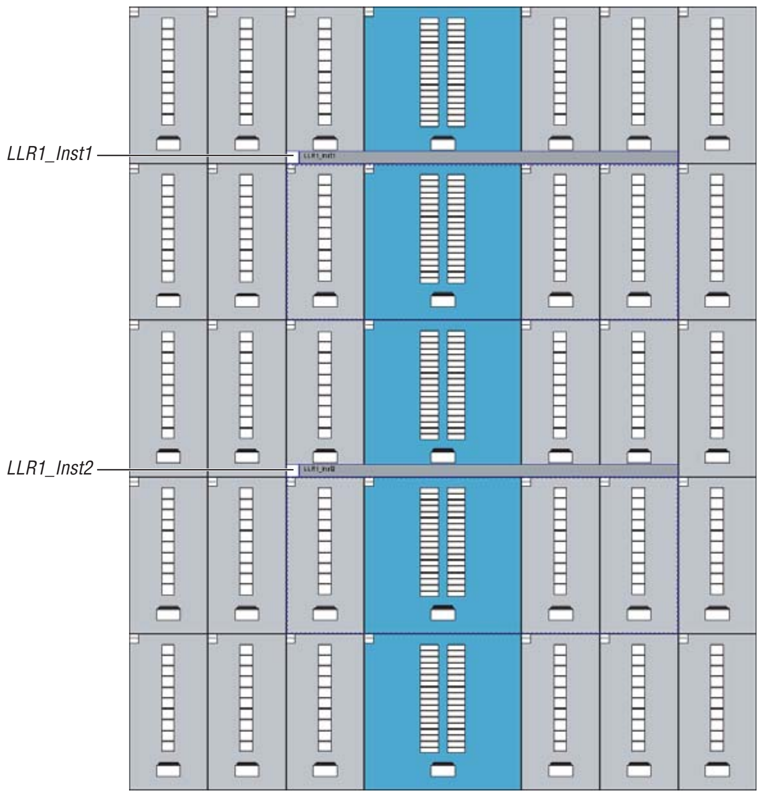
A fixed region must contain all of the resources required for the module. Although the Quartus II software can automatically place and size LogicLock regions to meet resource and timing requirements, you can manually place and size regions to meet your design needs. To do so, follow these guidelines:

- LogicLock regions with pin assignments must be placed on the periphery of the device, adjacent to the pins. (For Stratix, Stratix GX, Stratix II, MAX II, and Cyclone devices, you must also include the I/O block.)
- Floating LogicLock regions cannot overlap.
- It is recommended that you not create fixed and locked regions that overlap.
- After back-annotating a region, the software can place the region only in areas on the device with exactly the same resources.

These guideline are particularly important if you want to import multiple instances of a module into a top-level design, because you must ensure that the device has two or more locations with exactly the same device resources. If the device does not have another area with exactly the same resources, the Quartus II software generates a fitting error during compilation of the top-level design.

Figure 10–19 shows a floorplan with two instantiations of the same module. Both modules have the same LogicLock constraints and require exactly the same resources. The Quartus II software places the two LogicLock regions in different areas in the devices that have the same resources.

*Figure 10–19. Floorplan of Two Instances of a LogicLock Region*



*Notes for Figure 10–19:*

(1) The back-annotated regions LLR1_Inst1 and LLR1_Inst2 have the same resources.

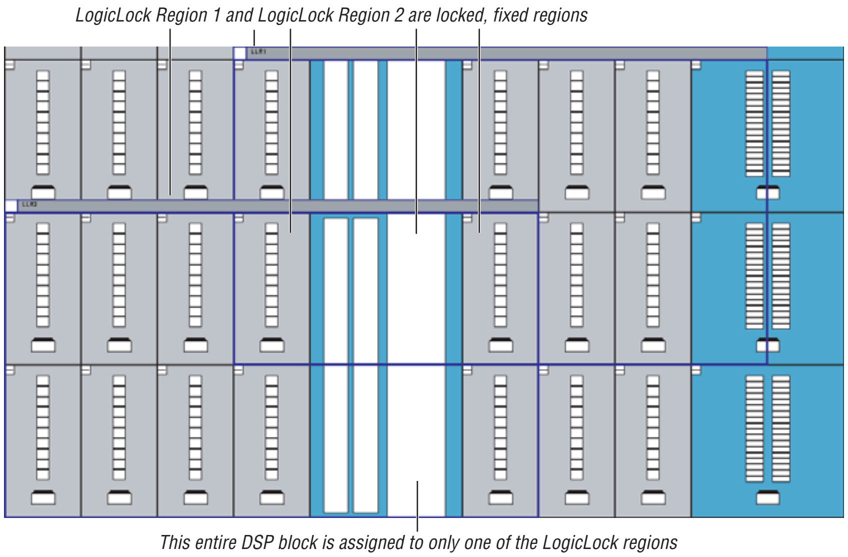
## Placing Memory, Pins & Other Device Features into LogicLock Regions

A LogicLock region includes all device resources within its boundaries. You can assign pins to LogicLock regions; however, this placement puts location constraints on the region. When the Quartus II software places a floating auto-sized region, it places the region in an area that meets the requirements of the LogicLock region's contents.

Pin assignments to LogicLock regions honor only fixed and locked regions. Pins assigned to floating regions do not influence the region's placement.

Only one LogicLock region can claim a device resource. If the boundary includes part of a device resource, such as a DSP block, the Quartus II software allocates the entire resource to the LogicLock region. Figure 10–20 shows two overlapping regions in the same Stratix DSP block. The Quartus II software can assign this resource to only one of the LogicLock regions. The region's resource requirements determine which region gets the assignment. If both regions require a DSP block, the Quartus II software issues a fitting error.

***Figure 10–20. Overlapping LogicLock Regions in a Stratix DSP Block***


LogicLock Region 1 and LogicLock Region 2 are locked, fixed regions
LLR1
LLR2
This entire DSP block is assigned to only one of the LogicLock regions

## Back-Annotating Routing Information

LogicLock regions not only allow you to preserve the placement of logic, from one compilation to the next, but also allow you to retain the routing inside the LogicLock regions. With both placement and routing locked, you have an extremely portable design module that can be used many times in a top-level design without requiring further optimization.

☞ Back-annotate routing only if necessary because this can prevent the Quartus II Fitter from finding an optimal fit for your design.

You can back-annotate the routing by selecting **Routing** in the **Back-Annotate Assignments** dialog box (Assignments menu) (see Figure 10–3 on page 10–5).

☞ If you are not using an atom netlist, you must turn On the **Save a node-level netlist into a Verilog Quartus Mapping File** option On in the **Back-Annotate Assignments** dialog box if back-annotation of routing is selected. Writing out a VQM file causes the Quartus II software to enforce persistent naming of nodes when saving the routing information between source and destination logic. The VQM is then be used as the design's source.

Back-annotated routing information is valid only for regions with fixed sizes and locked locations. The Quartus II software ignores the routing information for LogicLock regions you specify as floating and automatically sized.

The **Disable Back-Annotated Node locations** option in the **LogicLock Region Properties** dialog box is not available if the region contains both back-annotated routing and back-annotated nodes.

### Exporting Back-Annotated Routing in LogicLock Regions

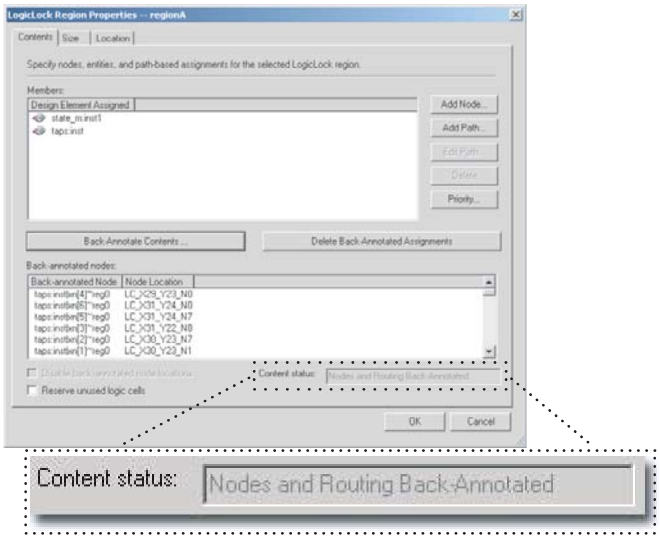
You can export the LogicLock region routing information by turning On the **Export Back-annotated routing** option On in the **Export Assignments** dialog box (Assignments menu). This generates a QSF and a RCF in the specified directory. The QSF file contains all LogicLock region properties as specified in the current design. The RCF contains all the necessary routing information for the exported LogicLock regions.

This RCF only works with the atom netlist for the entity being exported.

Only regions that have back-annotated routing information have their routing information exported when you export the LogicLock regions. All other regions are exported as regular LogicLock regions.

To determine if a LogicLock region contains back-annotated routing, see the **Content Status** box shown on the **Contents** tab of the **LogicLock Region Properties** dialog box. If routing has been back-annotated, the status is "Nodes and Routing Back-Annotated", shown in Figure 10–21.

***Figure 10–21. LogicLock Status***


LogicLock Region Properties -- regionA
Contents
Size
Location
Add Node...
Add Path...
Edit Path...
Delete
Priority...
Back Annotate Contents ...
Delete Back Annotated Assignments
OK
Cancel

Specify nodes, entities, and path-based assignments for the selected LogicLock region.

Members:

| Design Element Assigned |
|---|
| state_m:inst1 |
| taps:inst |

Back-annotated nodes:

| Back-annotated Node | Node Location |
|---|---|
| taps:inst\|bn[4]~reg0 | LC_X29_Y23_N0 |
| taps:inst\|bn[6]~reg0 | LC_X31_Y24_N0 |
| taps:inst\|bn[5]~reg0 | LC_X31_Y24_N7 |
| taps:inst\|bn[3]~reg0 | LC_X31_Y22_N8 |
| taps:inst\|bn[2]~reg0 | LC_X30_Y23_N7 |
| taps:inst\|bn[1]~reg0 | LC_X30_Y23_N1 |

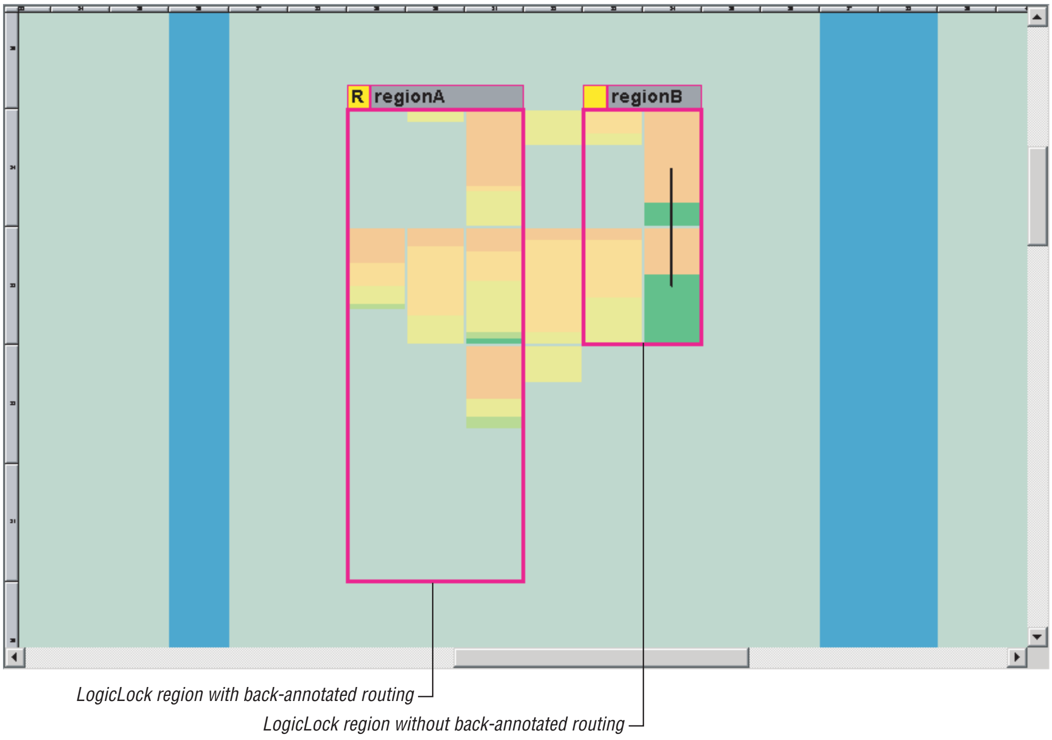
☐ Disable back-annotated node locations...
☐ Reserve unused logic cells

Content status: Nodes and Routing Back-Annotated

Content status: Nodes and Routing Back-Annotated

The Quartus II software also reports whether routing information has been back-annotated in the **Timing Closure Floorplan** (Assignments menu). LogicLock regions with back-annotated routing have an "R" in the top-left hand corner of the region as shown in Figure 10–22).

*Figure 10–22. Back-Annotation of Routing*

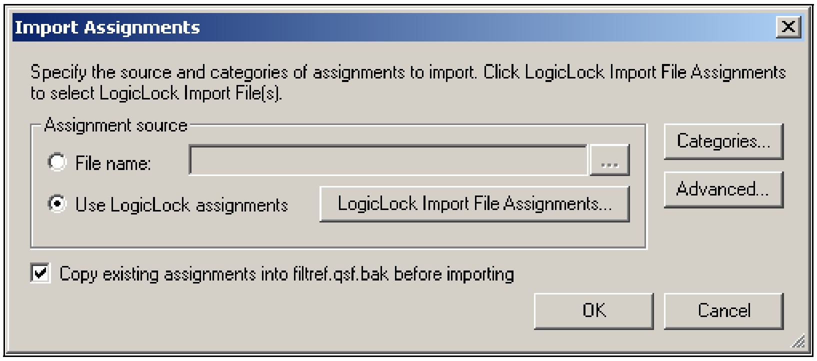

R
regionA
regionB
LogicLock region with back-annotated routing
LogicLock region without back-annotated routing

## Importing Back-Annotated Routing in LogicLock Regions

To import LogicLock region routing information, turn the **Back-annotated routing** option on in the **Advanced Import Assignments** dialog box (Assignments menu). Figure 10–23 shows this dialog box. The Quartus II software imports and applies all LogicLock region assignments for the appropriate instances automatically.

☞ An RCF must be explicitly defined using the LogicLock **Back-annotated Routing Import File Name** option for the Quartus II software to import routing information for your design.

*Figure 10–23. Import LogicLock Regions*


Import Assignments

Specify the source and categories of assignments to import. Click LogicLock Import File Assignments to select LogicLock Import File(s).

Assignment source

File name:

[✓ Use LogicLock assignments]

[✓ Copy existing assignments into filtref.qsf.bak before importing]

The Quartus II software imports LogicLock regions with back-annotated routing as regions locked to a location and of fixed size.

You can import back-annotated routing if only one instance of the imported region exists in the top level of the design. If more than one instance of the imported region exists in the top level of the design, the routing constraint is ignored and the LogicLock region is imported without back-annotation of routing. This is because routing resources from one part of the device may not be exactly the same in another area of the device.

When importing the RCF for a lower-level entity you must use the same atom netlist, i.e., the VQM, that was used to generate the RCF file. This ensures that the node names annotated in the RCF match those in the atom netlist.

## Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt. For detailed information about scripting command options, refer to the Quartus II Command-Line and Tcl API Help browser. To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp ↵
```

For more information about Tcl scripting, see the *Tcl Scripting* chapter in Volume 2 of the *Quartus II Handbook*. For more information about command-line scripting, see the *Command-Line Scripting* chapter in Volume 2 of the *Quartus II Handbook*.

### Initializing and Uninitializing a LogicLock Region

You must initialize the LogicLock data structures before creating or modifying any LogicLock regions and before executing any of the Tcl commands listed below.

Use the following Tcl command to initialize the LogicLock data structures:

```
initialize_logiclock
```

Use the following command to uninitialize the LogicLock data structures before closing your project:

```
uninitialize_logiclock
```

### Creating or Modifying LogicLock Regions

Use the following Tcl command to create or modify a LogicLock region:

```
set_logiclock -auto_size true -floating true -region \
<my_region-name>
```

In the above example the region's size will be set to auto and the state set to floating.

If you specify a region name that does not exist in the design, the command creates the region with the specified properties. If you specify the name of an existing region, the command changes all properties you specify, and leaves unspecified properties unchanged.

For more information about creating LogicLock regions, see "Creating LogicLock Regions" on page 10–2.

### Obtaining LogicLock Region Properties

Use the following Tcl command to obtain LogicLock region properties. This example returns the height of the region named `my_region`.

```
get_logiclock -region my_region -height
```

### Assigning LogicLock Region Content

Use the following Tcl commands to assign or change nodes and entities in a LogicLock region. This example assigns all nodes with names matching fifo* to the region named `my_region`.

```
set_logiclock_contents -region my_region -to fifo*
```

You can also make path-based assignments with the following Tcl command:

```
set_logiclock_contents -region my_region -from \
fifo -to ram*
```

For more information about assigning LogicLock Region Content, refer to "Assigning LogicLock Region Content" on page 10–13.

### Prevent Further Netlist Optimization

Use this Tcl code to prevent further netlist optimization for nodes in a back-annotated LogicLock region. In your code, specify the name of your LogicLock region.

```
foreach node [get_logiclock_contents -region \
<region name> -node_locations] {

   set node_name [lindex $node 0]

   set_instance_assignment -name
ADV_NETLIST_OPT_ALLOWED "NEVER ALLOW" -to $node_name
}
```

The `get_logiclock_contents` command is in the `logiclock` package.

For more information about preventing further netlist optimization, refer to "Prevent Further Netlist Optimization" on page 10–38.

### Save a Node-level Netlist into a Persistent Source File (.vqm)

Make the following assignments to cause the Quartus II Fitter to save a node-level netlist into a VQM file:

```
set_global_assignment \
-name LOGICLOCK_INCREMENTAL_COMPILE_ASSIGNMENT ON
set_global_assignment \
-name LOGICLOCK_INCREMENTAL_COMPILE_FILE <file name>
```

Any path specified in the file name must be relative to the project directory. For example, specifying **atom_netlists/top.vqm** places **top.vqm** in the **atom_netlists** subdirectory of your project directory.

A VQM file is saved in the directory specified at the completion of a full compilation.

For more information about saving a node-level netlist, see"Atom Netlist Design Information" on page 10–18.

### Exporting LogicLock Regions

Use the following Tcl command to export LogicLock region assignments. This example exports all LogicLock regions in your design to a file called **export.qsf**.

```
logiclock_export -file_name export.qsf
```

For more information about exporting LogicLock Regions see "Export the Module" on page 10–17.

### Importing LogicLock Regions

Use the following Tcl commands to import LogicLock region assignments. This example ignores any pin assignments in the imported region.

```
set_instance_assignment -name LL_IMPORT_FILE \
my_region.qsf

logiclock_import -no_pins
```

Running the import command imports the assignment types for each entity in the design hierarchy. The assignments are imported from the file specified in the `LL_IMPORT_FILE` setting.

For more information about importing LogicLock Regions, see "Import the Module" on page 10–19.

### Setting LogicLock Assignment Priority

Use the following Tcl code to set the priority for LogicLock region's members. this example reverses the priorities of the LogicLock region in your design.

```
set reverse [list]
foreach member [get_logiclock_member_priority] {
   set reverse [insert $reverse 0 $member]
{
set_logiclock_member_priority $reverse
```

For more information about Setting the LogicLock Assignment Priority, see "Constraint Priority" on page 10–30.

### Assigning Virtual Pins

Use the following Tcl command to turn on the virtual pin setting for a pin called `my_pin`:

```
set_instance_assignment -name VIRTUAL_PIN ON -to my_pin
```

For more information about Assigning Virtual Pins, see "Virtual Pins" on page 10–28.

### Back-Annotating LogicLock Regions

Use the following command line option to back-annotate a design called `my_project` and demote assignments to LAB-level assignments.

```
quartus_cdb --back_annotate=lab my_project
```

For more information about Tcl scripting, see the *Tcl Scripting* chapter in Volume 2 of the *Quartus II Handbook*. For more information about command-line scripting, see the *Command-Line Scripting* chapter in Volume 2 of the *Quartus II Handbook.*

## Conclusion

The LogicLock block-based design flow shortens design cycles because it allows design and implementation of design modules to occur independently, and preserves performance of each design module during system integration. You can export modules, making design reuse easier.

You can include a module in one or more projects while maintaining performance, and reducing development costs and time-to-market. LogicLock region assignments give you complete control over logic and memory placement so that you can use LogicLock region assignments to improve the performance of non-hierarchical designs.

# 11. Timing Closure in HardCopy Devices

qii52010-2.0

## Introduction

Timing analysis is performed on an FPGA design to determine that the design's performance meets the required timing goals. This analysis includes system clock frequency ($f_{MAX}$), setup and hold timing for the design's top-level input ports, as well as clock-to-output timing for all top-level output ports. Measuring these parameters against performance goals ensures that the FPGA design functions as planned in the end target system.

After the FPGA design is stabilized, fully tested in-system, and satisfies the HardCopy® design rules, the design can be migrated to a HardCopy device. Altera® performs the same rigorous timing analysis on the HardCopy device during its implementation, ensuring that it meets the same timing goals. Because the critical timing paths of the HardCopy version of a design are different from the corresponding paths in the FPGA version, meeting the same timing goals is particularly important.

Timing improvements in HardCopy as compared to the equivalent FPGA devices exist for several reasons. While maintaining the same rich set of features as the corresponding FPGA, HardCopy devices have a highly optimized die size to make them as small as possible. Because of the customized interconnect structure that makes this optimization possible, the delay through each signal path is less than the original FPGA design. Quartus® II software versions 4.0 and later determine routing and associated buffer insertion for the design and provides the Timing Analyzer with more accurate information on the delays than was possible in the previous version of the Quartus II software.

The differences in the timing between HardCopy devices and FPGAs is inconsequential as long as the HardCopy device is checked against a specification that fully defines the timing of the design. After this timing goal is fully defined, the HardCopy device is guaranteed to function correctly.

This chapter describes how to meet the required timing performance of HardCopy devices and improve it.

## Timing Closure

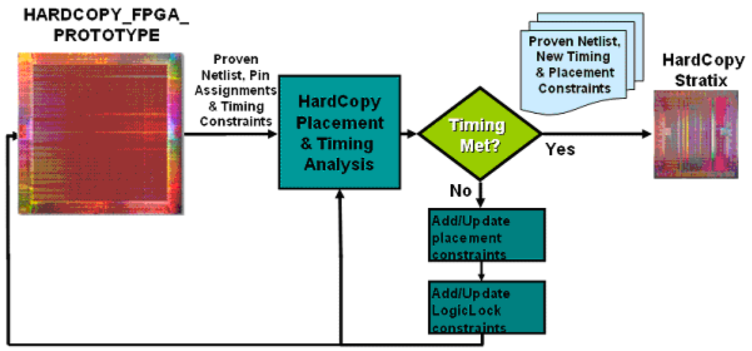
Many of today's developers are faced with the difficult task of meeting the timing goals of systems designed with an ASIC, which can consume many valuable months of intensive engineering effort. The slower development process exists because, in today's silicon technology

(0.18 µm and 0.13 µm), the delay associated with interconnect dominates the delay associated with the transistors used to make the logic gates. Consequently, ASIC performance is sensitive to the physical placement-and-routing of the logic blocks that make up the design.

On migration, the HardCopy device is structurally identical to its FPGA counterpart; there is no re-synthesis or library re-mapping required. Since the interconnect lengths are much smaller in the HardCopy device than they are in the FPGA, the place-and-route engine compiling the HardCopy design has a considerably less difficult task than it does in an equivalent ASIC development. Coupled with detailed timing constraints, the place-and-route is timing driven.

Figure 11–1 illustrates the design flow for estimating performance and optimizing the designs. You can target your designs to HARDCOPY_FPGA_PROTOTYPE devices, and pass the design information to the placement and timing analysis engine to estimate the performance of HardCopy Stratix® devices. In the event that the required performance is not met, you can modify or add placement and LogicLock™ constraints. If the performance goals are still not met, then change your RTL source, optimize the FPGA design, and estimate timing iteratively.

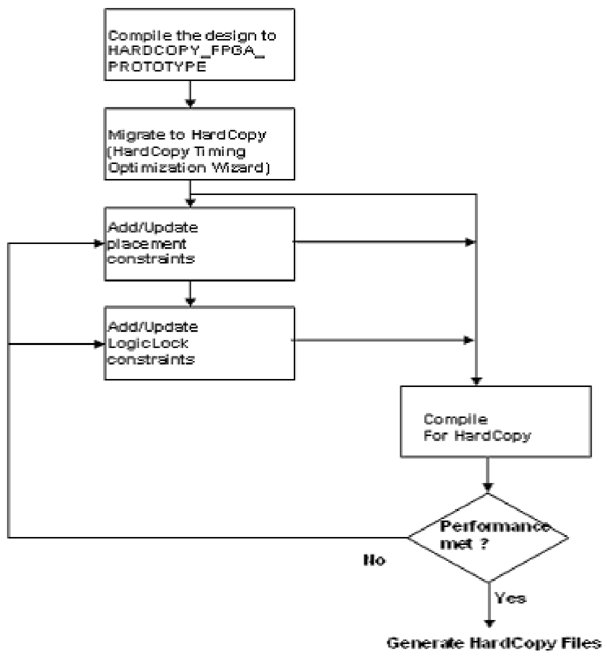
*Figure 11–1. Design Flow for Estimating Performance & Optimizing the Design*


HARDCOPY_FPGA_ PROTOTYPE
Proven Netlist, Pin Assignments & Timing Constraints
HardCopy Placement & Timing Analysis
Timing Met?
Yes
No
Proven Netlist, New Timing & Placement Constraints
HardCopy Stratix
Add/Update placement constraints
Add/Update LogicLock constraints

### Placement Constraints

The Quartus II software version 4.0 and later supports placement constraints and LogicLock regions for HardCopy Stratix devices. Figure 11–2 shows an iterative process to modify the placement constraints until the best placement for the HardCopy Stratix device is obtained to achieve the best performance.

*Figure 11–2. Placement Constraints Flow for HardCopy Stratix Devices*


Compile the design to HARDCOPY_FPGA_ PROTOTYPE
Migrate to HardCopy (HardCopy Timing Optimization Wizard)
Add/Update placement constraints
Add/Update LogicLock constraints
Compile For HardCopy
Performance met ?
No
Yes
Generate HardCopy Files

## Location Constraints

### Location Array Block (LAB) Assignments

Location constraints for HardCopy Stratix devices are supported. To achieve better performance, you can make LAB-level assignments after migrating the HARDCOPY_FPGA_ PROTOTYPE project, and before compiling the design for a HardCopy Stratix device. One important consideration for LAB reassignments is that the entire contents of a LAB is moved to another empty LAB. If you want to move the logic contents of LAB "A" to LAB "B," the entire contents of LAB A is moved to an empty LAB B. For example, the logic contents of LAB_X33_Y65 can be moved to an empty LAB at LAB_X43_Y56.

## LogicLock Assignments

### *LogicLock*

Quartus II software enables a block-based design approach using LogicLock. With LogicLock, designers can create and implement each logic module independently, and then integrate all of the optimized modules into the top-level design.

For more information about the LogicLock design methodology, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook*.

LogicLock constraints are supported when you are migrating the project from a HARDCOPY_FPGA_PROTOTYPE project to a HardCopy Stratix project. If the LogicLock region was specified as "Size=Fixed" and "Location=Locked" in the HARDCOPY_FPGA_PROTOTYPE project, it is converted to have "Size=Auto" and "Location=Floating", as shown in "Examples of Supported LogicLock Constraints". This modification is necessary because the floorplan of a HardCopy Stratix device is different from that of an equivalent Stratix device. If this modification did not occur, LogicLock assignments would lead to no-fits due to bad placement. Making the regions auto-size and floating maintains your module or entity LogicLock assignments, allowing you to easily adjust the LogicLock regions as required to improve the performance.

**Examples of Supported LogicLock Constraints**
LogicLock Region Definition in the HARDCOPY_FPGA_PROTOTYPE QSF File:

```
set_global_assignment -name LL_HEIGHT 15 -entity risc8 -section_id test

set_global_assignment -name LL_WIDTH 15 -entity risc8 -section_id test

set_global_assignment -name LL_STATE LOCKED -entity risc8 -section_id test

set_global_assignment -name LL_AUTO_SIZE OFF -entity risc8 -section_id test
```

LogicLock Region Definition in the Migrated HardCopy Stratix QSF File:

```
set_global_assignment -name LL_HEIGHT 15 -entity risc8 -section_id test

set_global_assignment -name LL_WIDTH 15 -entity risc8 -section_id test

set_global_assignment -name LL_STATE FLOATING -entity risc8 -section_id test

set_global_assignment -name LL_AUTO_SIZE ON -entity risc8 -section_id test
```

### Tutorial

To learn more about the LAB and LogicLock assignments, perform the tutorial available on www.altera.com/literature. The tutorial illustrates the performance improvement by LAB and LogicLock assignments. To know more about the performance improvements in general for FPGA designs, refer to the following application notes:

**Design Optimization for Altera Devices**
http://www.altera.com/literature/hb/qts/qts_qii52005.pdf

**Timing Closure Floorplan**
http://www.altera.com/literature/hb/qts/qts_qii52006.pdf

**LogicLock Design Methodology**
http://www.altera.com/literature/hb/qts/qts_qii52009.pdf

## Minimizing Clock Skew

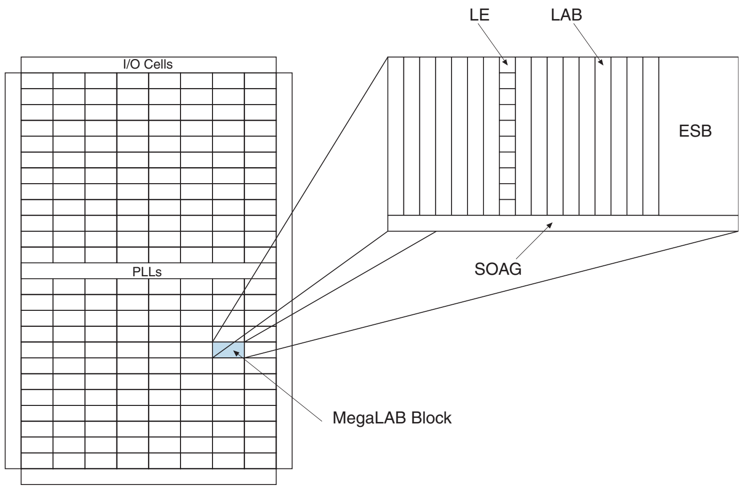
The clock is an important component that affects design performance in any digital integrated circuit. The discussion in the remainder of this section pertains to HardCopy APEX™ 20KE, HardCopy APEX 20KC, and HardCopy Stratix devices.

The architecture of the HardCopy HC20K device is based on the APEX 20KE and APEX 20KC FPGA devices and the HardCopy Stratix devices are based on the Stratix FPGA devices. The same dedicated clock trees (`CLK[3..0]`) that exist in APEX 20KE and APEX 20KC devices or (`CLK[15..0]`) that exist in Stratix devices also exist in the corresponding HardCopy device. These clock trees are carefully designed and optimized to minimize the clock skew over the entire device. The clock trees are balanced by maintaining the same loading at the end of each point of the clock trees, regardless of what resources (logic elements [LEs], embedded system blocks [ESBs], and input/output elements [IOEs]) are used in any design. The insertion delay of the HardCopy-dedicated clock trees is marginally faster than in the corresponding APEX 20KE, APEX 20KC, or Stratix FPGA device because of the smaller footprint of the HardCopy silicon.

Because there is a large area overhead for these global signals that may not be used on every design, the `FAST` bidirectional pins (`FAST[3..0]`) of the HardCopy APEX 20KE and HardCopy APEX 20KC or the dedicated fast regional I/O pins of HardCopy Stratix do not have dedicated pre-built clock or buffer trees in HardCopy devices. If any of the `FAST`/dedicated fast regional signals are used as clocks, a clock tree is synthesized by the place-and-route tool after the placement of the design has occurred. The skew and insertion delay of these synthesized clock trees are carefully controlled, ensuring that the timing requirements of the design are met. You can also use the FAST signals of HardCopy APEX or the dedicated fast regional I/O pins of HardCopy Stratix as high fan-out reset or enable signals. For these cases, skew is usually less important than insertion delay. To reiterate, a buffer tree is synthesized after the design placement.

The clock or buffer trees that are synthesized for the FAST pins of HardCopy APEX 20KE and HardCopy APEX 20KC or the dedicated fast regional I/O pins of HardCopy Stratix are built from special cells in the HardCopy base design. These cells do not exist in the FPGA. They are used in the HardCopy design exclusively to meet timing and testing goals. They are not available to make any logical changes to the design as implemented in the FPGA. These resources are called the strip of auxiliary gates (SOAG). There is one of these strips per MegaLAB™ structure in HardCopy devices. Each SOAG consists of a number of primitive cells, and there are approximately 10 SOAG primitive cells per logic array block (LAB). Several SOAG primitives can be combined to form more complex logic, but the majority of SOAG resources are used for buffer tree, clock tree, and delay cell generation. Figure 11–3 shows the SOAG architectural feature.

***Figure 11–3. SOAG Architectural Feature***



For detailed information on the HardCopy device architecture, including SOAG resources, see the *HardCopy APEX 20K Device Family Data Sheet* chapter in Volume 1 of the *HardCopy Device Handbook.*

## Checking the HardCopy Device Timing

To ensure that the timing of the HardCopy device meets performance goals, detailed static timing analysis must be run on the HardCopy design database. For this timing analysis to be meaningful, all timing constraints and timing exceptions that were applied to the design for the FPGA implementation must also be used for the HardCopy implementation. If no timing constraints, or only partial timing constraints, were used for the FPGA design, a full set of constraints must be specified for the HardCopy design by filling in the unspecified constraints with default values. If this is not done, there is no way of knowing if the HardCopy device meets the required timing of the end target system. The timing constraints can be captured through the Timing Wizard in the Altera Quartus II software. The following constraints must be included:

- Clock Definitions
- Primary Input Pin Timing
- Primary Output Pin Timing
- Combinatorial Timing
- Timing Exceptions

### Clock Definitions

These definitions are used to describe the parameters of all different clock domains in a design. Clock parameters that must be defined are frequency, time at which the clock edge rises, time at which the clock edge falls, clock uncertainty (or skew), and clock name. Figures 11–4 and 11–5 show these clock attributes.

*Figure 11–4. Clock Attributes*

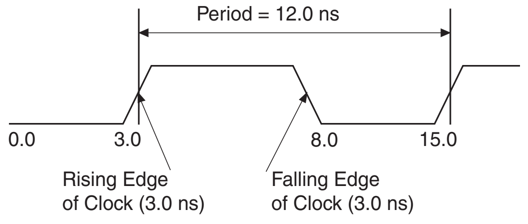
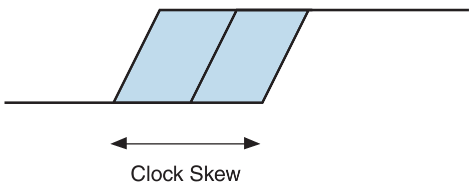


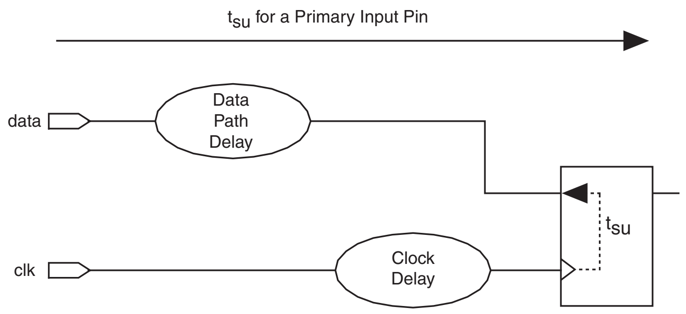
*Figure 11–5. Clock Skew*



## Primary Input Pin Timing

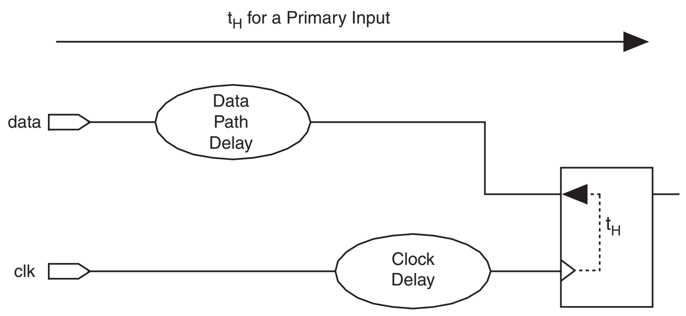
This constraint must be specified for every primary input pin in the design (and for the input path of every bidirectional pin). The input pin timing can be captured in two ways. The first is to describe what maximum on-chip delay is acceptable (i.e., the setup time of a primary input to any register in the design relative to a specific clock). Figure 11–6 depicts a generic circuit with an on-chip setup-time constraint, which may be different for each clock domain.

*Figure 11–6. On-Chip Setup-Time Constraint*



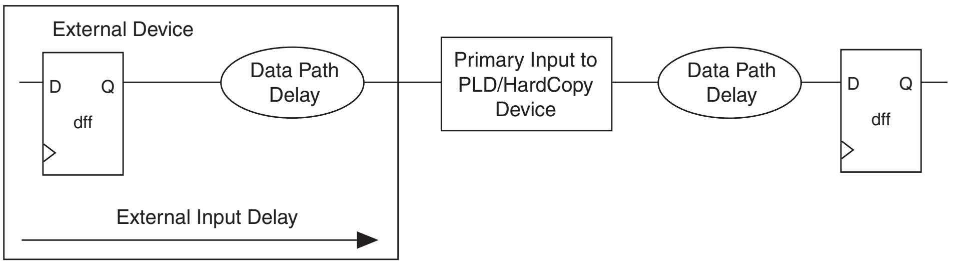
The minimum on-chip delay from any primary input pin must be specified to describe input hold-time requirements. Figure 11–7 depicts a generic circuit with an on-chip hold-time constraint.

*Figure 11–7. On-Chip Hold-Time Constraint*



The second way to capture the input pin timing is to describe the external timing environment, which is the maximum and minimum arrival times of the external signals that drive the primary input pins of the HardCopy device or FPGA. Figure 11–8 shows the external timing constraint that drives the primary input pin. This external input delay time can be used by the static timing analysis tool to check that there is enough time for the data to propagate to the internal nodes of the device. If there is not enough time, then a timing violation occurs.

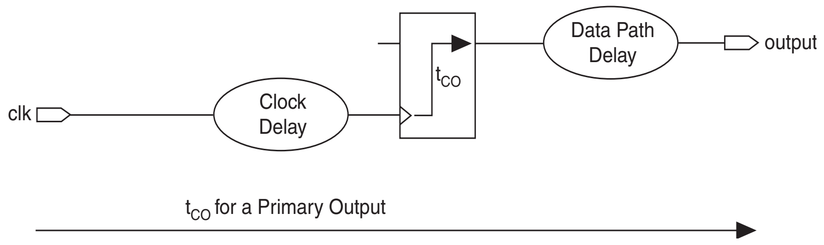
*Figure 11–8. External Timing Constraint Driving a Primary Input Pin*



## Primary Output Pin Timing

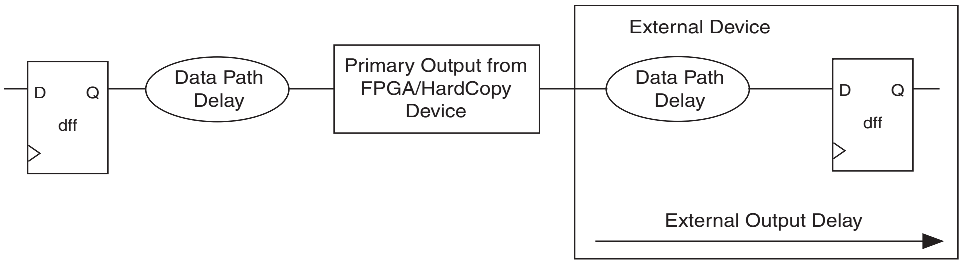
This constraint must be specified for every primary output pin in the design (and for the output path of every bidirectional pin). The output pin timing is captured in two ways. The first is to describe what maximum (and minimum) on-chip clock-to-output ($t_{CO}$) delay is acceptable (i.e., the time it takes from the active edge of the clock to the data arriving at the primary output pin). Figure 11–9 depicts a generic circuit with an on-chip $t_{CO}$ time constraint. Also, there can be a minimum $t_{CO}$ requirement.

*Figure 11–9. On-Chip Clock-to-Output ($T_{CO}$) Time Constraint*



The second way to capture output pin timing is to describe the external timing environment, which is the maximum and minimum delay times of external signals that are driven by the primary output pins of the HardCopy device or FPGA. Figure 11–10 shows the external timing constraint driven by the primary output pin. The static timing analysis tool uses this information to check that the on-chip timing of the output signals is within the desired specification.

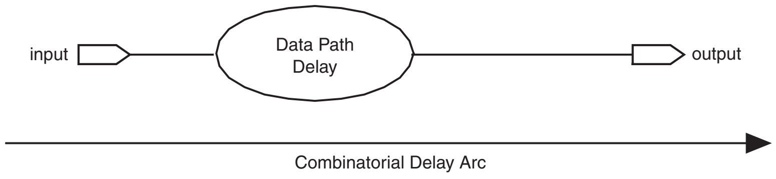
*Figure 11–10. External Timing Constraint for a Primary Output Pin*



## Combinatorial Timing

Combinatorial timing occurs when there is a path from a primary input pin to a primary output pin. This type of circuit does not contain any registers. Therefore, it does not require a clock for constraint specification. The maximum and minimum delay from the primary input pin to the primary output pin is all that is needed. Figure 11–11 shows a generic circuit where a combinatorial timing arc constraint must be placed.

*Figure 11–11. Combinatorial Timing Constraint*



### Timing Exceptions

Some circuit structures warrant special consideration. For example, when a design has more than one clock domain and the clock domains are not related, all timing paths between the two clock domains can be ignored. All timing paths using the static timing analysis tool can be ignored by specifying false paths for all signals that go from one clock domain to the other clock domain(s). Additionally, there are circuits that are not intended to operate in a single-clock cycle. These circuits require that you specify multi-cycle clock exceptions.

After the information is captured, it can be used by Altera to directly check all timing of the HardCopy device before tape out occurs. If any timing violations are found in the HardCopy device due to over-aggressive timing constraints, they must either be fixed by Altera, or waived by the customer.

For more information on timing analysis, see the *Quartus II Timing Analysis* chapter and the *Synopsys PrimeTime Support* chapter in Volume 3 of the *Quartus II Handbook*.

## Correcting Timing Violations

After the customized metal interconnect is generated for the HardCopy device, Altera checks the timing of the design with an industry standard static timing analysis tool, PrimeTime. Timing violations are reported by this tool, and they are subsequently corrected.

### Hold-Time Violations

Because the interconnect in a HardCopy device is customized for a particular application, it is possible that hold-time ($t_H$) violations exist in the HardCopy device after place-and-route occurs. A hold violation exists if the sum of the delay in the clock path between two registers plus the micro hold time of the destination register is greater than the delay of the data path from the source register to the destination register. The following equation describes this relationship.

$$t_H \text{ Slack} = \text{Data Delay} - \text{Clock Delay} - \text{Micro } t_H$$

If a negative slack value exists, there is a hold-time violation. Any hold-time violation present in the HardCopy design database after the interconnect data is generated is removed by inserting the appropriate delay in the data path. The inserted delay is large enough to guarantee no hold violations under fast, low-temperature, high-voltage conditions.

Table 11–1 shows an example report of a Synopsys PrimeTime static timing analysis of a typical HardCopy design. This report shows that the circuit has a hold-time violation and a negative slack value. Table 11–2 shows the timing report for the same path after the hold violation has been fixed. The instance and cell names shown in these reports are generated as part of the HardCopy implementation process, and are based on the physical location of those elements in the device.

*Table 11–1. Static Timing Analysis Before Hold-Time Violation Fix (Part 1 of 2)*

```
Startpoint:GR23_GC0_L19_LE1/um6
(falling edge-triggered flip-flop clocked by CLK0')
Endpoint: GR23_GC0_L20_LE8/um6
(falling edge-triggered flip-flop clocked by CLK0')
Path Group: CLK0
Path Type:min
```

| Point | Incr | Path | Reference to Figure 11–12 |
|---|---|---|---|
| clock CLK0' (fall edge) | 0.00 | 0.00 | |
| clock network delay (propagated) | 2.15 | 2.15 | (1) |
| GR23_GC0_L19_LE1/um6/clk (c1110) | 0.00 | 2.15 f | (2) |
| GR23_GC0_L19_LE1/um6/regout (c1110) | 0.36 * | 2.52 r | (2) |
| GR23_GC0_L19_LE1/REGOUT (c1000_2d7a8) | 0.00 | 2.52 r | (2) |
| GR23_GC0_L20_LE8/LUTD (c1000_56502) | 0.00 | 2.52 r | (3) |
| GR23_GC0_L20_LE8/um1/datad (indsim) | 0.01 * | 2.52 r | (3) |
| GR23_GC0_L20_LE8/um1/ndsim (indsim) | 0.01 * | 2.53 f | (3) |
| GR23_GC0_L20_LE8/um5/ndsim (mxcascout) | 0.00 * | 2.53 f | (3) |
| GR23_GC0_L20_LE8/um5/cascout (mxcascout) | 0.06 * | 2.59 f | (3) |
| GR23_GC0_L20_LE8/um6/dcout (c1110) | 0.00 * | 2.59 f | (3) |
| data arrival time | | 2.59 | |
| | | | |
| clock CLK0' (fall edge) | 0.00 | 0.00 | |
| clock network delay (propagated) | 2.17 | 2.17 | (4) |
| clock uncertainty | 0.25 | 2.42 | (5) |

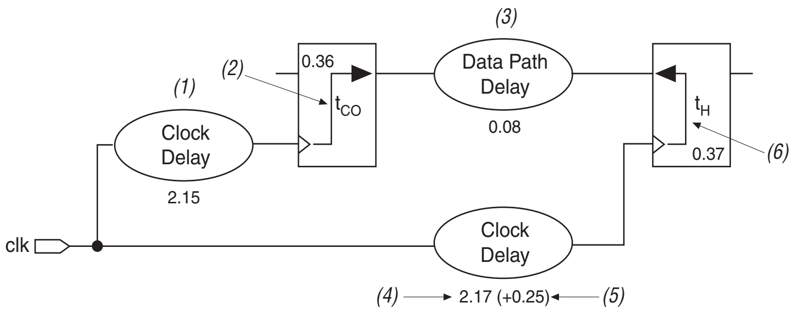
*Table 11–1. Static Timing Analysis Before Hold-Time Violation Fix (Part 2 of 2)*

```
Startpoint:GR23_GC0_L19_LE1/um6
(falling edge-triggered flip-flop clocked by CLK0')
Endpoint: GR23_GC0_L20_LE8/um6
(falling edge-triggered flip-flop clocked by CLK0')
Path Group: CLK0
Path Type:min
```

| Point | Incr | Path | Reference to Figure 11–12 |
|---|---|---|---|
| GR23_GC0_L20_LE8/um6/clk (c1110) | | 2.42 f | (6) |
| library hold time | 0.37 * | 2.79 | |
| data required time | | 2.79 | |
| data required time | | 2.79 | |
| data arrival time | | -2.59 | |
| slack (VIOLATED) | | -0.20 | |

Figure 11–12 shows the circuit described by the Table 11–1 static timing analysis report.

*Figure 11–12. Circuit with a Hold-Time Violation*



Placing the values from the static timing analysis report into the hold-time slack equation results in the following:

$$t_H \text{ Slack} = \text{Data Delay} - \text{Clock Delay} - \text{Micro } t_H$$

$$t_H \text{ Slack} = (2.15 + 0.36 + 0.08) - (2.17 + 0.25) - 0.37$$

$$t_H \text{ Slack} = -0.20 \text{ ns}$$

This result shows that there is negative slack in this path, meaning that there is a hold-time violation of 0.20 ns.

After fixing the hold violation, the timing report for the same path is regenerated (see Table 11–2). The netlist changes are in ***bold italic*** type.

*Table 11–2. Static Timing Analysis After Hold-Time Violation Fix (Part 1 of 2)*

```
Startpoint: GR23_GC0_L19_LE1/um6
(falling edge-triggered flip-flop clocked by CLK0')
Endpoint: GR23_GC0_L20_LE8/um6
(falling edge-triggered flip-flop clocked by CLK0')
Path Group: CLK0
Path Type: min
Static Timing Analysis After Hold-Time Violation Fix
```

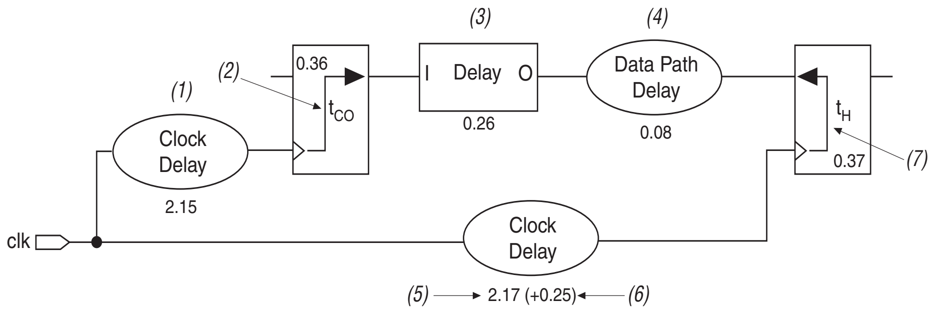
| Point | Incr | Path | Reference to Figure 11–13 |
|---|---|---|---|
| clock CLK0' (fall edge) | 0.00 | 0.00 | (1) |
| clock network delay (propagated) | 2.15 | 2.15 | (1) |
| GR23_GC0_L19_LE1/um6/clk (c1110) | 0.00 | 2.15 f | (2) |
| GR23_GC0_L19_LE1/um6/regout (c1110) | 0.36 * | 2.52 r | (2) |
| GR23_GC0_L19_LE1/REGOUT (c1000_2d7a8) | 0.00 | 2.52 r | (2) |
| ***thc_916/A (de105)*** | ***0.01 **** | ***2.52 r*** | (3) |
| ***thc_916/Z (de105)*** | ***0.25 **** | ***2.78 r*** | (3) |
| GR23_GC0_L20_LE8/LUTD (c1000_56502) | 0.00 | 2.78 r | (3) |
| GR23_GC0_L20_LE8/um1/datad (indsim) | 0.01 * | 2.78 r | (3) |
| GR23_GC0_L20_LE8/um1/ndsim (indsim) | 0.01 * | 2.79 f | (3) |
| GR23_GC0_L20_LE8/um5/ndsim (mxcascout) | 0.00 * | 2.79 f | (3) |
| GR23_GC0_L20_LE8/um5/cascout (mxcascout) | 0.06 * | 2.85 f | (3) |
| GR23_GC0_L20_LE8/um6/dcout (c1110) | 0.00 * | 2.85 f | (3) |
| data arrival time | | 2.85 | |
| | | | |
| clock CLK0' (fall edge) | 0.00 | 0.00 | |
| clock network delay (propagated) | 2.17 | 2.17 | (4) |
| clock uncertainty | 0.25 | 2.42 | (5) |
| GR23_GC0_L20_LE8/um6/clk (c1110) | | 2.42 f | (6) |
| library hold time | 0.37 * | 2.79 | |
| data required time | | 2.79 | |

*Table 11–2. Static Timing Analysis After Hold-Time Violation Fix (Part 2 of 2)*

| | |
|---|---|
| `data required time` | `2.79` |
| `data arrival time` | `-2.85` |
| `slack (MET)` | `+0.06` |

Figure 11–13 shows the circuit described by the Table 11–2 static timing analysis report.

*Figure 11–13. Circuit Including a Fixed Hold-Time Violation*



Placing the values from the static timing analysis report into the hold-time slack equation results in the following.

$$t_H \text{ Slack} = \text{Data Delay} - \text{Clock Delay} - \text{Micro } t_H$$

$$t_H \text{ Slack} = (2.15+ 0.36 + 0.26 + 0.08) - (2.17 + 0.25) - 0.37$$

$$t_H \text{ Slack} = +0.06 \text{ ns}$$

In this timing report, the slack of this path is reported as 0.06 ns. Therefore, this path does not have a hold-time violation. The path was fixed by the insertion of a delay cell (`del05`) into the data path, which starts at the REGOUT pin of cell `GR23_GC0_L19_LE1` and finishes at the `LUTD` input of cell `GR23_GC0_L20_LE8`. The instance name of the delay cell in this case is `thc_916`.

A clock_uncertainty of 0.25 ns is specified in this timing report, and is used to add extra margin during the hold-time calculation, making the design more robust. This feature is a part of the static timing analysis tool, not of the HardCopy design.

The delay cell is created out of the SOAG resources that exist in the HardCopy base design.

## Setup-Time Violations

A setup violation exists if the sum of the delay in the data path between two registers plus the micro setup time ($t_{SU}$) of the destination register is greater than the sum of the clock period and the clock delay at the destination register. The following equation describes this relationship:

$t_{SU}$ Slack = Clock Period + Clock Delay – (Data Delay + Micro $t_{SU}$)

If there is a negative slack value, it means that there is a setup-time violation. There are several potential mechanisms that can cause a setup-time violation. The first is when the synthesis tool is unable to meet the required timing goals. However, a HardCopy design does not rely on any resynthesis to a new cell library; the synthesis results that were generated as part of the original FPGA design are maintained, meaning that the HardCopy implementation of a design uses exactly the same structural netlist as its FPGA counterpart. For example, if you used a particular synthesis option to ensure that a particular path only contained a certain number of logic levels, the HardCopy design will contain exactly the same number of logic levels for that path. Consequently, if the FPGA was free of setup-time violations, no setup-time violations occur in the HardCopy device as a result of the netlist structure.

The second mechanism that can cause setup-time violations is differing placement of the resources in the netlist for the HardCopy device compared to the original FPGA. This scenario is extremely unlikely as the place-and-route tool used during the HardCopy implementation performs timing-driven placement. In extreme cases, some manual placement modification might be necessary. The placement is performed at the LAB and ESB level, meaning that the placement of logic cells inside each LAB is fixed, and is identical to the placement of the FPGA. IOEs have fixed placement to maintain the pin and package compatibility of the original FPGA.

The third, and most likely, mechanism for setup-time violations occurring in the HardCopy device is a signal with a high fan-out. In the FPGA, high fan-out signals are buffered by large drivers that are an integral part of the programmable interconnect structure. Consequently, a signal that was fast in the FPGA can be initially slower in the HardCopy version, which is before any buffering is inserted into the HardCopy design to increase the speed of the slow signal. The place-and-route tool detects these signals and automatically creates buffer trees using SOAG resources, ensuring that the heavily loaded, high fan-out signal is fast enough to meet performance requirements.

Table 11–3 shows the timing report for a path that contains a high fan-out signal *before* the place-and-route process. Table 11–4 shows the timing report for a path that contains a high fan-out signal *after* the place-and-route process. Before the place-and-route process, there is a large delay on the high fan-out net that is driven by the pin `GR12_GC0_L2_LE4/REGOUT`. This delay is due to the large capacitive load that the pin has to drive. For more information on this timing report, see Figure 11–14.

***Table 11–3. Timing Report Before the Place-&-Route Process (Part 1 of 2)***

```
Startpoint: GR12_GC0_L2_LE4/um6
(falling edge-triggered flip-flop clocked by clkx')
Endpoint: GR4_GC0_L5_LE2/um6
(falling edge-triggered flip-flop clocked by clkx')
Path Group: clkx
Path Type: max
```

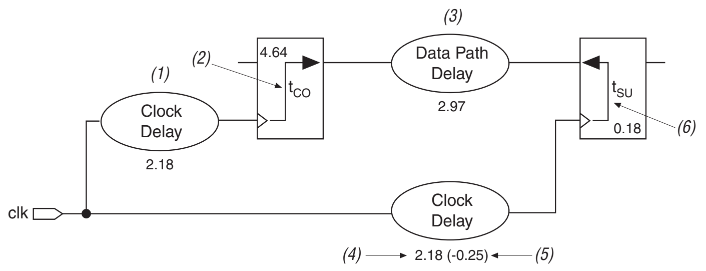
| Point | Incr | Path | Reference to Figure 11–14 |
|---|---|---|---|
| clock clkx' (fall edge) | 0.00 | 0.00 | (1) |
| clock network delay (propagated) | 2.18 | 2.18 | (1) |
| GR12_GC0_L2_LE4/um6/clk (c1110) | 0.00 | 2.18 f | (2) |
| GR12_GC0_L2_LE4/um6/regout (c1110) | | | (2) |
| GR12_GC0_L2_LE4/REGOUT (c1000_7f802) <- | | | (2) |
| GR4_GC0_L5_LE0/LUTC (c1000_0029a) | | | (3) |
| GR4_GC0_L5_LE0/um4/ltb (lt53b) | 2.36 | 9.18 f | (3) |
| GR4_GC0_L5_LE0/um5/cascout (mxcascout) | 0.07 | 9.24 f | (3) |
| GR4_GC0_L5_LE0/um2/COMBOUT (icombout) | 0.09 | 9.34 r | (3) |
| GR4_GC0_L5_LE0/COMBOUT (c1000_0029a) | 0.00 | 9.34 r | (3) |
| GR4_GC0_L5_LE2/LUTC (c1000_0381a) | 0.00 | 9.34 r | (3) |
| GR4_GC0_L5_LE2/um4/ltb (lt03b) | 0.40 | 9.73 r | (3) |
| GR4_GC0_L5_LE2/um5/cascout (mxcascout) | 0.05 | 9.78 r | (3) |
| GR4_GC0_L5_LE2/um6/dcout (c1110) | 0.00 | 9.78 r | (3) |
| data arrival time | | 9.79 | (3) |
| clock clkx’ (fall edge) | 7.41 | 7.41 | |

*Table 11–3. Timing Report Before the Place-&-Route Process (Part 2 of 2)*

| | | | |
|---|---|---|---|
| clock network delay (propagated) | 2.18 | 9.59 | (4) |
| clock uncertainty | -0.25 | 9.34 | (5) |
| GR4_GC0_L5_LE2/um6/clk (c1110) | | 9.34 f | |
| **Point** | **Incr** | **Path** | **Reference to Figure 11–14** |
| library setup time | -0.18 | 9.16 | (6) |
| data required time | | 9.16 | |
| data required time | | 9.16 | |
| data arrival time | | -9.79 | |
| slack (VIOLATED) | | -0.63 | |

Figure 11–14 shows the circuit described by the Table 11–3 static timing analysis report.

*Figure 11–14. Circuit that has a Setup-Time Violation*



The timing numbers in this report are based on pre-layout estimated delays.

Placing the values from the static timing analysis report into the setup-time slack equation results in the following.

$t_{SU}$ Slack = Clock Period + Clock Delay – (Data Delay + Micro $t_{SU}$)

$t_{SU}$ Slack = 7.41 + (2.18 - 0.25) - (2.18 + 4.64 + 2.97 + 0.18)

$t_{SU}$ Slack = -0.63 ns

This result shows that there is negative slack for this path, meaning that there is a setup-time violation of 0.63 ns.

After place-and-route, a buffer tree is constructed on the high fan-out net and the setup-time violation is fixed. The timing report for the same path is shown in Table 11–4. The changes to the netlist are in ***bold italic*** type. For more information on this timing report, see Figure 11–15.

*Table 11–4. Timing Report After the Place-and-Route Process (Part 1 of 2)*

```
Startpoint: GR12_GC0_L2_LE4/um6
(falling edge-triggered flip-flop clocked by clkx')
Endpoint: GR4_GC0_L5_LE2/um6
(falling edge-triggered flip-flop clocked by clkx')
Path Group: clkx
Path Type: max
```

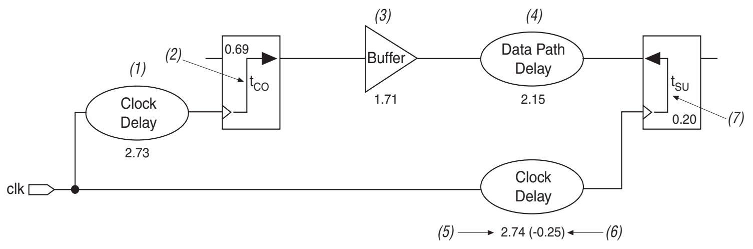
| Point | Incr | Path | Reference to Figure 11–15 |
|---|---|---|---|
| clock clkx' (fall edge) | 0.00 | 0.00 | |
| clock network delay (propagated) | 2.73 | 2.73 | (1) |
| GR12_GC0_L2_LE4/um6/clk (c1110) | 0.00 | 2.73 f | (2) |
| GR12_GC0_L2_LE4/um6/regout (c1110) | 0.69 * | 3.42 r | (2) |
| GR12_GC0_L2_LE4/REGOUT (c1000_7f802) <- | 0.00 | 3.42 r | (2) |
| ***N1188_iv06_1_0/Z (iv06)*** | ***0.06 **** | ***3.49 f*** | (3) |
| ***N1188_iv06_2_0/Z (iv06)*** | ***0.19 **** | ***3.68 r*** | (3) |
| ***N1188_iv06_3_0/Z (iv06)*** | ***0.12 **** | ***3.80 f*** | (3) |
| ***N1188_iv06_4_0/Z (iv06)*** | ***0.10 **** | ***3.90 r*** | (3) |
| ***N1188_iv06_5_0/Z (iv06)*** | ***0.08 **** | ***3.97 f*** | (3) |
| ***N1188_iv06_6_2/Z (iv06)*** | ***1.16 **** | ***5.13 r*** | (3) |
| GR4_GC0_L5_LE0/LUTC (c1000_0029a) | 0.00 | 5.13 r | (4) |
| GR4_GC0_L5_LE0/um4/ltb (lt53b) | 1.55 * | 6.68 f | (4) |
| GR4_GC0_L5_LE0/um5/cascout (mxcascout) | 0.06 * | 6.74 f | (4) |
| GR4_GC0_L5_LE0/um2/COMBOUT (icombout) | 0.09 * | 6.84 r | (4) |
| GR4_GC0_L5_LE0/COMBOUT (c1000_0029a) | 0.00 | 6.84 r | (4) |
| GR4_GC0_L5_LE2/LUTC (c1000_0381a) | 0.00 | 6.84 r | (4) |
| GR4_GC0_L5_LE2/um4/ltb (lt03b) | 0.40 * | 7.24 r | (4) |
| GR4_GC0_L5_LE2/um5/cascout (mxcascout) | 0.05 * | 7.28 r | (4) |
| GR4_GC0_L5_LE2/um6/dcout (c1110) | 0.00 * | 7.28 r | (4) |
| data arrival time | | 7.28 | (4) |
| Point | Incr | Path | Reference to Figure 11–15 |

*Table 11–4. Timing Report After the Place-and-Route Process (Part 2 of 2)*

| | | | |
|---|---|---|---|
| `clock clkx' (fall edge)` | `7.41` | `7.41` | |
| `clock network delay (propagated)` | `2.74` | `10.15` | `(5)` |
| `clock uncertainty` | `-0.25` | `9.90` | `(6)` |
| `GR4_GC0_L5_LE2/um6/clk (c1110)` | | `9.90 f` | |
| `library setup time` | `-0.20 *` | `9.70` | `(7)` |
| `data required time` | | `9.70` | |
| `data required time` | | `9.70` | |
| `data arrival time` | | `-7.28` | |
| `slack (MET)` | | `2.42` | |

The `GR12_GC0_L2_LE4/REGOUT` pin now has the loading on it reduced by the introduction of several levels of buffering (in this case, six levels of inverters). The inverters have instance names similar to `N1188_iv06_1_0`, and are of type `iv06`, as shown in the static timing analysis report. As a result, the original setup-time violation of -0.63 ns turned into a slack of +2.42 ns, meaning the setup-time violation is fixed. The circuit that the static timing analysis report shows is illustrated in Figure 11–15. The buffer tree (buffer) is shown as a single cell.

*Figure 11–15. Circuit Post Place-&-Route*



Placing the values from the static timing analysis report into the setup-time slack equation results in the following:

$t_{SU}$ Slack = Clock Period + Clock Delay – (Data Delay + Micro $t_{SU}$)

$t_{SU}$ Slack = 7.41 + (2.74 - 0.25) – (2.73 + 0.69 + 1.71 + 2.15 + 0.20)

$t_{SU}$ Slack = +2.42 ns

This result shows that there is positive slack for this path, meaning that there is now no setup-time violation.

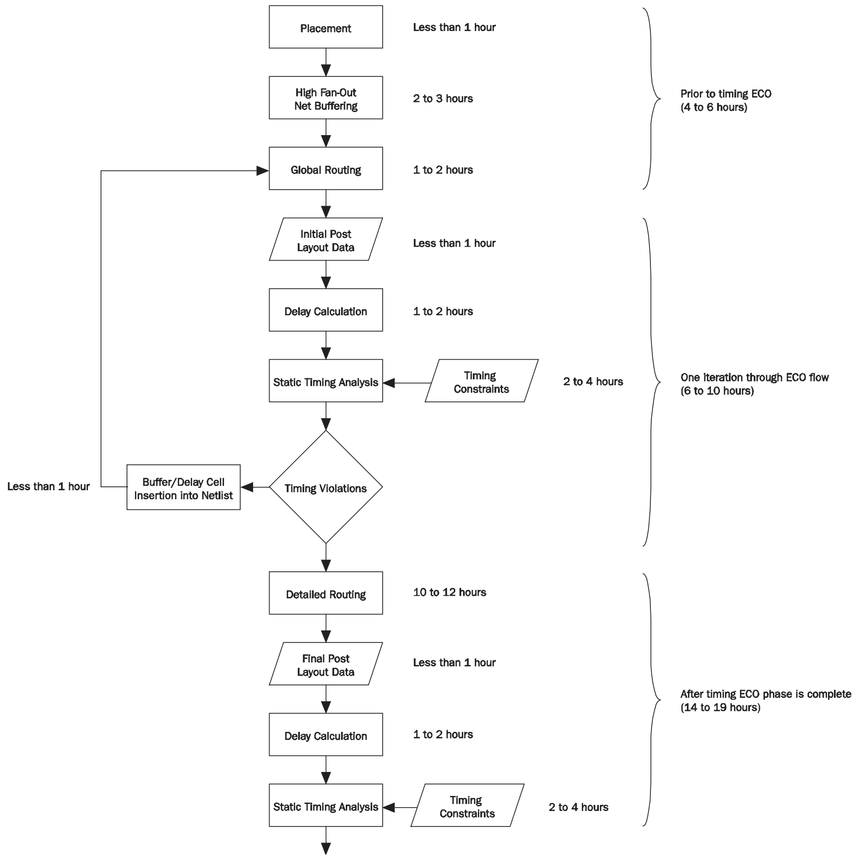
## Timing ECOs

In an ASIC, small incremental changes to a design database are termed engineering change orders (ECOs). In the HardCopy design flow, ECOs are performed after the initial post-layout timing data is available.

Static timing analysis is run on the design and a list of paths with timing violations are generated. The netlist is then automatically updated with changes that correct these timing violations (i.e., the addition of delay cells to fix hold-time violations). After the netlist update, the place-and-route database is updated to reflect the netlist changes. The impact on this database is minimized by maintaining all of the pre-existing placement-and-routing, and only changing the routing where new cells are inserted.

The parasitic (undesirable, but unavoidable) resistances and capacitances of the customized interconnect are extracted and then used in conjunction with the static timing analysis tool to re-check the timing of the design. Only a single iteration of this process is typically required to fix all timing violations. The entire ECO stage takes less than a day to complete. Figure 11–16 shows this flow in more detail, along with the typical duration of each stage.

*Figure 11–16. ECO Flow Diagram*



## Conclusion

When migrating a design from an FPGA implementation to a HardCopy implementation, it is critical to maintain performance even though all timing within the design does not remain exactly the same. These timing differences are inevitable. However, they are rendered inconsequential to the device's behavior in the end-system environment if the HardCopy device meets the system timing constraints. As a standard and automated part of the HardCopy design conversion process, this rendering is achieved through the exhaustive timing analysis that the design undergoes in conjunction with sophisticated timing-driven place-and-route. Static timing analysis can reveal timing violations that are then fixed automatically as part of the HardCopy design process.

# 12. Synplicity Amplify Physical Synthesis Support

qii52011-1.0

## Introduction

Synplicity has developed the Amplify Physical Optimizer physical synthesis software to help designers meet performance and time-to-market goals. You can use this software to create location assignments and optimize critical paths outside the Quartus® II software design environment. The Amplify Physical Optimizer design software, which runs on the Synplify Pro synthesis engine, creates a Tcl script with hard location assignments and LogicLock™ regions to control logic placement in the Quartus II software. Depending on the design, the Amplify Physical Optimizer software can improve Altera® device performance over Synplify Pro-compiled designs by reducing the number of logic levels and the interconnect delays in critical paths. Moreover, the Amplify Physical Optimizer software allows designers to compile multiple implementations in parallel to reduce optimization time.

For more information on the Synplify Pro software, see the *Synplicity Synplify & SynplifyPro Support* chapter in Volume 1 of the *Quartus II Handbook*.

This chapter explains the physical synthesis concepts, including an overview of the Amplify Physical Optimizer software and Quartus II flow.

## Software Requirements

The examples in this document were generated using the following software versions:

- Quartus II, version 4.0
- Amplify Physical Optimizer, version 3.2

## Amplify Physical Synthesis Concepts

The Amplify Physical Optimizer physical synthesis tool uses information about the interconnect architectures of Altera devices to reduce interconnect and logic delays in the critical paths. Timing-driven synthesis tools cannot accurately predict how place-and-route tools function; therefore, determining the real critical path with the synthesis tool is a difficult task.

Synthesis tools create technology-level netlist files that work with floorplans using place-and-route tools. Synthesis tools also define netlist names that are used in place-and-route, which means hard location assignments may not apply in the next revision of the resynthesized netlist as nodes names might have been renamed or removed.

Physical synthesis allows you to create floorplans at the register transfer level (RTL) of a design, giving you the ability to perform logic tunneling and replication. Physical synthesis also gives you the flexibility to make changes at the RTL level, allowing these changes to reflect in previously planned paths.

Physical synthesis uses knowledge of the FPGA device architecture to place paths into customized regions. This process will minimize interconnect delays as interconnect and placement information influences the synthesis process of the design.

When the Amplify Physical Optimizer software synthesizes a design, it creates a **.vqm** atom-netlist and Tcl script files, which are read by the Quartus II software. You can create a Quartus II project with the VQM netlist as the top-level module and source the Tcl script generated by the Amplify Physical Optimizer software. The Tcl script sets the design's device, timing constraints (Timing Driven Compilation [TDC] value, multicycle paths, and false paths), and any other constraints specified by the Amplify Physical Optimizer software. After you source the Tcl script, you can compile the design in the Quartus II software.

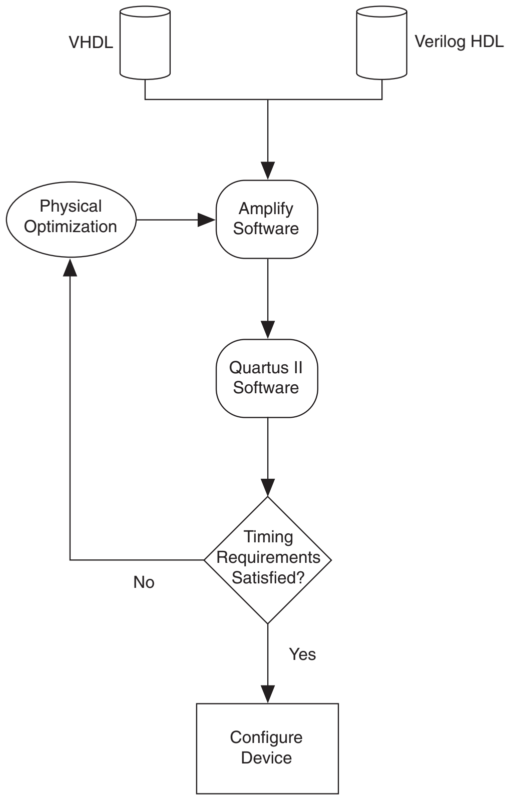
See "Forward Annotating Amplify Physical Optimizer Constraints into the Quartus II Software" on page 12–12 for more information on setting up a Quartus II project with Amplify Physical Optimizer Tcl script files.

After the Quartus II software compiles the design, the software performs a timing analysis on the design. The timing analysis reports all timing-related information for the design. If the design does not meet the timing requirements, you can use the timing analysis numbers as a reference when running the next iteration of physical synthesis through the Amplify Physical Optimizer software. This same timing analysis information is also reported in a file called <*project name*>**.tan.rpt** in the design directory.

## Amplify-to-Quartus II Flow

If timing requirements are not met with the Amplify Physical Optimizer flow, you should first place and route the design in the Quartus II software without physical constraints. After compilation, you can determine which critical paths should be optimized in the Amplify Physical Optimizer tool in the next iteration. Figure 12–1 shows the Amplify Physical Optimizer design flow.

*Figure 12–1. Software Design Flow*



## Initial Pass: No Physical Constraints

The initial iteration involves synthesizing the design in the Amplify Physical Optimizer software without physical constraints.

Before beginning the physical synthesis flow, run an initial pass in the Amplify Physical Optimizer without physical constraints. At the completion of every Quartus II compilation, the Quartus II Timing Analyzer performs a comprehensive static timing analysis on your design and reports your design's performance and any timing violations. If the design does not meet performance requirements after the first pass, additional passes can be made in the Amplify software.

### *Create New Implementations*

To set the Amplify Physical Optimizer software options, perform the following steps:

1. Compile the design with the **Resource Sharing** and **FSM Compiler** options selected and the **Frequency** setting specified in MHz. For optimal synthesis, the Amplify software includes the retiming, pipelining, and FSM Explorer options. For designs with multiple clocks, set the frequency of individual clocks with Synthesis Constraints Optimization Environment (SCOPE).

2. Select **New Implementation**. The **Options for Implementation** dialog box appears.

3. Specify the part, package, and speed grade of the targeted device in the **Device** tab.

4. Turn on the **Map Logic to Atoms** option in the **Device Mapping Options** dialog box.

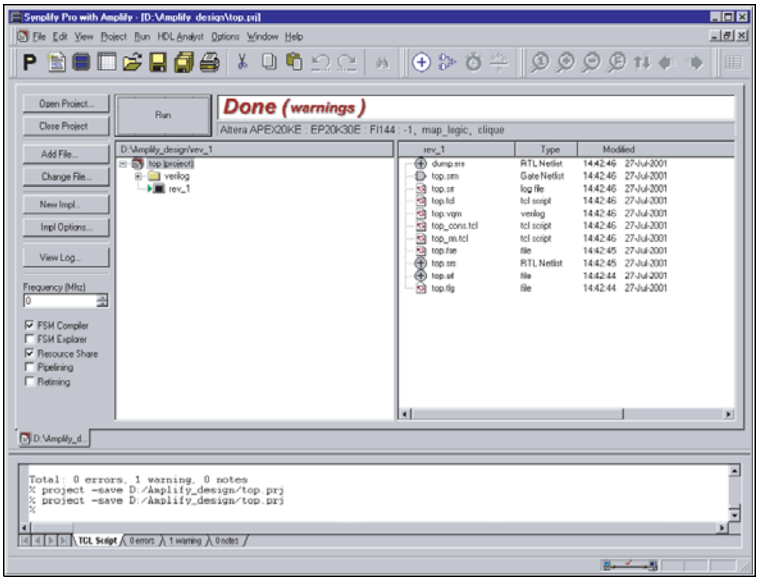
5. Turn off the **Disable I/O Insertion** and **Perform Cliquing** options.

6. Specify the name and directory in the **Implementation Results** tab. The result format should be VQM, and you should select **Optional Output Files** as the **Write Vendor Constraint File** option so that the software can generate the Tcl script containing the project constraints.

7. Specify the number of critical paths and the number of start and end points to report in the **Timing Report** tab. Figure 12–2 shows the main Amplify Physical Optimizer project window.

These steps create a directory where the results of this pass are recorded. Ensure that the Amplify Physical Optimizer software implementation options are set as described in the initial pass.

*Figure 12–2. Amplify Physical Optimizer Project Window*



Done (warnings)

Altera APEX20KE : EP20K30E : FI144 : -1, map_logic, clique

D:\Amplify_design\rev_1

- top (project)
  - verilog
  - rev_1

| rev_1 | Type | Modified |
|---|---|---|
| dump.srs | RTL Netlist | 14:42:46 27-Jul-2001 |
| top.srm | Gate Netlist | 14:42:46 27-Jul-2001 |
| top.srr | log file | 14:42:46 27-Jul-2001 |
| top.tcl | tcl script | 14:42:46 27-Jul-2001 |
| top.vqm | verilog | 14:42:46 27-Jul-2001 |
| top_cons.tcl | tcl script | 14:42:46 27-Jul-2001 |
| top_rm.tcl | tcl script | 14:42:46 27-Jul-2001 |
| top.fse | file | 14:42:45 27-Jul-2001 |
| top.srs | RTL Netlist | 14:42:45 27-Jul-2001 |
| top.srf | file | 14:42:44 27-Jul-2001 |
| top.tlg | file | 14:42:44 27-Jul-2001 |

```
Total: 0 errors, 1 warning, 0 notes
% project -save D:/Amplify_design/top.prj
% project -save D:/Amplify_design/top.prj
%
```

### Iterative Passes: Optimizing the Critical Paths

In the iterative passes, you optimize the design by placing logic in the device floorplan within the Amplify software. Amplify's floorplan is a high-level view of the device architecture. The floorplan view is dependent upon the target device family. When the Amplify Physical Optimizer re-optimizes the current critical path, additional critical paths may be created. Continue to add new constraints to the existing floorplan until it meets the performance requirements. The design may need several iterations to meet these performance requirements. Since optimizing critical paths involves trying different implementations, the creation of various Amplify project implementations will help in organizing the placement of logic in the floorplan.

## Using the Amplify Physical Optimizer Floorplans

When designs do not meet performance requirements with the initial pass through the Amplify Physical Optimizer software, you can create location assignments to reduce interconnect and logic delays to improve your design's performance.

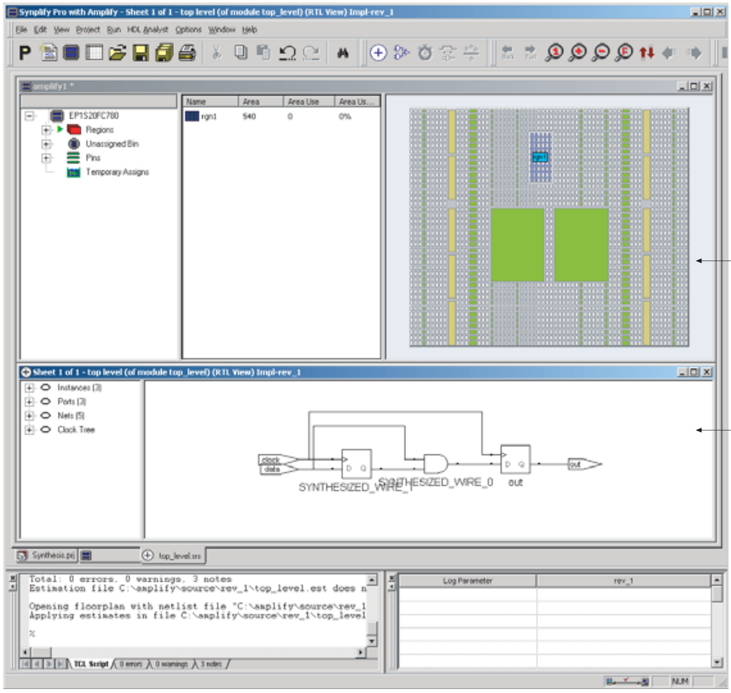
You must determine which paths to constrain based on the critical paths from the previous implementation. When Quartus II projects are launched with the Amplify Tcl script, the Quartus II software generates a <*project name*>**.tan. rpt** file that lists the critical paths for the design. You can then create custom structure regions for critical paths. After critical paths are implemented in a floorplan with the Amplify Physical Optimizer software, you must resynthesize the design. The software will then attempt to optimize the critical paths and reduce the number of logic levels. After the Amplify Physical Optimizer software resynthesizes the design, the Quartus II software must compile the new implementation. If the design does not meet timing requirements, perform another physical synthesis iteration.

Use the following steps to create a floorplan in the Amplify Physical Optimizer software:

1. Click the **New Physical Constraint File** icon at the top of the Amplify Physical Optimizer window.

2. Click **Yes** on the **Estimation Needed** dialog box; the floorplan window will appear (see Figure 12–3).

***Figure 12–3. Stratix 1S20 Floorplan in the Amplify Physical Optimizer Software***



| Name | Area | Area Use | Area Us... |
|---|---|---|---|
| rgn1 | 540 | 0 | 0% |

```
Total: 0 errors, 0 warnings, 3 notes
Estimation file C:\amplify\source\rev_1\top_level.est does n

Opening floorplan with netlist file "C:\amplify\source\rev_1
Applying estimates in file C:\amplify\source\rev_1\top_level

%
```

The floorplan view is located at the top of the screen and the RTL view is at the bottom of the screen.

You can specify modules or individual paths in the Amplify Physical Optimizer software. Using modules can quickly resolve timing problems.

Use the following steps in the software to create a floorplan module:

1. Create a region in the Amplify Physical Optimizer device floorplan window and select the module in the RTL view of the design.

2. Drag the module to the new region. The software will then report the utilization of the region.

3. Resynthesize the design in the software to reoptimize the critical path after the modules have location constraints.

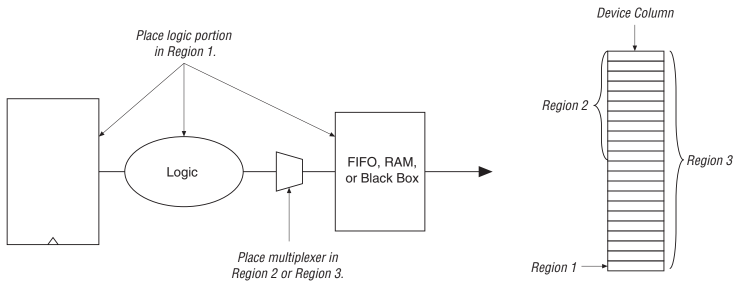
4. Write out the placement constraints into the VQM netlist and the Tcl script.

Repeat the above procedure to create as many regions as required.

## Multiplexers

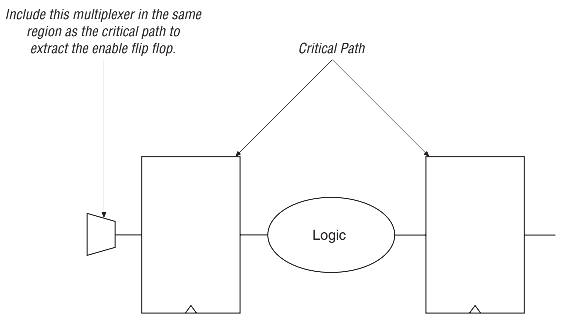
To create a floorplan for critical paths with one or more multiplexers, create multiple regions and assign the multiplexer to one region and the logic to another. Figure 12–4 shows placing critical paths with multiplexers.

*Figure 12–4. Placing Critical Paths with Multiplexers*


Place logic portion in Region 1.
Logic
FIFO, RAM, or Black Box
Place multiplexer in Region 2 or Region 3.
Device Column
Region 2
Region 3
Region 1

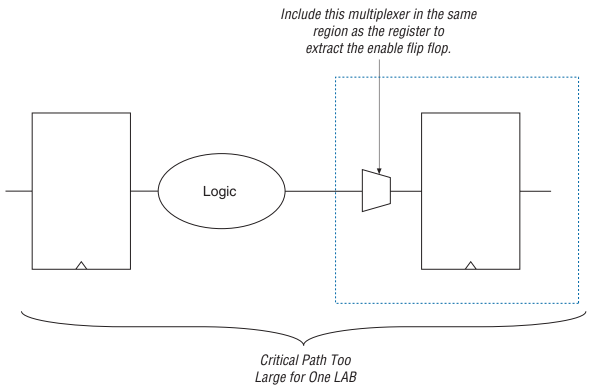
If the critical path contains a multiplexer feeding a register, create a region and place the multiplexer along with the entire critical path in the region. See Figure 12–5.

*Figure 12–5. Critical Paths with Multiplexers Feeding Registers*


Include this multiplexer in the same region as the critical path to extract the enable flip flop.
Critical Path
Logic

If the critical path is too large for the region, divide the critical path and ensure that the multiplexer and register are in the same region. Figure 12–6 shows large critical paths with multiplexers feeding registers.

*Figure 12–6. Large Critical Paths with Multiplexers Feeding Registers*


Include this multiplexer in the same region as the register to extract the enable flip flop.
Logic
Critical Path Too Large for One LAB

## Independent Paths

Designs may have two or more independent critical paths. To create an independent path in the Amplify Physical Optimizer software, follow the steps below:

1. Create a region and assign the first critical path to that region.
2. Create another region, leaving one MegaLAB structure between the first and second regions.

3. Assign the second critical path to the second region.

## Feedback Paths

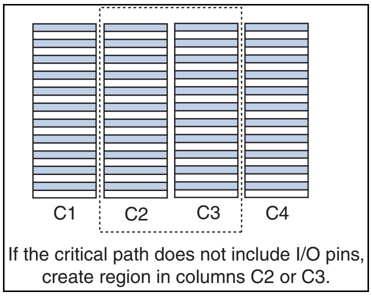
If critical paths have the same start and end points, follow the steps below in the Amplify Physical Optimizer software (see Figure 12–7):

1. Select the register and instance not directly connected to the register.

2. Select **Filter Schematic** twice (right-click menu).

3. Highlight the line leading out of the register and either press **P** or right-click the line. Select **Expand Paths**. Assign this logic to a region.

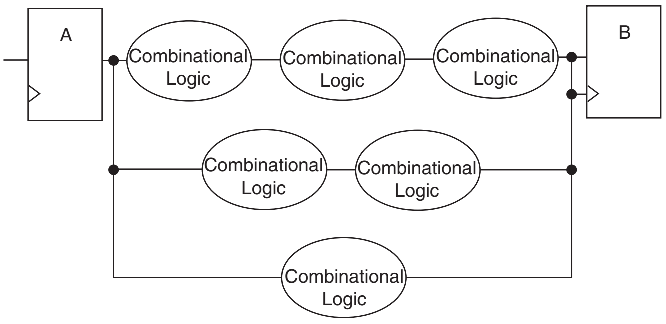
*Figure 12–7. Critical Paths with the Same Starting or Ending Points*


C1
C2
C3
C4
If the critical path does not include I/O pins, create region in columns C2 or C3.

## Starting and Ending Points

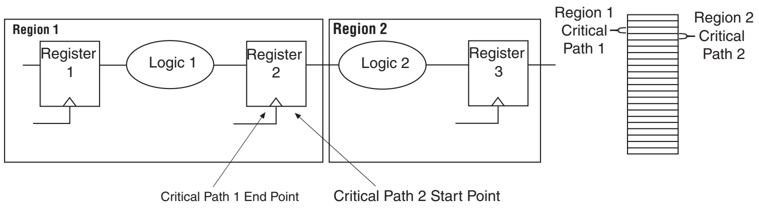
Figure 12–8 shows a critical path that has multiple starting and ending points. Use **Find** to display all the starting and ending points in the RTL view in Amplify. Expand the paths between those points. If there is unrelated logic between the multiple starting points and ending points, assign the starting points and ending points to the same region. Similarly, if there is unrelated logic between starting points and multiple ending points, assign the starting points and ending points to the same region.

*Figure 12–8. Critical Paths with Multiple Starting or Ending Points*



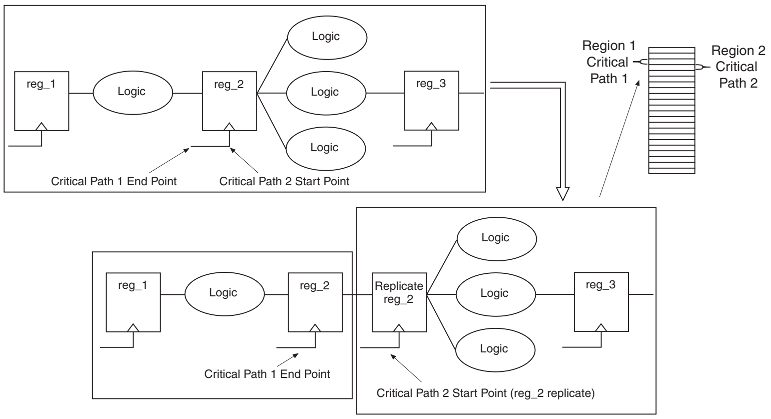
If the two critical paths share a register at the starting or ending point, assign one critical path to one region, and assign the other critical path to an adjacent region. Figure 12–9 shows two critical paths that share a register.

*Figure 12–9. Two Critical Paths Sharing a Register*



If the fanout is on the shared region, replicate the register and assign both registers to two regions (see Figure 12–10). This is done by dragging the same register to the required regions. Entities and nodes are also replicated by performing the same procedure.

*Figure 12–10. Fanout on a Shared Region*



## Utilization

Designs with device utilizations of 90% or higher may have difficulties during fitting in the Quartus II software. If the device has several finite state machines, you should implement the state machines with sequential encoding, as opposed to one-hot encoding.

To check area utilization, check the Amplify Physical Optimizer **log** file and .**srr** file for region utilization, after the mapping stage is complete. You can also update the utilization estimates by using the estimate region feature by selecting **Estimate Area** (Run menu).

## Detailed Floorplans

If the critical path does not meet timing requirements after physical optimization, you can create new regions to achieve timing closure. It is recommended that regions do not overlap. Regions should either be entirely contained in another region or remain entirely outside of it. Select the logic requiring optimization from the existing region. Deselect the logic and assign it to the new region. Run the Amplify Physical Optimizer software on the design with the modified physical constraints. Then place and route the design.

## Forward Annotating Amplify Physical Optimizer Constraints into the Quartus II Software

The Amplify Physical Optimizer software simplifies the forward annotating of both timing and location constraints into the Quartus II software through the generation of three Tcl scripts. At the completion of a physical synthesis run, in the Amplify Physical Optimizer software, the following Tcl scripts are generated:

- *<project name>*_**cons.tcl**
- *<project name>*.**tcl**
- *<project name>*_**rm.tcl**

Table 12–1 provides a description of each script's purpose.

*Table 12–1. Amplify Physical Optimizer Tcl Script Description*

| Tcl File | Description |
|---|---|
| *<project name>*_**cons** | This Tcl script will create and compile a Quartus II project. The *<project name>*.**tcl** will automatically be sourced when this script is sourced. |
| *<project name>* | This script contains forward annotation of constraint information including clock frequency, duty cycle, location, etc. |
| *<project name>*_**rm** | This script removes any previous constraints from the project. The removed constrainst is saved in *<project name>*_**prev.tcl** |

To forward annotate Amplify Physical Optimizer's constraints into the Quartus II software you must use **quartus_cmd**. The **quartus_cmd** command must be used as Amplify Physical Optimizer's Tcl scripts are not compatible with **quartus_sh**. The following command will execute the *<project name>*_**cons**, which will create a Quartus II project with all Amplify Physical Optimizer constraints forward annotated, and will perform a compilation.

```
<commnd prompt>quartus_cmd my_project_cons.tcl ↵
```

You must execute the *<project name>*_**cons.tcl** first.

After compilation, you may customize the project either in the Quartus II GUI or sourcing a custom Tcl script.

See the *Tcl Scripting* chapter in Volume 2 of the *Quartus II Handbook* for more information on creating and understanding Tcl scripts in the Quartus II software.

## Altera Megafunctions Using the MegaWizard Plug-In Manager with the Amplify Software

When you use the Quartus II MegaWizard® Plug-In Manager to set up and parameterize a megafunction, it creates either a VHDL or Verilog HDL wrapper file. This file instantiates the megafunction (a black box methodology) or, for some megafunctions, generates a fully synthesizeable netlist for improved results with EDA synthesis tools such as Synplify (a clear box methodology).

### *Clear Box Methodology*

The MegaWizard Plug-In Manager-generated fully synthesizeable netlist is referred to as a clear box methodology because the Amplify Physical Optimizer software can "see" into the megafunction file. The clear box feature enables the synthesis tool to report more accurate timing estimates and take better advantage of timing driven optimization.

This clear box can be turned on by checking the **Generate Clearbox body (for EDA tools only)** option in the **MegaWizard Plug-In Manager** (Tools menu) for certain megafunctions. If this option does not appear, then clear box models are not supported for the selected megafunction. Turning on this option will cause the MegaWizard Plug-In Manager to generate a synthesizable clear box netlist instead of the megafunction wrapper file described in "Black Box Methodology" on page 12–14.

#### Using MegaWizard Plug-In Manager-generated Verilog HDL Files for Clear Box Megafunction Instantiation

If you check the <*output file*>**_inst.v** option on the last page of the wizard, the MegaWizard Plug-In Manager generates a Verilog HDL instantiation template file for use in your Synplify design. This file can help you instantiate the megafunction clear box netlist file, <*output file*>**.v**, in your top-level design. Include the megafunction clear box netlist file in your Amplify Physical Optimizer project and the information gets passed to the Quartus II software in the Amplify Physical Optimizer-generated VQM output file.

#### Using MegaWizard Plug-In Manager-generated VHDL Files for Clear Box Megafunction Instantiation

If you check the <*output file*>**.cmp** and <*output file*>**_inst.vhd** options on the last page of the wizard, the MegaWizard Plug-In Manager generates a VHDL component declaration file and a VHDL instantiation template file for use in your design. These files help to instantiate the megafunction clear box netlist file, <*output file*>**.vhd**, in your top-level design. Include the megafunction clear box netlist file in your Amplify Physical Optimizer project and the information gets passed to the Quartus II software in the Amplify Physical Optimizer-generated VQM output file.

### *Black Box Methodology*

The MegaWizard Plug-In Manager-generated wrapper file is referred to as a black-box methodology because the megafunction is treated as a "black box" in the Amplify Physical Optimizer software. The black box wrapper file is generated by default in the **MegaWizard Plug-In Manager** (Tools menu) and is available for all megafunctions.

The black-box methodology does not allow the synthesis tool any visibility into the function module thus not taking full advantage of the synthesis tool's timing driven optimization. For better timing optimization, especially if the black box does not have registered inputs and outputs, add timing models to black boxes.

For more information on instantiating MegaWizard Plug-In Manager modules or black boxes see the *Synplicity Synplify & SynplifyPro Support* chapter in Volume 1 of the *Quartus II Handbook*.

## Conclusion

Physical synthesis uses improved delay estimation to optimize critical paths. The Amplify Physical Optimizer software uses the hierarchical structure of logic and interconnect in Altera devices so that designers can direct a critical path to be placed into several well-defined blocks. The Amplify Physical Optimizer-to-Quartus II software flow is one of the steps to solving the problem of achieving timing closure through physical synthesis.

# Index